University of the West of England
BRISTOL

Enhanced Online Programming for Industrial Robots

Christian Kohrt

A thesis submitted in partial fulfilment of the requirements of the
University of the West of England, Bristol, UK
for the degree of Doctor of Philosophy

This research programme was carried out in collaboration with the University of Applied Sciences Landshut, Landshut, Germany

Faculty of Environment and Technology
University of the West of England, Bristol

July 2013

Dedicated to my wife Erica.

Abstract

The use of robots and automation levels in the industrial sector is expected to grow, and is driven by the on-going need for lower costs and enhanced productivity. The manufacturing industry continues to seek ways of realizing enhanced production, and the programming of articulated production robots has been identified as a major area for improvement. However, realizing this automation level increase requires capable programming and control technologies. Many industries employ offline-programming which operates within a manually controlled and specific work environment. This is especially true within the high-volume automotive industry, particularly in high-speed assembly and component handling. For small-batch manufacturing and small to medium-sized enterprises, online programming continues to play an important role, but the complexity of programming remains a major obstacle for automation using industrial robots. Scenarios that rely on manual data input based on real world obstructions require that entire production systems cease for significant time periods while data is being manipulated, leading to financial losses. The application of simulation tools generate discrete portions of the total robot trajectories, while requiring manual inputs to link paths associated with different activities. Human input is also required to correct inaccuracies and errors resulting from unknowns and falsehoods in the environment. This study developed a new supported online robot programming approach, which is implemented as a robot control program. By applying online and offline programming in addition to appropriate manual robot control techniques, disadvantages such as manual pre-processing times and production downtimes have been either reduced or completely eliminated. The industrial requirements were evaluated considering modern manufacturing aspects. A cell-based Voronoi generation algorithm within a probabilistic world model has been introduced, together with a trajectory planner and an appropriate human machine interface. The robot programs so achieved are comparable to manually programmed robot programs and the results for a Mitsubishi RV-2AJ five-axis industrial robot are presented. Automated workspace analysis techniques and trajectory smoothing are used to accomplish this. The new robot control program considers the working production environment as a single and complete workspace. Non-productive time is required, but unlike previously

reported approaches, this is achieved automatically and in a timely manner. As such, the actual cell-learning time is minimal.

Acknowledgment

I thank my director of studies, Dr. Richard Stamp, and my supervisors, Dr. Antony Pipe and Dr. Janice Kiely at the University of the West of England at Bristol. I also thank my supervisor Dr. Gudrun Schiedermeier at the University of Applied Sciences Landshut.

Dr. Richard Stamp guided me through this work. I have learnt a great deal from him about scientific working and writing. I am glad to have had a director of studies who was willing to take a personal interest in his student.

Dr. Gudrun Schiedermeier made my PhD studies at the University of Applied Sciences Landshut possible. I thank her for her guidance, her constant interest in my PhD studies and for providing the robotics laboratory including its special equipment required for the experiments; without that, I would not have been able to accomplish this work.

Dr. Anthony Pipe gave me his advice, valuable comments and constructive discussions. Dr. Janice Kiely was always available for discussions during my PhD studies.

I also thank the persons at the University of the West of England who offered me the opportunity to study for a PhD within the Faculty of Environment and Technology, and who provided me with the academic support necessary to complete this work successfully. In particular, I thank Matthew Guppy for his advice during the studies.

I am thankful to have had the opportunity to conduct my research at the University of Applied Sciences, Landshut. I am grateful to the former President, Prof. Dr. Erwin Blum, and to the former vice President, Prof. Dr. Helmuth Gesch, to support this work.

I am grateful to BMW AG Munich and Hans-Joachim Neubauer for setting the idea for this work and also to Robtec GmbH for giving me insight to professional robot programming and for their support and experience in the field of robotics.

I also thank my mother who always supported me during all these years. I also thank my friends listening to me and motivating me during this part of my life.

Finally, I thank my wife, to whom I dedicate this thesis, for her patience, encouragement and support during the probably hardest time of this work.

Travelling costs for this work were partly funded by the “Bayerische Forschungsstiftung” of the Bavarian Government, Germany.

Contents

I. Glossary

Configuration space	A robot with n degrees of freedom is usually a manifold of dimension n. This manifold is called the configuration space of the robot, and is considered as a state-space.
Kinematics	The study of the motion of bodies without reference to mass or force.
Manufacture assistant	Manufacture assistants are clever systems, which help the worker to accomplish their task.
Mission	A mission defines a path-planning task with application information and locations.
Octree	An octree is a tree data structure in which each internal node has up to eight children. Octrees are most often used to partition a three dimensional space by recursively subdividing it into eight octants.
Online programming	Programming of robots by the help of the teachpendant or other robot control devices with the need of the real robot.
Offline programming	Programming of robots by the help of a simulation system without the need of the real robot.
Quadtree	A quadtree is a tree data structure in which each internal node has up to four children. Quadtrees are most often used for partitioning by recursively subdividing it into four quadrants.
Trajectory	The line or curve described by an object moving through space.
World model	The in-memory environment representation based on sensory input or external information source.

II. List of Abbreviations

Acronym	Definition
AD*	Anytime Dynamic A*
BSP	Binary Space Partitioning
CAD	Computer-Aided Design
CASE	Computer-Aided Software Engineering
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DDE	Dynamic Data Exchange
DH	Denavit Hartenberg
DLL	Dynamic Link Library
DSP	digital signal processor
DXF	Drawing Exchange Format
EMF	Eclipse Modelling Framework
EN	Elastic Net
ES	Evolution Strategy
FPGA	Field Programmable Gate Array
GPP	General purpose processor
GUI	Graphical User Interface
HMI	Human Machine Interface
HSV	Hue, Saturation, and Value Colour Space
ICE	Internet Communication Engine
IP	Internet Protocol
JET	Java Emitter Template
JNI	Java Native Interface
JVM	Java Virtual Machine
LED	Light Emitting Diode
MDA	Model Driven Architecture
MDG	Model Driven Generation
NARC	New Architecture Robot Controller
ODE	Open Dynamics Engine
OMG	Object Management Group
PC	Personal Computer
PLC	Programmable Logic Controller
RBF	Radial Basis Function
RFID	Radio-Frequency Identification

RGB	Red, Green, and Blue Colour Space
ROOM	Real Time Object Oriented Modelling
SMA	Simple Moving Average
SWT	Standard Widget Toolkit
TCP	Transmission Control Protocol
TCP	Tool Centre Point
TD	Temporal Difference
TSP	Traveling Salesman Problem
UDP	User Datagram Protocol
UK	United Kingdom
UML	Unified Modelling Language
XML	Extensible Mark-up Language
YCbCr	YCbCr Colour Space

III. List of Figures

IV. List of Tables

V. List of Listings

1 Introduction

Automated production is essential for industries, including the automotive, electronics, plastics and metal products industries. Automation can be realized by the introduction of industrial robots, which are efficient in terms of speed, flexibility and reliability. At the end of 2010, the worldwide stock of operational industrial robots numbered between 1,030,000 and 1,305,000 units. In 2010, the worldwide market value for robot systems was estimated to be US$5.7 billion (International Federation of Robotics, 2011). The use of robots and automation levels in the industrial sector is expected to continue to grow in future, and will be driven by the on-going need for enhanced productivity. Manufacturing industries continue to be faced with shortened product life cycles, increasing dynamics of innovation, and continuing diversification of their product ranges. Simultaneously, they have to lower the costs per item and the costs of hiring skilled workers. The dynamic requirement profile of production must be addressed in order to ensure compliance with high quality standards as well as time and cost efficiency. Industrial robots are capable of meeting the emerging needs with regards to flexibility and productivity, but the use of these robots remains difficult, time consuming and expensive. There are particularly high requirements in terms of capable robot programming and control technologies. Industries are strongly motivated to improve efficiency and effectiveness of robot programming and control. Production engineering and automation have been developed to an advanced level, and further improvements may be reached using new approaches to the programming of robots. Therefore, the vision of the industry is to realize a completely automated production process without any manual intervention from the product planning stage to the manufactured product. This vision has not been achieved, and even under ideal conditions of production machines, trajectory planning remains difficult. With regard to errors that result from unknowns and falsehoods in the environment, realizing this vision becomes even more unlikely. Robot programming for a specific application may require months, while the cycle time of the application is executed in only a few minutes or hours. Therefore, robotic automation requires that significant investments be made before commencing production.

While conventional online programming is simple, it is only useful for programming an uncomplicated application with simple geometries of the workpieces. Those workpieces are required to be present within the robot cell. Highly skilled operators are needed to execute this task, while the production is halted during online programming. Any scenario that relies on manual data input based on real world obstructions requires that the complete

production system be put offline and out of production for significant periods of time while the data are manipulated, e.g. upload of or programming of robot programs. This leads to production downtimes and financial loses. It also places a lot of pressure on the operators, which may have an impact on the quality of the created programs. Once the programs are created, it is difficult to make amendments. Nevertheless, conventional online programming is widely used because of its intuitiveness and low initial cost. Advances in online programming simplify the control of the robots, such as Master-Slave programming and demonstrational programming (Demiris and Billard, 2007), but have not yet led to crucial improvements.

On the other hand, offline programming reduces the production downtime, creating the robot programs beforehand with a simulation system (Kain *et al.*, 2008, Maletzki *et al.*, 2008). Many industries employ offline programming within a manually controlled and specified work environment. This is especially true within the high-volume automotive industry, particularly when it is related to high-speed assembly and component handling. Therefore, it is widely accepted in high volume manufacture industries with proven efficiencies and cost effective strategies. Its strength is in the programming of complex applications, and when compared to online programming, it is more reliable and allows the re-use of robot programs with ease. Because it relies on the modelling of the production cell, additional manual modifications of the generated robot programs are necessary to meet the accuracy requirements in production. Inaccuracies and errors resulting from unknowns and falsehoods in the environment have to be altered manually. The simulation of the production cell verifies the virtually programmed production process; subsequently, the robot program may be generated and uploaded to the real robot cell. An online robot programmer verifies and eventually modifies the programs to guarantee its actual function, but this task can be time consuming. Manual modifications may be made, and may involve a complete re-programming of the simulated robot. Possible reasons for this include inaccuracies of the simulation data, last minute changes in the production process, and a misunderstanding of the robot programs. The offline program developer and the online robot operator are not usually the same person, and they tend to have different skills, which may also be a source of misunderstanding. However, this is expensive, requires skilled workers, and depends on an accurate modelling of the realistic scenario, which is often not possible.

Investigations have been carried out with the aim being to optimize the robot-programming methodology for industrial high-volume and small-batch manufacturing. To address this aim, two key aspects have been identified that are different but related. First, the reduction of financial investments required that an analysis be made of the current robot programming approaches in order to explore all possible cost reduction options. Because production cost is measured in terms of the product cost, the production volume is an important feature that highlights the difference in the requirements for small-batch and high-volume production. In particular, in the area of small-batch production, the investments required for offline programming are prohibitive, and attention has been turned to developing approaches to online robot programming. High-volume production would also benefit from a change to the online robot programming approach, given that production downtimes are within the current range, and that the functionality of the current offline approach is still supported. A new online robot programming approach has been analysed, with the focus being on the fulfilment of requirements for both production volumes. This has required significant investigations into existing approaches to robot programming, including assisted interaction with the operator to help less experienced operators use this system. An enhanced online robot programming support system has been adopted for this task.

As the second key aspect, online robot programming is very demanding for a requirements-driven trajectory-planning algorithm. This also includes the ability to handle inaccurate information (which may be obtained by sensors) and the environment, as well as pre-existing information. Research has been undertaken to develop a trajectory-planning algorithm and to fuse inaccurate information into an in-memory occupancy grid to represent the production environment. It is understood that the development of large software systems requires a modern software development approach to integrate the entire system that consist of robot control devices, sensors and software components. Research has been carried out to implement a model-driven code generation toolchain.

The research started with a comparison of robot programming approaches and the exploration of important requirements of the production industry, focusing on the employment of robots. The chosen robot-programming approach, including the associated requirements, is summarized in Chapter 5. The realization of the robot-programming software application requires robot control capabilities for the articulated and mobile robots used. Robot modelling with the ‘Denavit and Hartenberg’ formulation is applied

throughout this study, which enables the execution of forward and inverse calculations between the robot coordinate system and the world coordinate system. Robot control and modelling are illustrated in Section 5.7. The environment in which the robot operates is stored in a so-called world model, which is an in-memory occupancy grid. Information, such as sensor data and pre-existing models, is fused into the grid to obtain coherent data. The world model and information fusion are illustrated in Chapter 6. Based on robot control and the in-memory occupancy grid, a trajectory-planning algorithm that supports the chosen robot programming approach has been introduced. It also includes path finding, trajectory generation and automated robot program creation that is ready to be uploaded to the real system. The results are stated in Chapter 7. The implementation of the system requires the incorporation of many software and hardware components. A modern software development toolchain has been analysed and implemented to support the development of the enhanced online robot programming support system. Chapter 9 addresses the development toolchain.

Chapter 2 presents a review of the whole research activity covered in this investigation, and it helps to collate important results that address the original aims of the study. The conclusions made are itemised in Chapter 12. Sample source code is presented in Appendix I. The investigation has highlighted particular aspects, many of which were unknown at the beginning of the research described herein, and which could themselves form the basis for additional studies. These are stated in Chapter 13.

Significant findings of this research have already been published, and Appendix A presents a summary of these. The papers themselves are appended to this thesis. One international journal paper were subject to peer-review and have now been published. Selected findings have been presented by the author at international conferences, and five papers were published between 2006 and 2012. The research was directed to the enhancement of the car production at BMW AG, Munich, and discussed with the robot programming company Robtec GmbH. Currently, the processes and techniques developed are intended to be scientifically and commercially used in close co-operation with the University of Applied Sciences Landshut, Germany. The system will be permanently installed at the lab of the University of Applied Sciences Landshut, and further improvements are planned for future study.

The result of this work leads to an enhanced online robot programming system for robot arms. The proposed system will be a novel, rapid, convenient and flexible method to program industrial robots. Programming within the real environment becomes possible and will decrease offline programming time and render offline simulation systems unnecessary when physical production parts and fixtures are to hand either as real objects or as Computer-Aided Design (CAD) data.

The system will have greatest benefit within the production industry, however its use will not be restricted to this application area. It could also assist in areas as diverse as home robots, surgery and health care assistant machines.

2 Literature Survey

2.1 Industrial Manufacture

Based on the product lifetime and production volume, Hägele *et al.* (2001) classifies industrial manufacture into conventional, pre-configured, decentralized and assisted manufacturing.

Conventional manufacturing lines can be effectively employed when the product lifetime and the production volume are known beforehand. The degree of automation is based on the technological feasibility and cost of each operation.

Pre-configured robot work cells produced in medium numbers at low cost for standard manufacturing processes such as welding, painting and palletising may even be cost-effective when operated below full capacity (Westkämper *et al.*, 1999).

Modern decentralized paradigms restructure the production into a network of configurable working cells, which are connected such that they achieve flexibility in terms of changing products. These production cells are often called 'holonic' or 'bionic' manufacturing systems (Westkämper *et al.*, 1999).

The greatest flexibility is required in assisted manufacturing co-operating with the worker in handling, transporting, machining and assembly tasks (Hägele *et al.*, 2002, Kristensen *et al.*, 2001, Kristensen *et al.*, 2002, Prassler *et al.*, 2002, Stopp *et al.*, 2002, Thiemermann, 2005, Wösch *et al.*, 2002).

Motion planning is required for any of the above-mentioned types of manufacturing. Assisted manufacturing is based on reactive motion planning, whereas conventional, pre-configured and decentralized manufacturing processes are often based on fixed robot programs, and are further explained in Subsection 2.1.1. The limitation of fixed robot programs is reached when the task execution requires a level of perception, dexterity and decision making which cannot be met technically in a cost effective or robust way. To achieve better productivity, assisted manufacturing relies on the co-operation between the robot and the operator. These robots can be considered to be manufacturing assistants, and are further described in Subsection 2.1.2.

2.1.1 Robot Programming

Online and offline robot programming approaches have been established in practical industrial applications. Offline programming is based on a model of a complete robot working-cell, and it shifts the programming tasks from the robot operator to the software

engineer in the office. It has its strength in the programming of complex systems, and it has been proven to be more efficient and cost-effective for production with large volumes.

Pan *et al.* (2010) reviews modern robot programming approaches and summarizes sensor-assisted online programming and offline programming approaches. Advancements in online programming have led to a simplification of the control of the robots, such as Master-Slave programming and demonstrational programming, but they have not yet led to any significant improvements. Offline programming reduces the production downtimes by creating the robot programs beforehand with a simulation system (Kain *et al.*, 2008, Maletzki *et al.*, 2008, Pan *et al.*, 2010). Nevertheless, the calibration phase and the offline programming phase are still expensive, and result in significant programming effort, large capital investment and long delivery times (Pan *et al.*, 2010).

Online Programming

Online programming is carried out by skilled operators in the robot working-cell, and requires that the production be offline. The robot is guided through the desired path using a teach pendant to record specific points into the robot controller, which is further utilized for the manual creation of movement commands (Pan *et al.*, 2010). The robot operator maintains the robot programs including the positions and orientations with a teach pendant. Many coordinate systems like the world, tool and work piece coordinate systems have to be tracked by the operator. This task is difficult and not intuitive. Guiding the robot accurately through the working space without any collisions is usually a very difficult and time-consuming task, especially when the work piece has a complex geometry or the process itself is very complicated. The created robot program often lacks flexibility and reusability. Online robot programming remains the choice for low and medium volume production. Currently, more intuitive human machine interfaces and sensory interfaces are being researched to reduce the reliance on the operator skill, and to improve automation (Bjorn Solvang, 2008, Gonzalez-Galvan *et al.*, 2007, Hu *et al.*, 2007, Hui *et al.*, 2006, Myoung Hwan and Woo Won, 2001, Nicholson, 2005, Pan and Zhang, Schraft and Meyer, 2006, Sugita *et al.*, 2004, Takarics *et al.*, 2008). Pan *et al.* (2010) highlight that only the research outcomes from Hui *et al.* (2006) have led to the development of a commercial tool. Pan *et al.* also identified the limitation to specific setups as being one of the main reasons for the failure to commercialize the remaining approaches. In particular, small-, medium- and high-volume manufacture may benefit from enhanced sensor-assisted online programming.

Offline Programming

High-volume manufacture utilises offline programming to simulate and generate robot programs with specialized simulation software. The software engineer evaluates the reachability, fine-tunes properties of robot movements, and handles the process-related information before generating a program that can be downloaded to the robot. The actual robot is not required for programming, minimizing the production downtime. Usually, robot programs are developed at the beginning of the product development and production cycle. However, a simulation and programming phase executed by skilled engineers is time consuming and requires specialized and expensive simulation software. Thus, small- and medium-volume manufacture does not benefit from this technology (Pan *et al.*, 2010), whereas large companies, for example BMW AG in the automotive industry, apply offline programming as a standard process. High volume production justifies the costly simulation and programming phase in order to assure high quality production.

Offline programming incorporates models of the work pieces, the robots and the environment. While the robot model is usually delivered by the manufacturer, the work pieces and the environment have to be created manually or, for example, with laser scanning (Bi and Sherman, 2007).

Successively, application locations have to be created in a manual or automatic fashion. The offline programming tools often provide functions that are used to extract features, e.g. edges and corners, and which can be utilized to define the required robot application task. Additional aspects related to the application type, e.g. equipment control, have to be considered in order to produce the robot programs. It becomes evident that the software engineer also requires skills in the specific application type to produce high-quality robot programs. The approach proposed by Pires *et al.* (2004) attempts to extract robot motion information from the models automatically.

The creation of the trajectory connecting all application locations and paths is often executed manually. Automatic solutions are usually not provided by the vendor of the software package, and have to be incorporated by third party tools or developments. Connecting large amounts of application locations and paths may result in the well-known ‘travelling salesman’ problem, which may be solved using various approaches (Al-Mulhem and Al-Maghrabi, 1997, Fritzke and Wilke, 1991, Kim *et al.*, 2002).

The entire production cycle, or parts of it, can be simulated after robot program creation to verify the production process without the physical production system (Heim, 1999). Successively, the robot program can be uploaded and executed within the real production environment. Extensions have been developed, for example by Wenrui and Kampker (1999), to enhance the simulation and offline programming process. In practice, inaccuracies and errors resulting from unknowns and falsehoods in the environment have to be altered manually using the real production system.

2.1.2 Manufacture Assistants

Hägele *et al.* (2002) describes manufacture assistants as clever systems which help the worker to accomplish their task. However, high-volume manufacturing is presently fully automated, and human-robot interaction is not always required. The high level of automation is attained through the robot-programming task, which is executed once during installation of the production cell. Thus, assisted robot-programming produces a robot program and is distinguished from manufacture assistants. Helms (2002) proposes a 'human centred automation' to improve the usability of robots, with the aim being to combine the sensory skill, the knowledge and the skilfulness of the worker with the advantages of the robot, e.g. strength, endurance, speed and accuracy. Manufacturing assistants represent a generalization of industrial robots characterized by their advanced level of interaction. Nevertheless, the human-machine interface and the underlying technology realizing the assistance functionality also play an important role.

The human-machine co-operation has been addressed by numerous researchers, and is viewed as a prime research topic by the robotics community (EURON, 2012). Haegele *et al.* (2001) also state the typical requirements for the human-machine interface. The human and the robot assistant should co-operate and safely interact, even in complex situations. This implies that the assistant understands the human intent through natural speech, haptic or graphical interfaces. In addition, effective cooperation depends on the recognition and perception of typical production environments, as well as on the understanding of tasks within their own contexts. Effective assistance requires the technical intelligence of the robot as well as the knowledge and skill transfer between the human and the robot.

A typical example of learning is programming by demonstration (Pan *et al.*, 2010). During human-machine interaction, motions have to be planned and quickly co-ordinated. For motions without physical human contact, skills such as avoiding obstacles,

approaching humans, and presenting objects have to be performed. In the more difficult case of physical contact with the human, typical skills would comprise compliant motion, anthropomorphic grasping and manipulation. A suitable safety concept has to account for the integrity of the system just as it must account for the integrity of its surroundings. External events affecting the proper function of the system and internal error conditions have to be identified beforehand and classified according to their inherent risk factors.

2.2 Modelling of the Robot

The robot manipulator is an essential tool for the development of automated manufacturing. A robot manipulator, also known as robot-arm, is a non-linear system with n rotatory joints (Craig, 2003). The motion equations of the robots are coupled, non-linear high-order differential equations, and the expenditure required for their evaluation is generally very high. Either procedures for the evaluation of the motion equations work numerically, or the motion equations are determined in symbolic form. An overview is given by Schiehlen (1990), Paul (1981), Spong *et al.* (2004) and Kucuk and Bingul (2006). The motion equations of robots may always be produced in closed form, but lead to a high complexity of the equations. Equations in symbolic form are usually much more efficient in the evaluation than the purely numeric procedures, because many simplifications can be employed (Craig, 2003, Fisette and Samin, 1993, Vukobratovic and Kircanski, 1982, Westmacott, 1989).

The Robotics Toolbox for Matlab (Corke, 1996) allows the user to create and manipulate fundamental data types with ease, such as homogeneous transformations, quaternion and trajectories. Functions provided for arbitrary serial-link manipulators include forward and inverse kinematics, and forward and inverse dynamics.

In most cases, the manipulator has to be controlled in the workspace, which is defined by external world coordinates and not in the configuration space, which is defined by internal joint coordinates. Therefore, a transformation between world and configuration space is required (Craig, 2003, Lenz and Pipe, 2003, Maël, 1996, Russell and Norvig, 2002).

The forward kinematics is a continuous mapping of the joint coordinates from the multi-dimensional configuration space to the world coordinates, and is described in detail by Craig (2003).

The inverse kinematics problem involves finding joint coordinates so that a desired world coordinate is reached. Calculating the inverse kinematics is generally hard, especially for robots with many degrees of freedom. This problem is ill posed because the solution does not have to be unique. In particular, considering an unreachable target, no solution exists at all (Russell and Norvig, 2002).

The kinematics of a robot may also be seen as a non-linear system, which can be approximated by mapping the input space to an output space of a function. Neural networks have the ability to learn such mappings, and they are therefore called 'function approximators'. A general example with a Continuous Self-Organizing Map is given by Aupetit (2000). Features of neural networks are utilized to learn the kinematics of a robot, which is an open- or closed-loop kinematic chain, and is not often precisely known. Maël (1996) proposes a hierarchical network for visual servo coordination which is based on the publication of Ritter *et al.* (1992). The hierarchical approach allows the learning of geometric models of realistic robots with six or more axes. The network consists of several one-dimensional sub networks which learn the coordinate transform and rotation axis for each joint below the visual error. A dynamically-sized radial basis function Neural Network was developed by Lenz and Pipe (2003) to control a six-axis Puma 500 robot on a slow 16-bit microcontroller. Following Ge (2004), given a nonlinear robot system, model-based control is superior to non-model-based control. On the other hand, for complex nonlinear systems, it is more difficult to obtain a realistic model than to design a working control system in reality.

2.3 Configuration Space Discretization

The configuration space of the robot is often used by a path-planner to solve the problem of finding a collision-free path. A robot with n degrees of freedom is usually a manifold of dimension n (LaValle, 2006). This manifold is called the configuration space of the robot, and is considered as a state-space. Within such a state-space, the problem of path finding may be abstracted to the problem of finding a path that goes through the manifold. Discretisation of the configuration space is also important. Often, uniform discretisation is used, but in the work by Reif and Wang (2000), they developed a non-uniform discretisation approximation method with interesting properties for path planning.

The ego-kinematic space of a robot has been defined in literature (Glavina, 1990, Mínguez *et al.*, 2002). A robot can be considered a 'free-flying robot' with no constraints.

Thus, the path-planning algorithm does not need to take care of the configuration space of the robot. This is realized by ego-kinematic transformations. Because the kinematic constraints are embedded in the ego-kinematic transformation, the admissible paths are mapped onto straight lines in the transformed space, and each point of the ego-kinematic space may be reached by a straight-line motion of 'free-flying behaviour'.

The state space of the robot configuration space is often infinite. Sampling-based planning algorithms may consider a small number of samples to reduce the running time (LaValle, 2006). Therefore, path planners often use sampling strategies that are based on the specific path planning problem and environment. Known strategies are random and deterministic sampling schemes (LaValle, 2006). Random sampling schemes take samples from the configuration space of the robot in a uniform manner; every state of the configuration space must have an equal opportunity to appear in the sample. Deterministic sampling schemes are pre-defined sampling techniques (LaValle and Kuffner, 2000). They have the advantages of classical grid search approaches and a good uniform coverage of the configuration space, but require long processing times (Branicky *et al.*, 2001, Lavalle *et al.*, 2000, Lindemann and LaValle, 2004). Reif and Wang developed (2000) an algorithm with non-uniform discretisation for motion planning, where the discretization is greater in regions that are farther from all obstacles.

2.4 Path and Trajectory Planning

Trajectory planning is a fundamental problem, and significant research has been conducted over the past few decades, either in static or in dynamic environments, for example in the process of spray painting (Chen *et al.*, 2008). Trajectory planning includes the generation of a trajectory from the start to the target position, giving consideration to objectives, such as minimizing path distance or motion time, and avoiding obstacles in the environment and satisfying the robot kinematics. Motion planning is usually decomposed into path planning and trajectory tracking. The former generates a nominal trajectory, whereas the latter tracks that trajectory.

In robotics, the search space is most often the configuration space (LaValle, 2006). Some path-planning algorithms try to compute the entire configuration space, which is useful for low degree-of-freedom robots to find a global path (LaValle and Kuffner, 2000). However, for systems with high degrees-of-freedom, the computing time rises exponentially.

Therefore, path-planning algorithms often try to find an approximated solution to reduce computation time (LaValle, 2006). For example, roadmap methods (Geraerts and Overmars, 2002) do not compute the whole configuration space, but try to generate a roadmap of suitable configurations. Apart from roadmap-based techniques, the potential field approach (Koren and Borenstein, 1991, Warren, 1989) and cell-based method (Kitamura *et al.*, 1995, Ranganathan and Koenig, 2004) are two popular path planning approaches.

Path planning often includes searching the shortest path within a given graph. This can be accomplished with shortest-path search algorithms like the Dijkstra, A* or D* (Goto *et al.,* Likhachev *et al.*, 2005, Xiang and Daoxiong, 2011). The A* algorithm is one of the most important algorithms because its implemented heuristic enhances the search algorithm by directing the search to the target node.

2.4.1 Graph based Path Planning

Graph based approaches are also known as skeleton (Yang and Hong, 2007) or roadmap (Bhattacharya and Gavrilova, 2008) approaches. A free space, such as the set of feasible motions, is mapped onto a network of one-dimensional lines. The visibility (Yang and Hong, 2007) and cell decomposition graph (Lingelbach, 2004), Voronoi diagram (Bhattacharya and Gavrilova, 2008) and probabilistic roadmap (Kazemi and Mehrandezh, 2004b) are frequently-used skeletons, and are presented in the following subsections.

Visibility Graph

In a visibility graph, all obstacles are formed by polygons. These may be enlarged to allow a minimum clearance of the robot to the obstacle. A graph is generated by connecting the edges of the polygons and the start and target locations with linear polygon lines. Subsequently, this graph is used to find an optimal path. An example is demonstrated in Figure 1. The algorithm can be easily extended to a three-dimensional space, but it requires all obstacles being available and real-time calculation of the trajectory seems difficult, especially when new obstacles are detected.

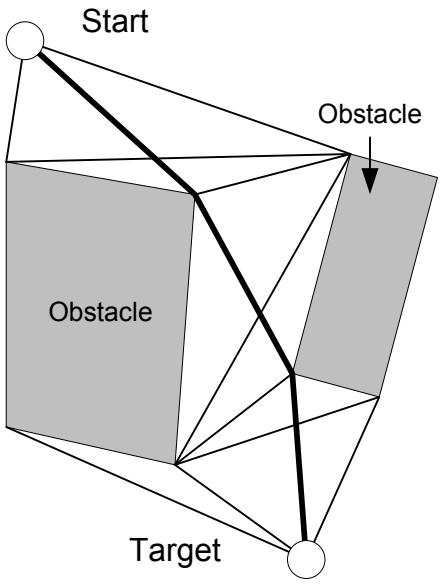


Figure 1: Illustration of the visibility graph (by author).

Cell Decomposition Graph

The cell decomposition graph subdivides a given free space into cells. One example of such subdivision is illustrated in Figure 2. The world model, which is the in-memory model of the surrounding, is delimited to a rectangle. For each edge of the obstacles, a horizontal line is included. The bisecting of each line is a point of the graph, and therefore, the horizontal clearance to obstacles is maximized. Extension to a three-dimensional space is difficult and the path obtained in this way is long.

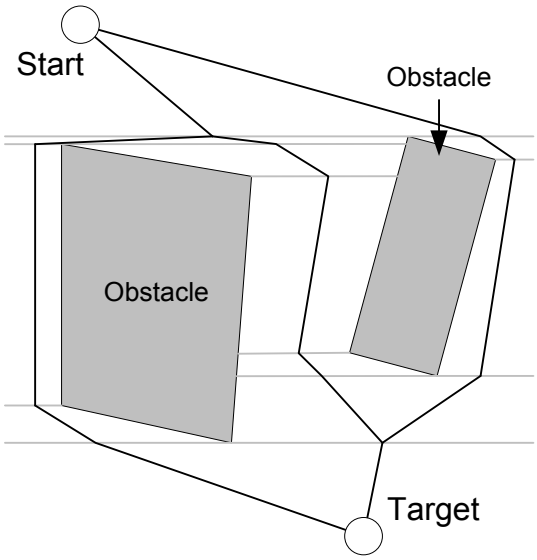


Figure 2: Cell decomposition graph (by author).

Cell decomposition methods generally divide the robot's free space into cells. The connectivity graph is built by connecting adjacent cells. A channel leading from the start to the target configuration through the graph may then be computed. A path may be chosen leading through the midpoints of the intersections of two successive cells. Examples of grid-based approaches are cell decomposition methods, which convert the configuration space of the robot in discrete cells. The cell division may be either object-dependent or -independent. Both cases are shown in Figure 3. A path is required to connect the start and the target node with a sequence of adjacent cells, which can be computed using a shortest-path search algorithm.

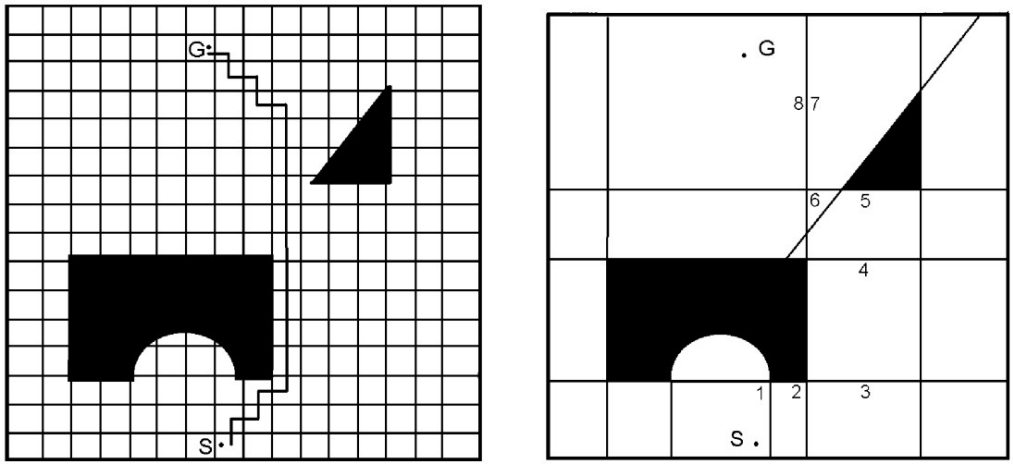


Figure 3: Cell decomposition with black obstacles and free space (by author).

Voronoi Diagrams

According to Hoff *et al.*, a Voronoi diagram consists of a given set of Voronoi sites, which partitions space into regions, where each region consists of all points that are closer to one site than to any other (Hoff *et al.*, 1999). An example of a Voronoi diagram is illustrated in Figure 4.

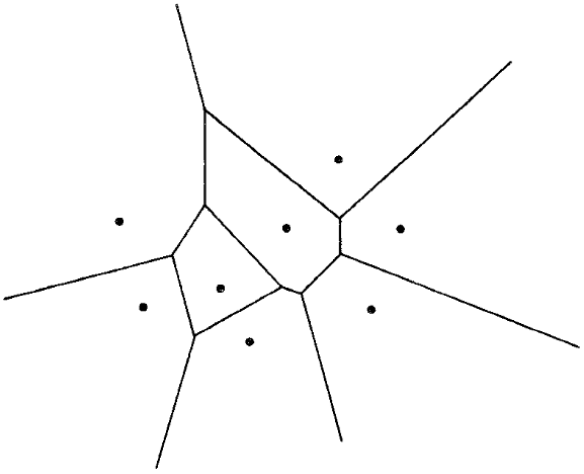


Figure 4: A Voronoi diagram with regions, where each region consists of all points that are closer to one site than to any other (by author).

Voronoi diagrams have been shown to be powerful tools in solving seemingly unrelated computational problems, and therefore have increasingly attracted the attention of computer scientists in the last few years. Efficient and reasonably simple techniques have been developed for the computer construction and representation of Voronoi diagrams.

Voronoi-based path planning methods have been studied in literature (Bhattacharya and Gavrilova, 2008, Fortune, 1986, Hoff *et al.*, 2000, Hoff *et al.*, 1999, Kim *et al.*, 2009, Vleugels *et al.*, 1993). The basic properties of a Voronoi diagram are treated by Aurenhammer (1991), who also recommended the publications of Preparata and Shamos (1985) and Edelsbrunner (1987). Hoff *et al.* (1999) presented a computational algorithm for generalized Voronoi diagrams, and did a survey of existing Voronoi computation algorithms for two and higher dimensions. The presented Voronoi computations are the divide-and-conquer algorithm (Shamos and Hoey, 1975) and the sweep line algorithm (Fortune, 1986). Numerically robust algorithms for constructing Voronoi diagrams have also been proposed in literature (Ingaki *et al.*, 1992, Sugihara and Iri, 1994). Higher-order Voronoi diagram computations have been summarized by Okabe *et al.* (2008) based on incremental and divide-and-conquer techniques. The set of algorithms includes divide-and-conquer algorithms for polygons (Lee, 1982, Martin, 1998), an incremental algorithm for polyhedra (Milenkovic, 1993), and three-dimensional tracing for polyhedral models (Culver *et al.*, 1999, Milenkovic, 1993, Sherbrooke *et al.*, 1995).

Hoff *et al.* (1999) stated that the computation of generalized Voronoi diagrams involves representing and manipulating high-degree algebraic curves and surfaces and their

intersections, and as a result, there are no known algorithms for their computation that are both efficient and numerically robust. Many algorithms compute approximations of generalized Voronoi diagrams based on the Voronoi diagram of a point sampling of the sites (Sheehy *et al.*, 1995). However, the derivation of any error bounds on the output of such an approach is difficult, and the overall complexity is not well understood.

Recent work aimed at reducing the length of the path obtained from a Voronoi diagram was presented by Yang and Hong (2007). The method involves the construction of polygons at the vertices in the roadmap where more than two Voronoi edges meet. This results in a smoother and shorter path than that obtained directly from the Voronoi diagram. The authors Wein *et al.* (2005) created a new diagram called the Visibility-Voronoi diagram to obtain an optimal path for a specified minimum clearance value.

Vleugels *et al.* have presented an approach that adaptively subdivides space into regular cells, and computes the Voronoi diagram up to a given precision (Vleugels *et al.*, 1996, Vleugels and Overmars, 1995). Lavender *et al.* (1992) used an octree representation of objects, and performed spatial decomposition to compute the approximation. Teichmann and Teller (1997) computed a polygonal approximation of Voronoi diagrams by subdividing the space into tetrahedral cells. All of these algorithms require considerable amounts of time and memory for large models that are composed of a large number of triangles, and therefore cannot be easily extended to handle dynamic environments directly.

Probabilistic Roadmap

Sampling-based motion planners such as probabilistic roadmap methods (Kavraki and Latombe, 1994) or those based on the rapidly exploring random tree (Kuffner and LaValle, 2000) provide good results for robot path planning problems with many degrees-of-freedom. Its success is based on the sampling method of the configuration space, e.g. the explicit characterization of configuration space obstacles is not required, and the aim of avoiding collisions is reached only by checking sample configurations of the configuration space. To improve the sampling efficiency and to find a path with as few configuration space samples as possible, several variants have been proposed to bias the sampling towards the most promising and difficult regions. For instance, a sample distribution is defined such that it increases the number of samples on the border of the configuration space obstacles (Boor *et al.*, 1999) around the medial axis of the free configuration space

(Wilmarth *et al.*, 1999) or around the initial and goal configurations (Sánchez and Latombe, 2002). In addition, the use of an artificial potential field was proposed to bias the sampling towards narrow passages (Aarno *et al.*, 2004, Kazemi and Mehrandezh, 2004a).

A probabilistic road map path planner was described by Sánchez and Latombe (2003) with a single query, bi-directional and systematic lazy collision-checking strategy. It is shown that this approach reduces planning times by 'large factors', making it possible to efficiently handle difficult planning problems, for example problems involving multiple robots in geometrically complex environments. This approach was successfully employed for several planning problems involving robots with 3 to 16 degrees-of-freedom operating in known static environments.

Narrow passages in configuration space can hardly be found. Results published by Hsu *et al.* (1998) attempt to solve that problem using a new random sampling scheme. An initial roadmap is built in a 'dilated' free space allowing some penetration distance of the robot into the obstacles. This roadmap is then modified by re-sampling around the links that do not lie in the true free space. Experiments have shown that this strategy allows relatively small roadmaps to capture the free space connectivity reliably.

2.4.2 Potential Field Based Path Planning

Potential field methods are straightforward approaches used to calculate a vector field based on target and obstacle locations; the robot follows the vector field until it reaches the target (Khatib, 1986, Koditschek and Rimon, 1990, Waydo, 2003). These planning algorithms often divide the free space into a fine regular grid, and use this grid to search for a free path. Different potentials are assigned to the cells of the grid, where 'attractive' potentials are given to cells close to the target and 'repulsive' potentials are assigned to cells close to obstacles. A path is constructed along the most promising direction. An example is shown in Figure 5.

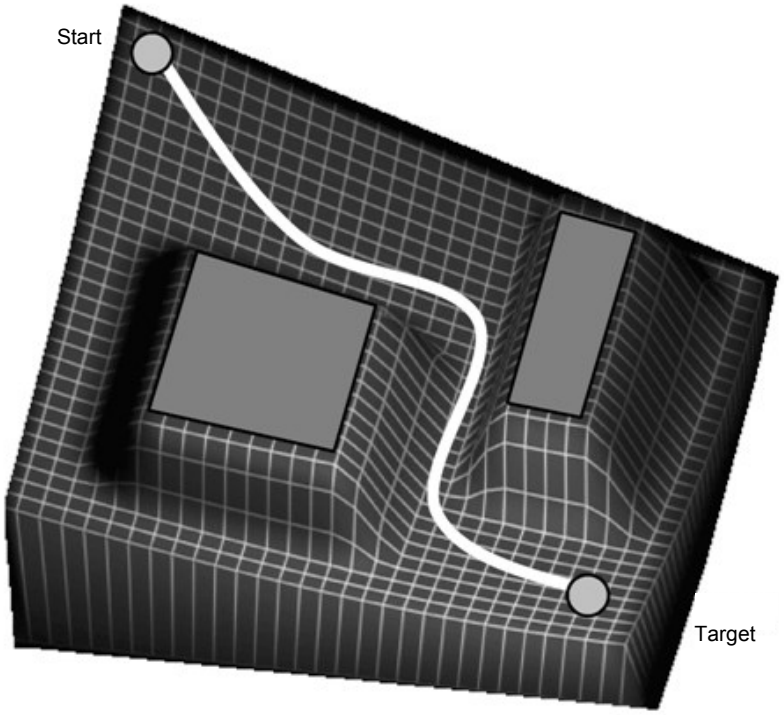


Figure 5: Potential field method.

Potential field methods have given good results, although not in high-dimensional configuration spaces, since an approximated decomposition of the configuration space is usually required (Barraquand and Latombe, 1991).

The cell-based method has been studied in combination with the potential field by Kitamura *et al.* (1995), and has been successfully applied to arbitrarily shaped robots in dynamic environments.

Yang and LaValle (2003) extended potential-field based methods to higher dimensional configuration spaces, combined with a random sampling scheme. A similar approach proposes global navigation functions over a collection of spherical balls of different radius that cover the free configuration space (Yang and LaValle, 2004). Those balls are arranged as a graph that is incrementally built following sampling-based techniques. The original concept of potential-field navigation is summarized by Khatib (1986). The topological properties of navigation functions are described by Koditschek and Rimon (1990).

The potential-field method developed by Pipe (2001) utilizes topographical cognitive mapping to store the locations of the robot environment, linking them with a value for ‘pleasant’ and ‘unpleasant’ experiences. Obstacles are for negative reinforcement and

energy charging sites for positive reinforcement. The knowledge is stored in a Radial Basis Function (RBF) neural network using techniques such as temporal difference (TD) learning and evolution strategy (ES). Inherent features of this neural network type lead to the creation of a potential-field structure that exerts appetitive and aversive 'forces' on the robot while moving in the environment. Potential-field methods are powerful approaches which appear to be promising, especially in a mixture of neuronal nets. Much more work can be found in literature (Arkin, 1992, Arkin, 1989, Arkin, 1987, Arkin and Craig, 1989a, Arkin and Craig, 1989b, Chuang, 1998, Ge and Cui, 2000, Koren and Borenstein, 1991, Masoud and Masoud, 2000, Rao and Arkin, 1990a, Rao and Arkin, 1990b, Valavanis *et al.*, 2000).

A three-dimensional potential field was proposed by Fujimura (1995) considering collision avoidance in static environments. It was demonstrated that both potential functions and their gradients due to polyhedral surfaces can be derived analytically, and this may facilitate efficient collision avoidance. The continuity and differentiability properties of a particular potential function were investigated. Koren and Borenstein (1991) discussed limitations of the mentioned potential-field methods, and Ge and Cui (2000) discussed solutions for non-reachable targets in potential fields, that is, when obstacles are near to the goal. Repulsive functions are improved by taking into consideration the relative distance between the robot and the goal. This ensures that the goal position is the global minimum of the total potential.

The potential field approach requires the decomposition of the configuration space (Barraquand and Latombe, 1991) that might lead to high processing times. In addition, in a real-time scenario, where the obstacles are not known beforehand, a complete recalculation of large portions of the potential field might be unavoidable. The algorithm may get stuck in local minima.

2.4.3 Harmonic Functions Based Path Planning

Potential-field approaches based on harmonic functions have good path planning properties, although an explicit knowledge of the robot configuration space is required.

Kazemi *et al.* (2005) applied a sensor-based probabilistic approach to build an online map with the use of harmonic functions for path planning. It iteratively extends the knowledge of the environment using laser range sensors, thereby extending the map. No prior knowledge of the environment is needed.

Connolly *et al.* (1990) described the application of numerical solutions of Laplace's equation to robot navigation, which lead to harmonic functions. These are resolution-complete planners without local minima (Connolly, 1992). The panel method of hydrodynamic analysis is applied by Kim and Khosla (1992) to develop analytic approximations to stream functions for complex geometries. Important reference work on potential field navigation is given by Masoud and Masoud (2000).

A combination of harmonic functions and sampling-based probabilistic cell decomposition methods for path planning is used by Rosell and Iniguez (2005) to bias cell sampling towards more promising regions of the configuration space. Cell classification is performed by evaluating a set of configurations of the cell obtained with a deterministic sampling sequence that provides a uniform and incremental coverage of the cell. In general, sampling-based methods allow the use of the harmonic functions approach without the explicit knowledge of the configuration space.

An electrostatic field approach without minima is described by Valavanis *et al.* (2000). In addition to path planning in static environments, dynamic environments are also treated. The well-formulated and well-known laws of electrostatic fields are used to prove that the proposed approach generates a resolution-complete optimal path in a real-time frame.

Harmonic functions suffer from the same disadvantages like the potential field approach, although they do not have local minima. Their extension to higher configuration spaces is reported to be difficult (Kazemi and Mehrandezh, 2004b).

2.4.4 Neural Network Based Path Planning

Literature for robot motion planning in unknown environments using neural networks has been discussed in various publications (Lebedev *et al.*, 2003b). A situation-action map is introduced (Knobbe *et al.*, 1995) for car-like robots (Latombe, 1991). New edge detection on the visible objects generated possible motions to escape from dead end situations while backtracking has been employed to choose from different possibilities.

Vleugels *et al.* (1993) present a new probabilistic road map approach that combines a neural network and deterministic techniques with the scope of solving the path-planning problem with a coloured version of a Kohonen map. Random configurations of the robot are inputted to the network, which constructs a road map of possible motions in the

workspace, and approximates the obstacles. This road map is searched to find motions connecting the given start and target configurations of the robot.

The sampling scheme of the presented algorithm by Vleugels *et al.* (1993) requires random configurations of the robot, which is infeasible for a real-time path planning approach.

2.4.5 Movement Planning

Movement planning usually takes the geometric and kinematic constraints of the robot into account. Different approaches have been developed using randomized or graph-based planners. Movement planners often have a constraint on the steering angle (Barraquand and Latombe, 1989, Fraichard, 1999). Such robots have dependent degrees-of-freedom, and thus, the motion is restricted. A feasible trajectory has to be found for the robot, to be able to route the robot position from the start to the target without collisions. In addition, the boundary conditions imposed and dynamics of the kinematic model of the robot have to be satisfied.

In the geometric formulation of the movement problem, the robot is reduced to a point on a two-dimensional surface with a behaviour that is similar to Dubins car (Dubins, 1957), which is only able to drive forward, and the radius of the steering is bounded. The resulting paths must be differentiable and feasible for the robot. An extension of the Dubins car is given with the Dubins airplane, which applies to three-dimensional spaces (Chitsaz and LaValle, 2007).

2.5 World Model

The modelling of the environment of the robot is necessary for an inner representation of the world. Often, this format is a boundary representation or a solid representation. While the former is a surface representation of the objects within the environment, the latter is a collection of points in space.

A number of approaches such as memory-based techniques (Blaer and Allen, 2002, Matsumoto *et al.*, 1996, Payeur, 2004), expressions by features such as line segments (Gutmann *et al.*, 2001, Newman *et al.*, 2002), parametric expressions (Brooks, 1983, Quek *et al.*, 1993) and mesh modelling methods (Hilton *et al.*, 1996, Wheeler, 1996) are suitable for world modelling. For example, Boada *et al.* (2004) combined topological and geometric information to model the environment. This map is obtained from a Voronoi

diagram using measurements of a laser telemeter. In addition, other approaches can also be found in literature (Kagami *et al.*, 2003).

Gran (1999) shows simplification algorithms for the generation of a multi-resolution family of solid representations from an initially polyhedral solid. Discretised polyhedral simplifications using space decomposition models are introduced based on a new error distance. This approach provides a scheme for the error-bounded simplification of geometry topology, preserving the validity of the model.

Another proposed method uses trihedral discretised polyhedral simplifications and an octree for topology simplification and error control (Garland, 1999). This method is able to generate approximations that do not affect the original model. It is either completely contained in the input solid or bounded to it, and can handle complex objects. A brief overview to object simplification with various algorithms was presented.

Knuth (1973) employed a uniform grid to store the data. The space is divided into equal sized cells, that is, squares and cubes for two- and three-dimensional data, respectively.

Hierarchical data structures were also presented (Gargantini, 1982a, Gargantini, 1982b, Payeur *et al.*, 1997, Schrack, 1992), and can be applied in order to save memory consumption. The most important approach is a linear region quadtree or octree that recursively subdivides the space into four or eight equal-sized space regions. Such space partitioning data structures are used to store geometric data in a specified resolution. In robotics, it is often useful to find the neighbours of a cell. Finding the neighbours either on the same level or on a higher or deeper level within the hierarchy is explained in literature (Balmelli *et al.*, 1999, Bhattacharya, 2001, Lee and Samet, 2000, Samet, 1990, Schrack, 1992). Among other techniques such as Binary Space Partitioning (BSP) trees or k-Dimensional (k-D) trees, hierarchical data structures are also explained by Chang (2001).

Of the different existing neural network types, the growing neural network type is discussed in many applications such as surface reconstruction (Ivrissimtzis *et al.*, 2003) and robot path planning (Fritzke, 1991, Fritzke and Wilke, 1991, Vleugels *et al.*, 1993). Also, a self-organizing neural network is often employed for data visualization, clustering and vector quantization. The main advantage lies in its ability to find a suitable network structure and size automatically. This ability can also be exploited to reconstruct objects such as obstacles in the workspace of the robot.

However, growing neural net adaptation rules are mostly based on different approaches (Blackmore and Miikkulainen, 1993, Cheng and Zell, 1999, Fritzke, 1995, Fritzke, 1991, Fritzke, 1993, Fritzke and Wilke, 1991, Ivrissimtzis *et al.*, 2003, Lenz and Pipe, 2003). Fritzke (1995) explained in detail the power of growing neuronal nets, which are able to learn the important topological relations in a given set of input vectors by means of a simple Hebb-like learning rule. The net grows and continue to learn and add units and connections until a specified performance criterion has been met.

The concept of the coloured Kohonen map introduced by Vleugels *et al.* (1993) uses an adapted version of the growing neural network presented by Fritzke (1991) to identify the free and occupied working space for two different colours.

Another variant of the approach by Fritzke (1995) was proposed by Cheng and Zell (1999). The goal of their paper was to speed up the convergence of the learning process. A performance comparison between a Kohonen Feature Map and growing neural networks was explained in depth by Fritzke (1993).

Blackmore and Miikkulainen (1993) presented a growing feature map that is able to represent the structure of high-dimensional input data. An extension has been given with the approach used by Rauber (2002), where a growing hierarchical self-organizing map is built. This is an artificial neural network model with a hierarchical architecture, which is composed of independent growing self-organizing maps. The motivation of the authors was to provide a model that adapts its architecture during its unsupervised training process according to the particular requirements of the input data.

The algorithm proposed by Ivrissimtzis *et al.* (2003) samples a target space randomly and adjusts the neural network accordingly which also include the connectivity of the network. The speed is virtually independent from the size of the input data, making it particularly suitable for the reconstruction of a surface from a very large point set.

Triangle primitives are popular in computer graphics for surface reconstruction because they are also used by graphics acceleration hardware (LaValle, 2006). Combining neural network algorithms with triangle meshes leads to an algorithm for path planning, which is presented by Vleugels *et al.* (1993). An optimization of the quadtree is presented by Hwang *et al.* (2003) using triangles instead of a quadtree to improve object approximation.

They presented a path-planning algorithm that simplifies the triangle mesh into a compact and obstacle-dependent mesh to reduce the search space.

Data structures and algorithms of progressive triangle meshes were presented by Hoppe (1998). For a given mesh, this representation defines a continuous sequence of level-of-detail approximations, which allows smooth visual transitions among them and makes an effective compression scheme.

2.6 Vision and Perception

Path planning in robotics considers model-based and sensor-based information to capture the environment of the robot. Perception, which is initiated by sensors, provides the system with information about the environment and subsequently interprets them. Those sensors are, among others, cameras or tactile sensors which are often used for robot manipulators. Gandhi and Cervera (2003) presented a sensor skin for a robot manipulator. An approach based on touch sensors was also mentioned by Zlajpah (1999).

Vision-based sensing is the most useful sense for dealing with the physical world (Russell and Norvig, 2002). Extracting the pose and orientation of objects in images or an image stream and the detection of motion delivers useful information for path planning. Object recognition converts the features of an image into a model of known objects. This process consists of segmentation of the scene into distinct objects, determining the orientation and pose of each object relative to the camera, and determining the shape of each object. Those features are given with motion, binocular stereopsis, texture, shading and contour.

Motion estimation algorithms are presented in literature (Hsu *et al.*, 2002, Lippiello, 2005) to estimate motions of obstacles online for realistic environments. An introduction of image processing is given by Pollefeys (2000) and Russell and Norvig (2002). Peter Corke's Machine Vision Toolbox for Matlab (Corke, 2005, MathWorks, 1997) allows developers to use professional image processing capabilities with ease.

In many cases, the sensor data are redundant, uncertain, imprecise, inconsistent and contradictory. The knowledge of the spatial relationships among objects is also inherently uncertain (Nandi and Mitra, 2005). Those data should be considered to recognize errors. A review of papers on uncertainty analysis in the context of manipulator control (Di *et al.*, 1998, Langlois *et al.*, 2001, Mao-Lin and Meng, 2000, Smith *et al.*, 1990) shows that a

common step involved in all these systems is the interpretation of identical information that has been acquired through multiple sensory units. The fused information needs to be represented with minimized uncertainty, and the level of this minimization depends on task-specific applications.

2.7 Collision Detection and Avoidance

Path planning in a dynamic environment with moving obstacles is computationally hard (Hsu *et al.*, 2002), and several solutions have been proposed in the past (Akgunduz *et al.*, 2005).

One solution is to ignore moving obstacles and to compute a collision-free path of the robot among the static obstacles; the robot's velocity along this path is tuned to avoid colliding with moving obstacles (Kant and Zucker, 1986). However, the resulting planner is clearly incomplete. The planner developed by Fujimura (1995) tries to reduce incompleteness by generating a network of paths. The planner proposed by Fraichard (1999) dealt concurrently with velocity and acceleration constraints and moving obstacles, such as car-like robots. It extends the approach of Donald *et al.* (1993) and Erdmann and Lozano-Perez (1987) to the state-time-space, which solves the trajectory-planning problem for velocity- and acceleration-constrained movements. It also transforms the problem of searching the time-optimal canonical trajectory to one of searching the shortest path in a directed graph embedded in the state-time-space. The concept augments the state space with the time dimension, and is useful for trajectory planning.

Hsu *et al.* (2002) presented a randomized motion planner for robots that avoids collisions with moving obstacles under kinematic and dynamic constraints. The planner does not pre-compute the roadmap; instead, for each planning query, it generates a new roadmap to connect the start and target state-time points. A vision module estimates the obstacle motions just before planning, and the planner is then allocated a small amount of time to compute a trajectory. If a change in the obstacle motion is detected while the robot executes the planned trajectory, the planner re-computes a trajectory on the fly (Boada *et al.*, 2005, Etzion and Rappoport, 2002, Kim *et al.*, 2009, Kitamura *et al.*, 1995, Lebedev *et al.*, 2003a, Nagatani and Choset, 1999, Vleugels and Overmars, 1995).

Another approach employed for collision detection was given by Sánchez and Latombe (2003). To reduce the time needed to check collisions, this strategy postpones collision checks until they are absolutely needed. Schwarzer *et al.* (2004) provided a collision-

checking method that tests single straight-line segments, sequences of such segments, or more complex paths in the configuration space. It was shown that this approach is faster when compared to resolution-based approaches with a suitable resolution. The spatial potential field by Chuang (1998) shows that potential functions and their gradients can be derived, and may therefore facilitate efficient collision avoidance.

2.8 Model Driven Software Development

The object management group (OMG) (Object Management Group, 2011) is an international, open membership, not-for-profit computer industry consortium that provides modelling standards such as the Unified Modelling Language (UML), Model Driven Architecture (MDA), and Common Object Request Broker Architecture (CORBA). These standards have been applied to several projects of the eclipse development environment (Eclipse Foundation, 2006, Eclipse Foundation, 2011a, Eclipse Foundation, 2011b, Eclipse Foundation, 2011c, Eclipse Foundation, 2011d). A model-based execution system has been presented by the eTrice Group (2011) and Pontisso and Chemouil (2006). An overview of domain-specific programming, which is most often part of a model-based code generation framework, was given by Shani and Sela (2010).

2.9 Summary

Industrial manufacturing requires more intuitive human-machine interfaces and sensory interfaces to reduce reliance on the operator skill and to improve automation. Online robot programming leads to a loss of production and reduces preparation times, which are necessary for the counterpart of online programming, namely, offline programming. The offline generation of robot programs needs a simulation and programming phase executed by skilled engineers. This is time consuming and requires specialized and expensive simulation software. Thus, small- and medium-volume manufacturing do not benefit from this technology. Industrial production may be improved with enhanced online programming for industrial robots.

This enhancement can be attained with an assisted online robot programming system, which can be operated with ease. The required human-machine interface is closely connected with the underlying trajectory-planning algorithm to support the robot-programming task.

The perception of the environment and the representation of the in-memory world model play an important role in the efficient utilization of the environment information for trajectory planning. Vision and perception has to be appended by other sensor types and fused into the in-memory representation of the environment. Research is required, especially for robot programing in the industrial surrounding to utilize existing data sources.

The trajectory planner has to deal with both the available information and the operational requirements of the enhanced programming system.

In general, the computation time of algorithms can be reduced by introducing hierarchical subdivision approaches such as quadtree- and octree-based methods (Gargantini, 1982b).

Cell-based planning methods often generate a path that connects the midpoints of the cells. The publication by Hwang *et al.* (Hwang *et al.*, 2003) identifies two limitations with cell-based methods. First, the detection of small passages requires high accuracy of the octree or quadtree. Secondly, the shortest path is not always identified since the distance calculations of the cells often use the midpoints of the cells. Thus, the paths obtained by the cell-based method are not optimal because of the connectivity limitations in a grid.

The potential-field approach has several limitations, as outlined in the work of Koren and Borenstein (1991). In particular, the robot may get stuck at a local minimum and the reported paths can be arbitrarily long.

Trajectories that are directly obtained from Voronoi-based path planning methods are often long, and are not smooth. In recent years, much research has focused on improving the quality of the path. Masehian and Amin-Naseri (2004) combine the Voronoi diagram with the visibility graph and potential-field approach into a single path-planning algorithm to obtain a trade-off between safest and shortest paths. The algorithm is complicated, but the path length is shorter than the paths obtained from the potential-field method or the Voronoi diagram.

Neural networks have the ability to learn from input vectors. Among its most important benefits are object and environment recognition, generalization to new situations, evaluation of situation-contexts, short and long-term memory and their real-time ability.

Research of trajectory planning with neural networks for real robot systems has been given less attention in the past.

The literature survey shows that a large amount of scientific work has been done in the last decades. However, in the context of robot-program file generation for robot manipulators in deterministic industrial environments, other prerequisites have to be taken into consideration.

The usability of the robot-program generation application is an important factor and it has to be analysed in detail. For example, not only the usage but also the created final robot program file is relevant for a good usability. It has to comply with guidelines for manually created robot program files to allow manual amendments. Nevertheless, the trajectory planning process within the production system has to be applicable by inexperienced users, which requires an intelligent expert system to support the user. The intelligence contains the human machine interaction, the path-planning algorithm and the knowledge transfer between the user and the expert system. An additional aspect is the trajectory planning process that might become easier in a deterministic production environment where only objects with a predictable motion may exist.

This work is focused on an intuitive expert system for industrial use and the acquisition of industry requirements sets the basis for further investigations, such as the trajectory-planning algorithm, the world model, the robot kinematics and a suitable software development framework. The following aims chapter summarizes the aim and specifies the objectives treated in this work.

3 Aims

3.1 Motivation

Industrial production systems within the high-volume automotive industry are highly optimized. Further advancements may be achieved through a systematic improvement of the production process. Existing online robot programming approaches have not been completely accepted because of the required production downtime. Consequently, offline programming is generally employed even if it requires serious financial investments in terms of additional personnel and equipment costs. Furthermore, offline programming requires expensive simulation systems and skilled operators who are able to create the model of the specific production environment and to produce high-quality robot programs. Exact modelling of the production environment is a time-consuming task, although models of the production machines are most often provided by the manufacturer. Simulation systems generally allow the use of modelled production parts and fixtures to optimize the offline-programming process. This represents an improvement, especially when the models are not available as physical objects. The quality of the robot programs is highly dependent on the knowledge of the operator, who must be experienced in online robot programming and in the use of simulation systems. Nevertheless, offline programming still requires installation time to upload the robot programs and to adjust inaccuracies and errors resulting from unknowns and inaccuracies in the environment. Finally, offline-created low-quality robot programs are most often re-programmed online, presenting the risk of a production loss. This also affects the performance of the robot programmer, set under high pressure.

Costs may be reduced by the development of a new robot programming system which is executed solely online, and which creates robot programs in a period of time that is comparable to the time necessary for the installation of offline programming approaches. A seamless integration into the existing industrial environment is required to reach a high acceptance level. This may be realized by combining the advantages of existing robot programming approaches and a new trajectory-planning algorithm, which is extended with an intuitive user interface.

Robot use and automation levels in the industrial sector will continue to grow in future, driven by the ever-present need for lower item costs and enhanced productivity. In order to support this market-driven requirement, more capable programming and control technologies will be necessary. Therefore, research has been undertaken to optimize the

robot programming process and to reduce personnel and equipment costs. Accordingly, this work addresses the future needs of the production industry.

3.2 Objectives

This work aims to present a method that can substitute the current robot programming approach with an enhanced robot programming system, in effect rendering offline programming an unnecessary technology. Offline programming is still an accepted and proven programming approach; the present production environment setup is well established in industry. Therefore, an analysis of the current key aspects regarding robot programming is required. The integration of those aspects into the new programming approach guarantees a high acceptance level and future employment of the new technology.

With no offline programming phase, robot programming can only be accomplished online. This aspect defines the scope of the new online robot programming approach. The required information for online programming, like mission data and computer aided design data, must be managed and processed online. The complexity of a tool that executes information management and the robot-programming task itself requires a usable frontend that encapsulates the complexity. The frontend provides assistance in order to ensure the simple use of such a complex tool which is able to interpret information and execute the robot-programming task by itself.

Online robot programming approaches are generally time critical since production downtimes have to be minimal. A crucial aspect that is able to support the general need to reduce the time lies in the development of a fast trajectory-planning algorithm. The knowledge acquired during the process will be efficiently employed to optimize online robot programming. This also includes the ability to handle inaccurate information, which may be obtained through sensors, the environment and pre-existing modelled information. The combination of the robot, the sensors and the software components requires a modern software development approach, which supports their integration into the proposed enhanced online robot programming system.

It is the intention to introduce an enhanced robot programming system that should be used solely online to reduce costs by entirely omitting the offline programming phase. Its handling should be kept simple, and should allow less experienced workers to apply the programming system in a more productive way. It can also be utilized by an expert to

increase his or her productivity. An online robot programming system can also be easily applied in small-batched productions, which is a field that is very sensitive to robot programming speed, system flexibility and cost efficiency. The high quality results of the system are reproducible, and the process itself still has the potential for further optimization and modernization.

Objective one: Requirements for adoption by industry of online programming. Important key features of current robot programming approaches have to be supported by an enhanced online robot programming system to reach a high acceptance level. Some of the key features may include the use of modelled production parts and fixtures that are physically unavailable, and the creation of high-quality robot programs. In addition, technical aspects of the industrial environment have to be considered to allow a seamless integration of the system. The results of the analysis are summarized in the requirements definition for the enhanced online robot programming system, also affecting the robot programming process. The results are described in Chapter 5.

Objective two: Investigation into an efficient probabilistic world model for data fusion. Trajectory planning relies on inexact information about the environment in which the robot operates, although sensor information is almost incomplete and inaccurate. Additional information such as the utilization of modelled data may be incorporated to improve the in-memory environment representation. The information sources are fused according to their reliability to provide cohesive information. A probabilistic world model stores the information statistically, and considers the history of the information. Objective two is to develop an efficient data-structure and information fusion algorithm which allows statistical environment data to be stored. The world model and information fusion system are described in Chapter 6.

Objective three: Research of the robot kinematics model and the robot control capabilities. The use of industrial-scale experimental machinery robot systems such as the Mitsubishi RV-2AJ manipulator is essential throughout the investigation to prove new theories. Furthermore, autonomous mobile robots such as the Festo Robotino robot may also be applied to verify control algorithms in a simplified two-dimensional space. This requires a robot communications and control framework for both robot types. In particular, the kinematics of the robots is required for forward and inverse calculations; they transform positions of the real world into the robot coordinate system. In this work, the

robot geometry and the joint types are applied to create a kinematic model of the utilized robots. The robot communications and control framework and the kinematics model of the used robots are described in Chapter 7.

Objective four: Investigation into a trajectory planning algorithm to support intuitive use of the robot programming system. The user and the enhanced online robot programming system should co-operate and safely interact, even in complex situations. Effective assistance requires that the robot be technically intelligent, and that there is a knowledge and skill transfer between the human and the robot. The co-operation depends on the recognition and perception of typical production environments as well as on the understanding of tasks in their context. During human-machine interaction, robot motions have to be planned and quickly co-ordinated. In compliance with the requirements for the enhanced online robot programming system, online programming needs to be a simple and fast method compared to other robot programming approaches. This also applies to less experienced operators. This is only possible with a high level of automation of the trajectory-planning task. Considering that the operation of an industrial robot is restricted to a small set of commands, the planned trajectories consist of circular and linear movement primitives. The robot-program generation transforms the trajectories into robot programs, which are stored in the robot-type specific program syntax file. The trajectory planning approach is described in Chapter 8.

Objective five: Research of a software development framework for complex systems. The system development and implementation of many hardware and software components require a clear and structured implementation approach. Model driven approaches have been shown to overcome this complexity, but have to be setup for their use in specific problem domains. Starting with the analysis of the current state-of-the-art technology, a model-driven code generation toolchain has been developed and implemented. The results derived using this analysis are presented in Chapter 9.

Much of the results have been published and the findings are appended.

4 Experimental

The executed experiments focused on human-machine interaction and trajectory planning algorithm development. The human-machine interface (HMI) and the trajectory planning are interconnected tasks, which have had an impact on the graphical user interface (GUI) design. A task-oriented approach was chosen to provide only relevant information and functions to the operator, based on a user interaction finite-state-machine. The user interface consists of a dynamic toolbar which proposes a standard robot programming workflow. It simultaneously offers extended interaction possibilities and maintains the effectiveness of the interface. The GUI itself provides a dynamic main screen that displays only task-relevant widgets.

The experiments regarding user interaction and the GUI design concentrated on usage experiences and an evaluation of standard graphical interface design rules. The trajectory-planning algorithm was first tested with an autonomous mobile robot to omit forward and inverse robot position calculations and robot arm constraints. In the second step, the experiments were extended to an industrial scenario, which includes an articulated arm. These experiments were designed to prove the feasibility of the user-interaction and the trajectory generation.

The experiments completed as a part of this investigation were carried out using the Mitsubishi RV-2AJ manipulator and the autonomous mobile robot Robotino produced by Festo (Festo, 2011).

The mobile robot is a platform equipped with wireless communication and infrared distance measurement units, and it was employed for early algorithm tests. In addition, the implementation of a simulated robot accelerated the algorithm development and the user-interaction design because no direct connection to the real robot was necessary.

The manipulator is an advanced, but mature and industrially proven machine, and its commercial viability has already been demonstrated in the manufacture of car sub-assemblies, semiconductor memories and other industrial/consumer goods.

The connection to the robots was established using C# for the mobile robot (Festo, 2011) and a Java framework for the manipulator (Kohrt *et al.*, 2008). The communication and control capabilities of the manipulator were enhanced to extend sensor measurement and robot movement capabilities. The Mitsubishi documentation regarding controller commands is not complete. However, the data sent between the controller and the

Mitsubishi Software Cosirop (Mitsubishi-Electric, 2011) is not encoded, which allowed listening to the Ethernet communication between the controller and the personal computer. This helped to identify undocumented commands. Cosirop is a software development and simulation environment from Mitsubishi, which is used to program in Melfa Basic IV (Mitsubishi-Electric, 2003), illustrated in Figure 6.

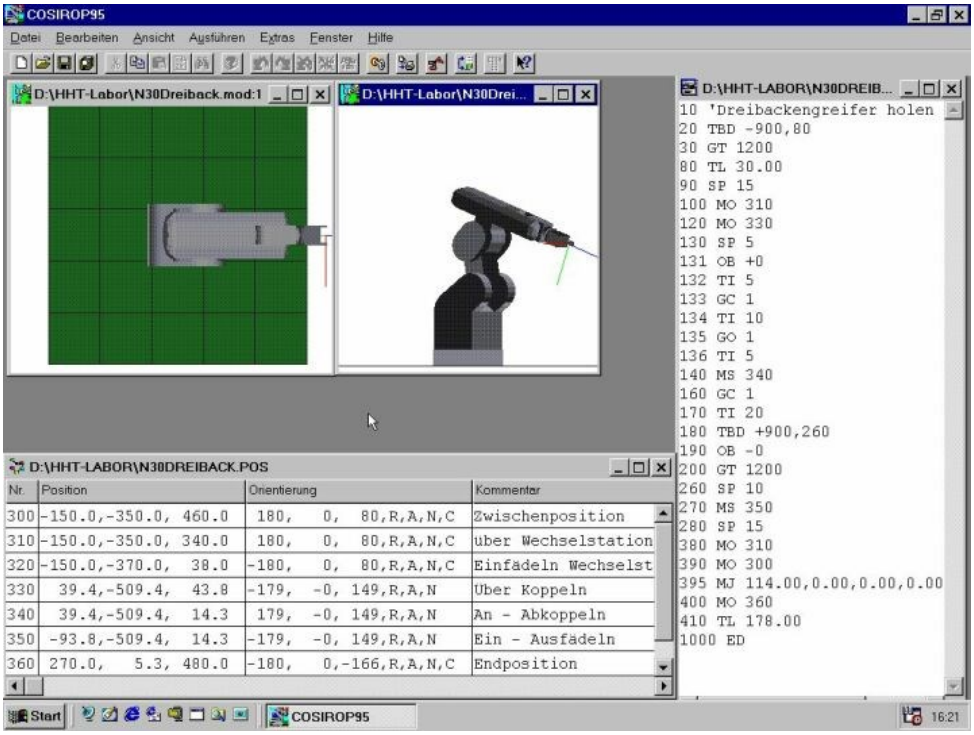


Figure 6: The Cosirop robot programming software.

Figure 7 shows the equipment employed, which is described in more detail in Appendix B.

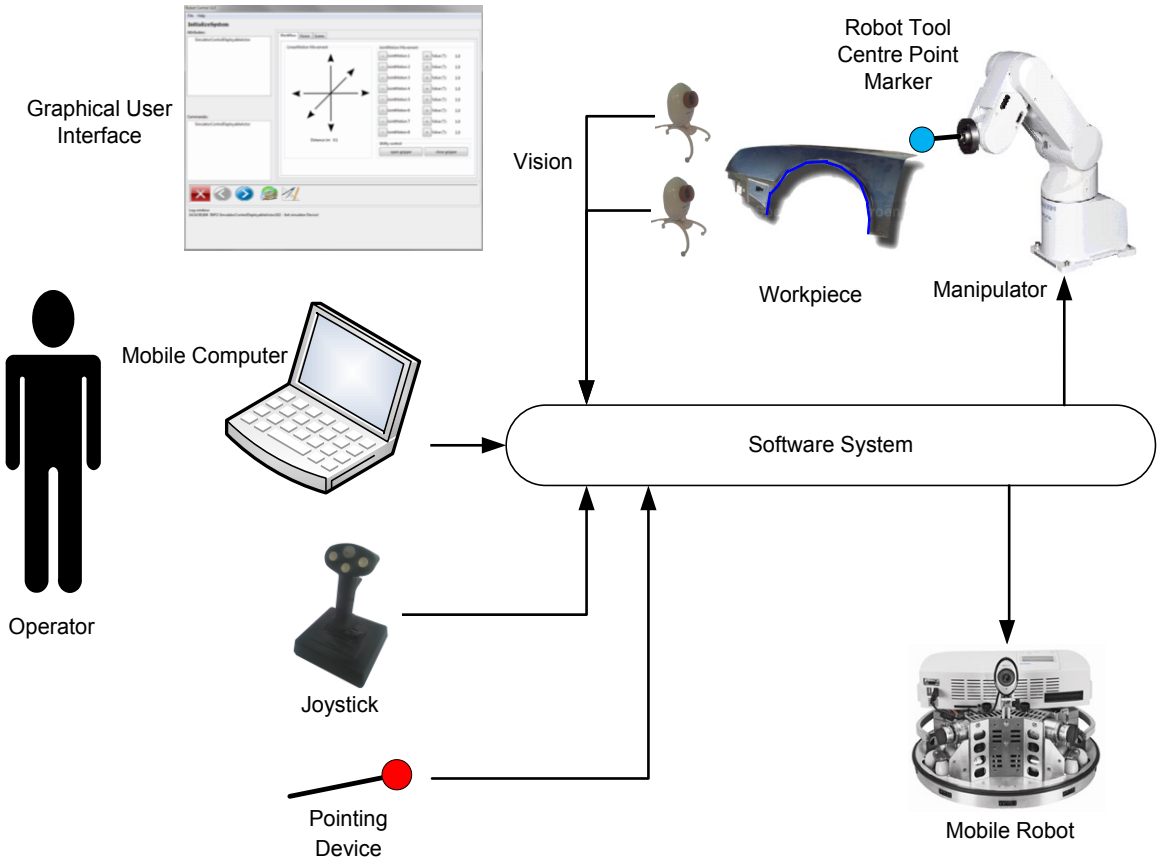


Figure 7: The experimental system.

The operator utilizes a GUI that was developed with the Java Standard Widget Toolkit (SWT) framework on a Windows operating system (Kohrt *et al.*, 2006a). The buttons on the GUI and the Joystick were applied to indicate collisions. Vision sensors are connected and processed by a Matlab/Simulink generated C++ code. The GUI, the joystick and pointing device allow the control of the employed robots.

The pointing device is a 50 cm long stick with a single coloured 2.5 cm-diameter red ball that is used as a marker for position recognition. Different marker colours were chosen, e.g. for the robot-arm and the pointing device, so that they can be distinguished from each other.

Other sensory modalities, such as machine vision, distance measurement and ultrasonic sensors, may also be included through the sensor fusion framework. The choice of sensor types depends greatly on the application. The vision system was utilized for the recognition of the markers.

The software was installed on a mobile computer with a 32-bit Microsoft Windows 7 operating system running on an Intel Core i5 processor with a maximum frequency of 2.4 GHz. Other real-time capable systems, such as a PowerPC with a VxWorks operating system, may improve the performance of the system.

5 Requirements for Adoption by Industry of Online Programming

This chapter presents the findings from the investigation to the requirements for adoption of online programming by industry. This is objective number one, as outlined in Chapter 3. It identifies and specifies requirements for robot programming for small-batched, medium sized and high-volume manufacturing industries.

In Section 5.1, a typical production cell in the automotive industry is introduced and in the subsequent Section 5.2, offline programming approaches are analysed. The analysis identifies industry requirements for robot programming, which are summarized in Section 5.3. A new robot programming approach is presented in Section 5.4, which was researched based on the identified requirements. Section 5.5 compares the proposed programming approach with conventional online and offline programming. Moreover, a first system design which implements the new robot programming approach is introduced in Section 5.6. Finally, Section 5.7 summarizes the system requirements for the implemented enhanced robot programming support system.

5.1 Industrial Production Environment

A typical production environment within the automotive industry is illustrated in Figure 8. A work object, such as the chassis of a car, may be transported into a production system which consists of four robots installed on two external axes. Cameras may be used to measure the offset position of the work object. The robots may use this information to calibrate their robot programs in order to compensate positioning inaccuracies of the work object. In industry, it is also common to transport the work object with a conveyor during robot operation. The robot programs have to consider these usage scenarios which are most often supported by the robot manufacturer with special movement commands. Typical applications are welding, gluing, assembling, spraying, handling and picking and placing.

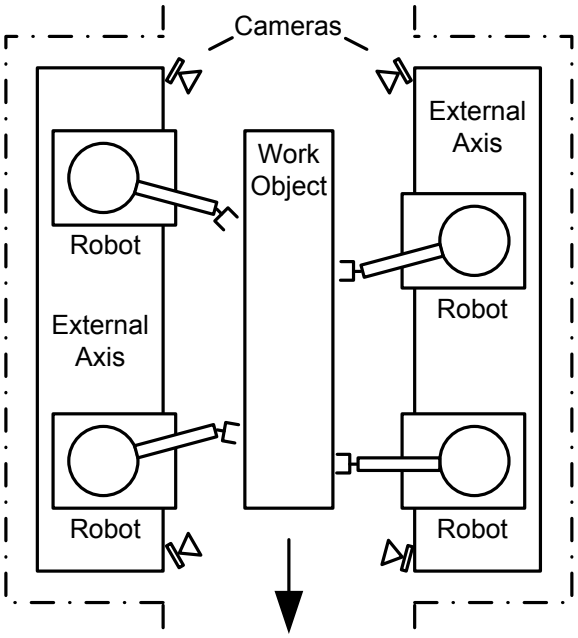


Figure 8: Typical production cell.

A production system usually requires a specialized communication, logic and control system, an example of which is shown in Figure 9. The logic component may be implemented using a Programmable Logic Controller (PLC), personal computer (PC) or one of the robot controllers to synchronize the entire production process with preceding and successive working tasks. The control component requires information such as mechanical, physical, electrical and logical data to control the production system. Robots are often utilized for production systems, and are usually equipped with robot control devices such as a teach pendant or other HMI.

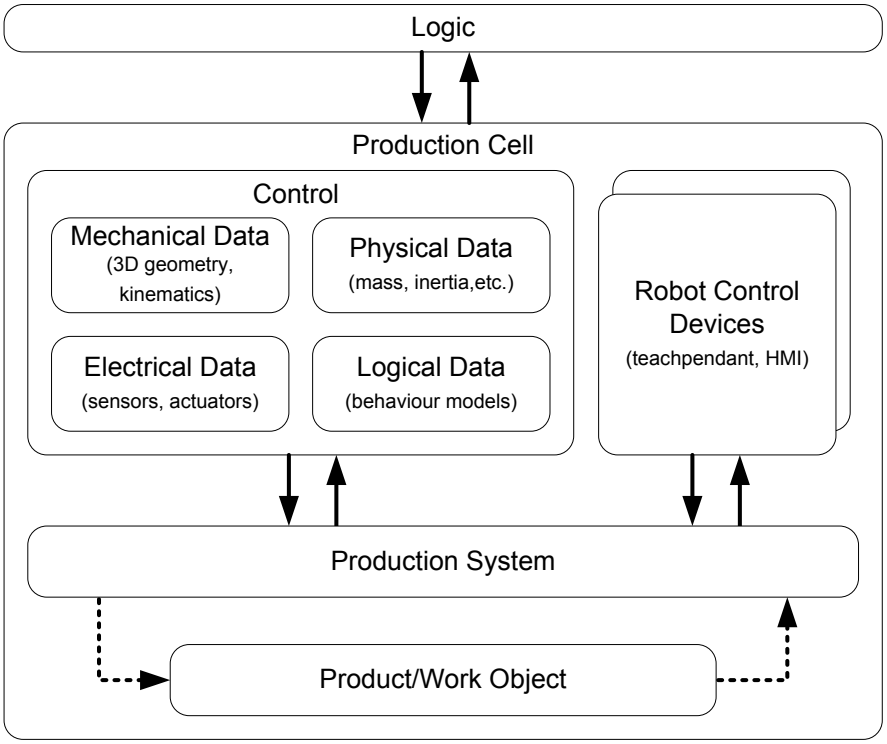


Figure 9: Schematic view of a production cell.

The control component may execute robot programs to control the robots. Increased product diversity is realized by implementing work-object dependent robot program execution. The identification of work objects is often achieved by bar codes or radio-frequency identification (RFID) chips on the work objects. The increased flexibility is also demanding for the material flow automation, since the correct production parts have to be delivered just in time.

The flexibility of robots makes them important for production applications, especially within the automotive industry. For example, Mercedes Benz uses robots for rear-axle assembly tasks of their C-Class car (Kiefer *et al.*, 2010). The analysis of a robot program in Figure 10 indicates that 68% of the program is related to the production task (movement instructions, variable declarations and syntactical instructions), while 32% are related to external communication and assembly procedures (plausibility checks).

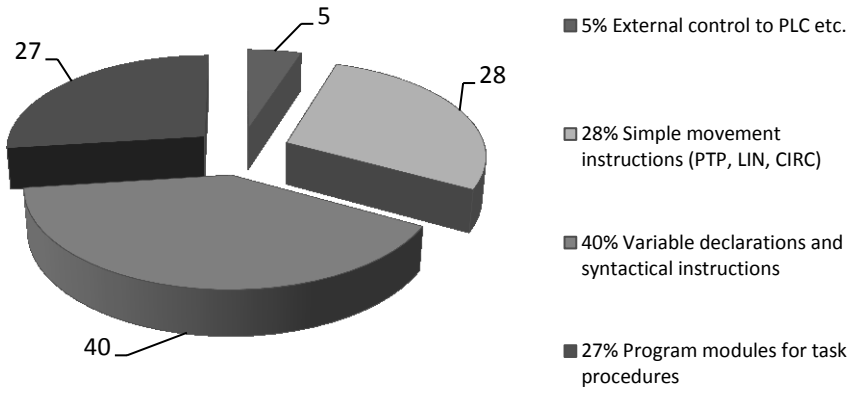


Figure 10: Robot program structure.

The automation of robot programming implies the automatic creation of the robot program structure which is illustrated in Figure 10. The life cycle of a production cell from the initial design to the operation stage is illustrated in Figure 11. To create and modify robot programs, research focused on the ‘Installation & Initial Setup‘ and the ‘Operation and Maintenance‘ phases.

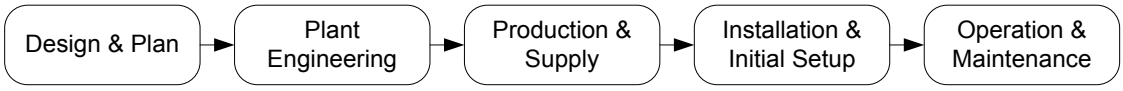


Figure 11: Production cell life cycle.

5.2 Analysis of Existing Robot Programming Approaches

The analysis of existing robot programming approaches focused on conventional online teach-in programming and offline programming amended by online teach-in programming. These two programming approaches are frequently employed in industry, for example at BMW AG Munich, Germany. A general description is given for each approach to allow the derivation of industry needs. A new programming approach was examined based on the identified needs, and it is then compared with the existing robot programming approaches in Section 5.5.

5.2.1 Conventional Online Teach-In Programming

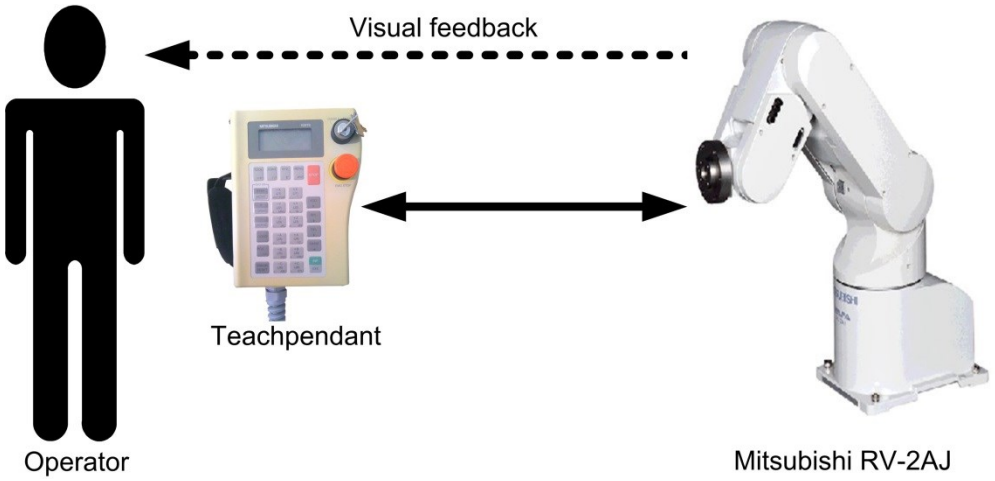


Figure 12: Online teach-in programming.

Conventional online teach-in programming is carried out within a real robot cell without any preparation. However, some robot programmers attempt to create the program structure beforehand to speed up the programming task, and to minimize the production downtimes. Nevertheless, this programming approach is often used when the expected production downtimes are acceptable and all physical parts are available. This approach may result in high production downtimes, and leads to high costs. All work objects have to be available, and thus robot programming may not commence until these objects are physically available. In contrast, this approach is simple, and has been approved and widely accepted. It may be cost efficient when downtimes are acceptable, that is, when robot programming is performed during regular production breaks.

5.2.2 Offline-Programming Amended by Online Teach-In

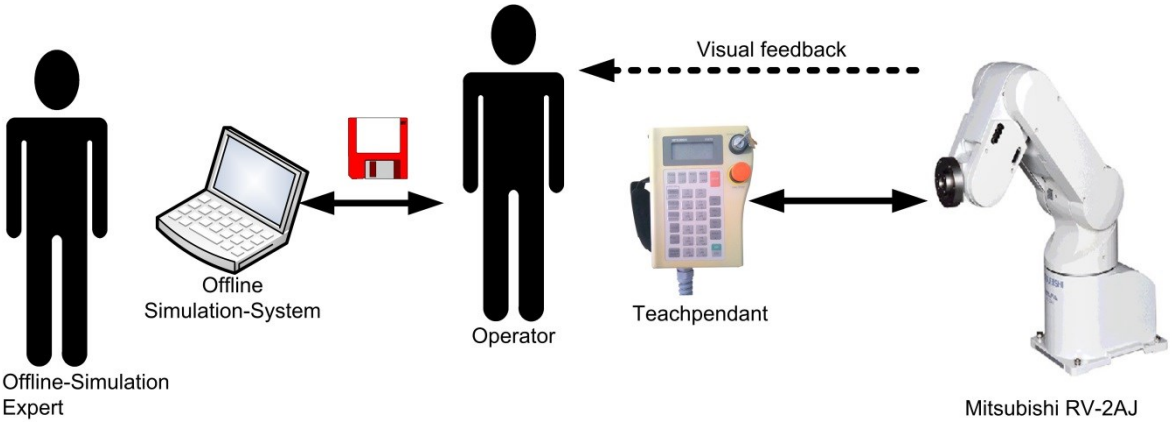


Figure 13: Offline-programming amended by online teaching.

Offline-programming amended by online teach-in utilizes Computer-Aided Software Engineering (CASE) tools to simulate the robot cell beforehand. All work objects have to be modelled to enable offline robot programming. Models of work objects often exist before the real prototype, and they may be adopted during offline programming. Offline programming tools are usually complex and time consuming. To produce high-quality results, the operator is required to be experienced in the use of both, CASE tools and online robot programming.

The online programmer modifies the programs created offline within the real robot cell to compensate for inaccuracies. If the offline robot program is not sufficiently accurate, or if the program structure does not satisfy the online programmer, the entire robot program is often created manually without the use of the offline program. This results in duplicate costs for both offline and online programming of the whole program. Nevertheless, this approach is mainly approved in industries because of the generally shorter production downtimes, even though greater capital is required for robot-programming investments.

5.3 Identification of Industry Robot Programming Requirements

An up-to-date industry requires a modern production system which is able to combine and support flexibility, high-speed and optimization (International Federation of Robotics, 2005); the overall production time available must be maximized to guarantee the highest productivity possible.

The high level of complexity of typical robot-programming tasks for human operators has to be considered; consequently, the robot application-software presented in this study takes over the most complicated task, which is robot motion planning. The remaining manageable tasks which are related to the given mission, e.g. spraying, handling and painting, remain the responsibility of the operator. For example, in a handling mission, the operator provides information about what the robot has to do, e.g. placing objects in specific positions in a specified order, while the online robot software application knows how to control the robot.

Modifications to the existing industrial environment in order to execute the robot programming software should be minimized. In addition, permanently installed hardware and software should not be required.

Supported online programming must be fast and flexible to reduce possible production downtimes. The generated trajectories must conform to the given requirements in terms of quality, such as the smoothness and shortness, and the possible speed of the robot movement.

Physical production parts and fixtures are often not available during online robot programming, and the support system must therefore handle such situations to permit its use.

Nevertheless, robot programs may be modified manually during their lifecycles due to changes that may occur during production. Those robot programs are usually stored as robot program files in a specific robot programming language on the robot controller. Therefore, the generated programs must be readable and maintainable. The proposed method helps to generate such robot programs, and it is therefore easy for these programs to be manually changed by the human operator.

Using the robot application-software presented here, there is still some non-productive time, but unlike previously reported approaches, this is mostly achieved automatically, and therefore rapidly. As such, the actual cell-learning time is minimal, and consequently, offline systems become unnecessary, leading to reduced costs for the offline preparation of robot programs.

5.4 The Proposed Enhanced Online Robot Programming Approach

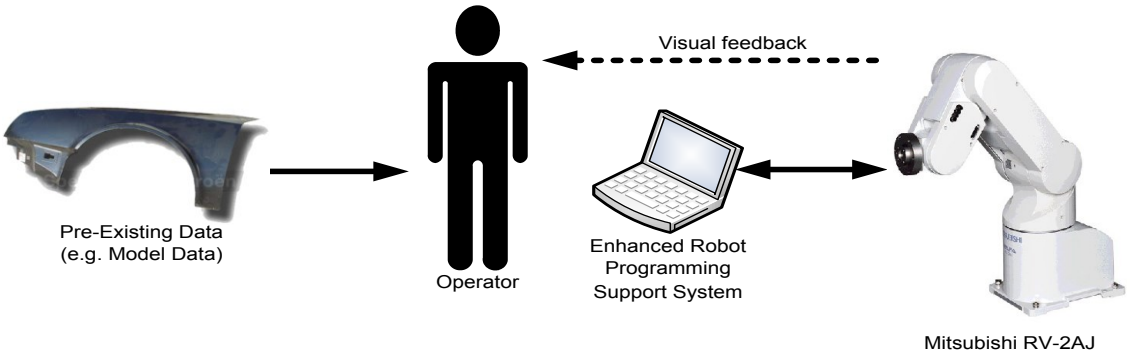


Figure 14: The enhanced online robot programming approach.

The main disadvantages of offline programming are the investments that are required for programming within the simulation system, including the required skilled operators, computers and infrastructure. Therefore, online programming was further studied, leading to the combination of online and offline programming properties. This required an expert

support system that is able to support the operator in robot programming. To enable online robot programming, it should be simple to use and efficient. The support system is required to lead the operator through the required steps to produce high quality robot programs. The approach has to combine the flexibility of online programming and the speed of offline programming. Additional aspects, which include a simple integration into the existing environment, short production downtimes and high quality results have also had to be considered. These aspects required a complete system solution, specialized path planning and robot programming algorithms.

The proposed support system is used within the real robot cell. Changes within the environment or to the equipment are considered immediately. The turnaround time to produce robot programs with such changes is shorter compared to offline programming.

In offline programming, small changes are often made directly online, while the corresponding offline simulation remains unsynchronized to the real production cell. Then, changes to the robot program within the simulation system often require an additional task to merge the robot program with the simulation. This task requires special skills and is not reliable. Because the proposed system eliminates the simulation, this aspect is no longer relevant.

A single robot operator is required to perform the online robot-programming task without the need for any special skills. The available CAD data information is utilized to speed up the automatic programming procedure and to enable the use of model data. The system is semi-autonomous, takes over the complicated low-level tasks, and leaves the high-level tasks to the operator. This approach is simple and cost effective. Without the need for offline programming the company no longer needs the offline programmer, the required hardware and infrastructure. Nevertheless, the integration of the approach into the existing offline-programming systems may also be possible to simplify the offline robot-programming task. The online integration is also helpful when robot program inaccuracies are to be corrected.

This leads to fewer investments for skilled online and offline programmers, rendering offline programming unnecessary. In the automotive industry, offline programming may take up to several weeks. For example, the offline programming of a single robot cell with two robots, each of which is installed on a conveyor for a painting application requires about 10 person-days for offline simulation, 1 day for online programming, and a few days

for CAD data preparation. The cost incurred by ten person-days of a skilled online programmer is about 7000 EUR plus the cost for equipment, infrastructure, offline simulation systems and CAD data preparation.

5.5 Comparison of Programming Approaches

A comparison of the presented robot programming approaches and the required robot programming steps is stated in Table 1. Only the first and last programming approaches omit offline programming, which was identified as the main research objective.

No.	Programming Approach		Steps			
1	**Online Teach-In Programming**		Online-programming within the real cell			
2	**Offline-Programming Amended by Online Teaching**		Offline simulation	Creation of the offline robot programs	Uploading of the programs into the real cell	Manual amendment of the robot program
3	**Supported Programming**	**Offline integration**	Offline simulation	Run the assistant within the simulated cell	Uploading of the programs into the real cell	Manual amendment of the robot program
4		**Online integration**	Offline simulation	Creation of the offline robot programs	Calibration of the cell	Run the assistant within the real cell with simulation data
5		**Enhanced online programming**	Optional preparation of data, that is, robot kinematic or model data	Start the assistant within the real cell	Calibration of model data, teaching of the locations	Run the assistant in the real cell

Table 1: The robot programming scenarios.

Approach 1, online teach-in, has already been evaluated as being insufficient with respect to production downtimes for high-volume production.

The second approach requires high investments but it can be applied to reduce downtimes of the production system.

It was assumed that offline integration, approach 3, would help the offline programming expert to generate suitable trajectories automatically, while built-in special features of the simulation tools are still applicable.

An online integration would take the results of the offline-programming phase to modify the generated program automatically in approach 4. This may simplify the process of amending the online programming, although tool development costs that are incurred may reduce its benefit.

Although the integration of enhanced programming into existing programming approaches would be beneficial, research has focused on the programming approach 5, enhanced online programming, while the remaining approaches may be researched in future.

An evaluation considering the previously defined requirements has produced the results in Table 2 for high-volume production. The summary column also supports the enhanced online programming approach.

No.	Programming Approach	Costs	Production Downtimes	Integration	Quality of Programs	Handling of model data	Usability	Summary
1	**Online Teach-In Programming**	-2	-2	+1	+1	-2	-2	**-6**
2	**Offline-Programming Amended by Online Teach-In**	-2	+1	0	+1	+2	-1	**+1**
3	**Offline integration**	-1	+1	0	+2	+2	0	**+4**
4	**Online integration**	-1	+1	+1	+2	+1	+1	**+5**
5	**Enhanced online programming**	+2	+1	+2	+2	+1	+2	**+10**

Table 2: Comparison of scenarios.

(+ positive - negative o neutral)

5.6 The General Design of the Enhanced Online Programming System

A general overview of the integration of the enhanced online robot programming support system software into the system is given in Figure 15. The system is connected to the robot system, receives input from the environment and the operator, who also creates a mission plan with the support system which in turn generates a robot program file.

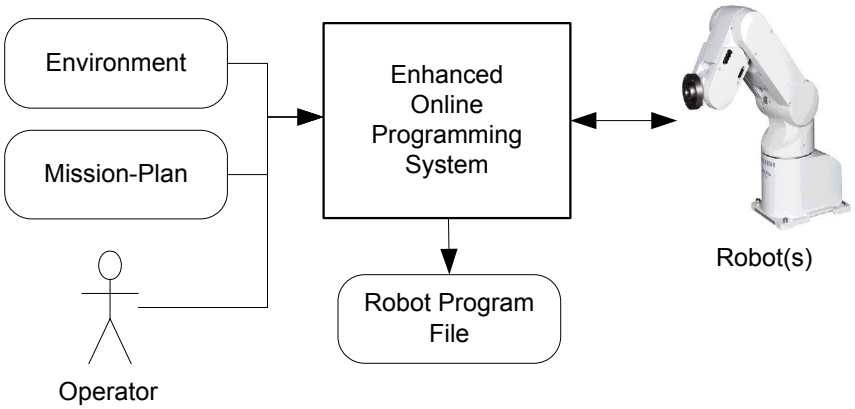


Figure 15: Overview of the enhanced online programming system.

Model-based and sensor-based information were considered to capture the environment of the robot within the system. Those sensors include vision systems, input devices and tactile sensors, which are often used for path planning and control of robots. The robot, work objects, and the obstacles are available within the robot cell. Model data may also be utilized when the physical objects are not available. The logical diagram in Figure 16 shows a typical system architecture and the robot control loop, which consists of the sensor input, actuator output and control functions. Those control functions were implemented as mission and motion planners, based on a world model that stores the in-memory model of the environment.

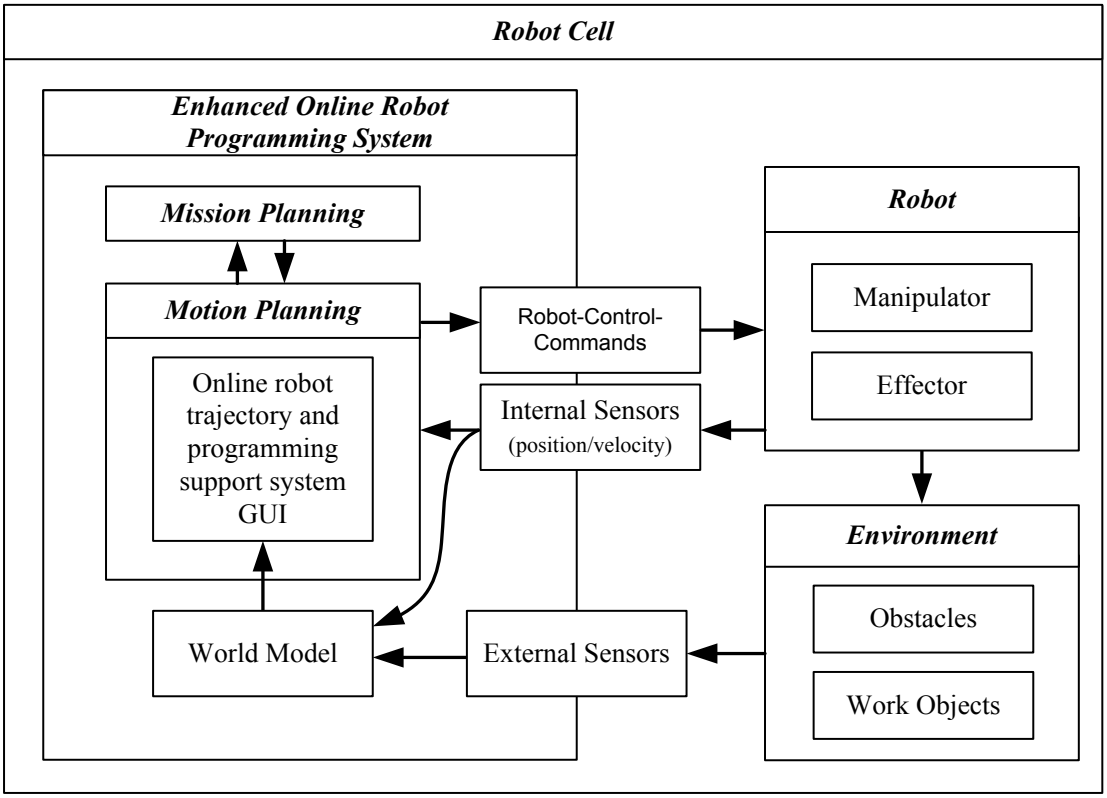


Figure 16: Path planning system: a logical view.

The overview in Figure 17 shows the interconnected system components and devices. The proposed support system is executed on a personal computer which is connected to the robot controller via an Ethernet or serial connection, depending on the robot type and its communication capabilities. In addition, a teach pendant and the robot are connected to the controller. A vision system, a pointing device, and a joystick are plugged into the personal computer. Model data may be imported from files. The hardware and devices are introduced in detail in Appendix B.

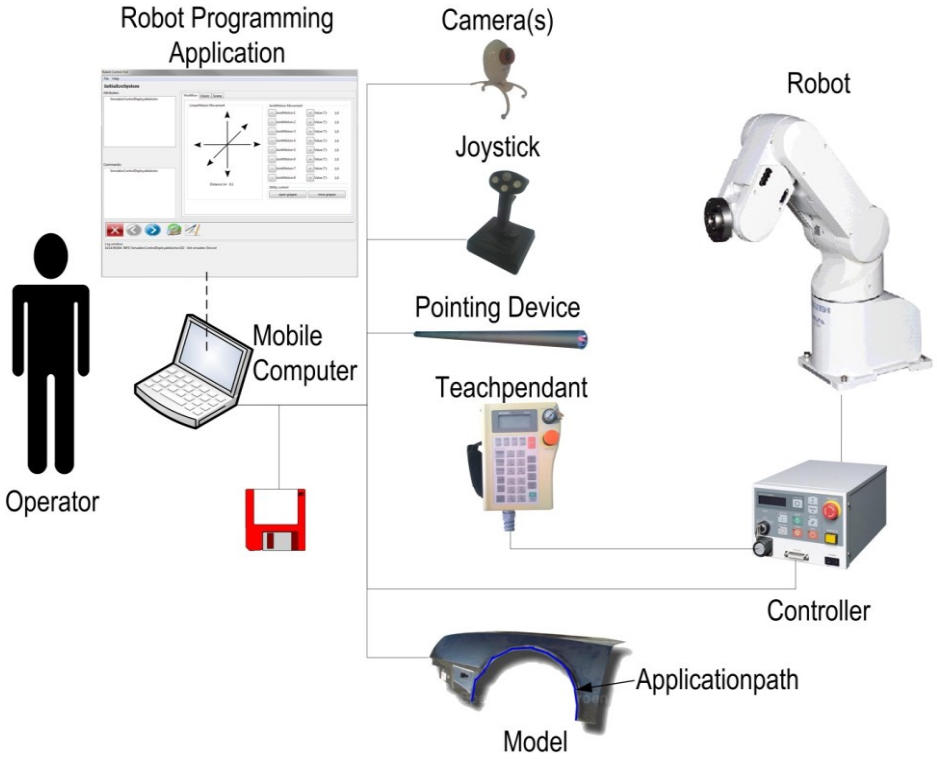


Figure 17: System overview of parts and devices.

5.7 Summary

The development of large software systems requires a structured and homogeneous development strategy to cover aims like reusability, maintainability and testing. This should be accomplished using a model-driven development toolchain. The toolchain should allow the integration of hardware devices such as robots, joysticks, mice, keyboards and pointing devices. Artefacts produced by other tools and toolchains, such as Matlab/Simulink, should be integrable by dynamic link libraries.

The HMI should be simple and easy for inexperienced users to use. It should control all parts of the software system including start and stop procedures, installation procedures and life-cycle management of the connected software components. The GUI should be easily extendable.

The world mode stores a model of the environment, especially the robot cell and the working space of the robot. Information in the form of CAD and robot joint-space data should be handled. Additional requirements are the access delay times to the stored information and the storage size in memory. The information input should be fused to overcome inaccuracies of the data and to provide cohesive information.

The mission planner is responsible for storing the mission data provided by the HMI with several input possibilities. Each mission consists of a start and a target location, and may have multiple application trajectories with application data such as for painting and gluing. Those application trajectories should be connected to control the robot from the start to the target position, including all application trajectories in a path-length optimized manner. Known algorithms for the travelling-salesman-problem (TSP) should also be considered.

The path planner should be controlled by the mission planner, and should create a trajectory with given start and goal positions. Real robot control should be considered to direct the robot from the start to the target position. Inputs from the operator and the sensors should be possible during trajectory planning to incorporate collision indications. The process of planning should also be fast, and the planned trajectories should have a short trajectory length and the generated program should be readable, changeable and similar to those that are manually programmed. Virtual objects should also be considered.

Vision should be incorporated using webcams to recognize the pointing device and the robot-tool-centre-point. Further developments of image processing algorithms using specialized tools such as Matlab/Simulink should be enabled.

A robot model is used throughout the software system. It should provide forward and inverse calculations of the robot kinematic of the Mitsubishi RV2-AJ robot. Those calculations should be based on the ideal, theoretic geometry of the robot.

6 Investigation into a Probabilistic Data Fusion World Model

Path planning is based on data of the physical environment, as illustrated in Figure 18. Information of the environment was retrieved with internal and external sensor perception amended by pre-existing model data and stored within an in-memory model, the world model. It is a hierarchically structured data storage which saves position and collision information. A position can be given either in Cartesian (position and orientation) or robot joint space (with n joints of the robot). In addition, model data has to be stored as well. The presented probabilistic data fusion world model is the data basis for the enhanced robot programming system and it is illustrated in Figure 18 as 'World Model'.

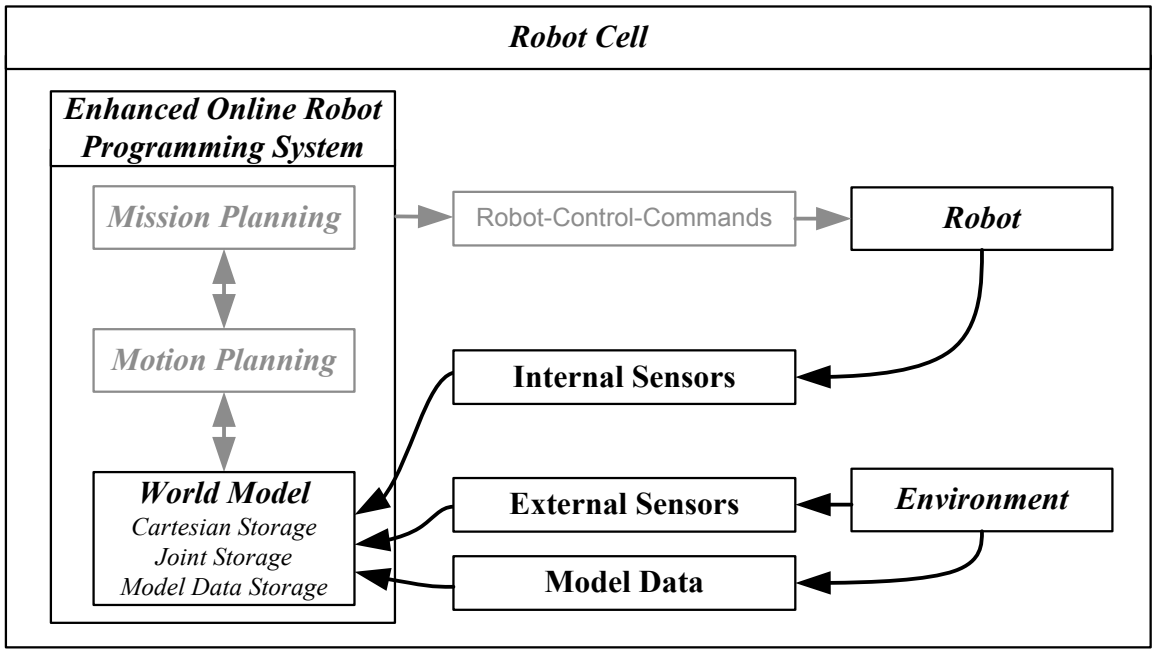


Figure 18: The logical view of the path planning system
with the highlighted flow of sensor information.

The Cartesian storage was realized by a linear octree which was introduced by Gargantini (1982b), and detailed in Section 6.1. The robot joint space positions are stored in a specialized hierarchical binary tree structure, which is presented in Section 6.2. Both the octree and the joint position storage are able to deliver information with a specified level of detail. Sensors such as vision systems, ultra-sonic detectors, and laser-distance measurement systems can be employed to retrieve dynamic information. The proposed system is equipped with a specific button for the operator on the control panel and a joystick button to indicate collision points. The model data storage was implemented using a Java3D scene graph, which is presented in Section 6.3. The model data was retrieved utilizing CAD drawings of the working-cell construction process.

In general, real robot applications have demonstrated that sensors may deliver wrong information (Hall and Llinas, 1997). The world model combines the different data sources using a data fusion architecture. It includes sensor abstraction, algorithms and architectures (Hall and Llinas, 1997), and was implemented as a voting system.

The data fusion architecture presented in Section 6.4 filters the data sources through a simple moving average (SMA) filter and incorporates the reliability of the data sources. A value is defined for each data source to reflect the reliability. Thus, the averaged weighted sum of the sensor values was applied to deliver cohesive information.

The vision system presented in Section 6.5 both delivers information about the environment and interprets the markers presented in Subsection 6.5.3. However, object recognition is a major problem in path planning because of the sparseness of information. A solely vision-based recognition system may not be capable of delivering enough information within an industrial environment, and model data was incorporated into the world model to utilize additional data, although models are often inaccurate.

Results obtained contributed to a journal publication and a conference paper. This chapter corresponds with objective three.

6.1 Cartesian Position Storage

A linear octree presented by Gargantini (1982b) was implemented to store spatial coordinates of the robot environment. The implemented octree is a region octree type. Compared to conventional methods for storing octrees, where the access delay time to certain subdivisions is exponentially increasing (8^n) with the level of accuracy n, special properties of a certain index assignment scheme were utilized, and provides linearity in terms of the access delay time to cells of an arbitrary accuracy. The linear octree allows high-speed access to the cells lying on the movement path of the robot. Finer resolutions of the octree may be reached by more subdivisions, which increases the relative speed (cells per second) of the robot and thus requires shorter access delay times. The use of an octree structure significantly decreases the required storage space, since only fully- and partly-occupied cells are stored. A detailed description of the properties and structure has been presented in various papers (Bhattacharya, 2001, Chang, 2001, Frisken and Perry, 2002, Gargantini, 1982b, Globus, 1991, Gran, 1999, Hwang *et al.*, 2003, Kitamura *et al.*, 1995, Mahler, 2003, Merkle, 2004, Payeur, 2004, Payeur *et al.*, 1997, Peng *et al.*, 2005, Samet, 1994, Schrack, 1992).

The implemented linear octree is initialized with its octree size and accuracy. The required number of subdivisions is automatically determined. Using this representation, the encoding, decoding and determination of adjacent voxels within a specified radius are implemented basic operations that can also be found in literature (Bhattacharya, 2001, Samet, 1994, Schrack, 1992).

6.1.1 Index Assignment

A linear octree stores points using equally sized cubic cells. Each cell represents an element (I, J, K) of a spatial array. The three dimensions also represent the coordinate system normalized to integer values, which denote the number of cell steps in each direction. An octree with two subdivisions is illustrated in Figure 19 to illustrate the basic concept of the index assignment. The accuracy of the octree cell grid increases with the number of subdivisions.

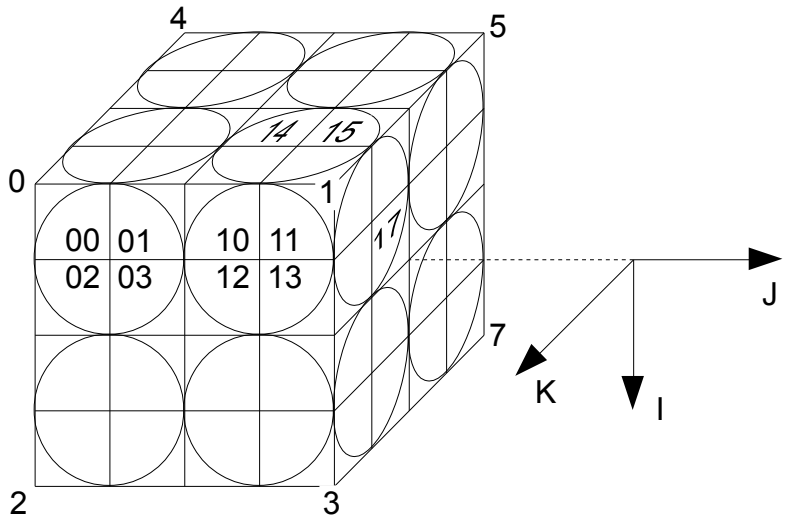


Figure 19: A linear octree with two subdivisions.

The indexing scheme is recursive from the root to the child cells. Child cells inherit the index from their parent voxel and extend it by one digit. The cells may also be represented in two dimensions, as illustrated in Figure 20, where the cells may be either empty, partly or fully occupied.

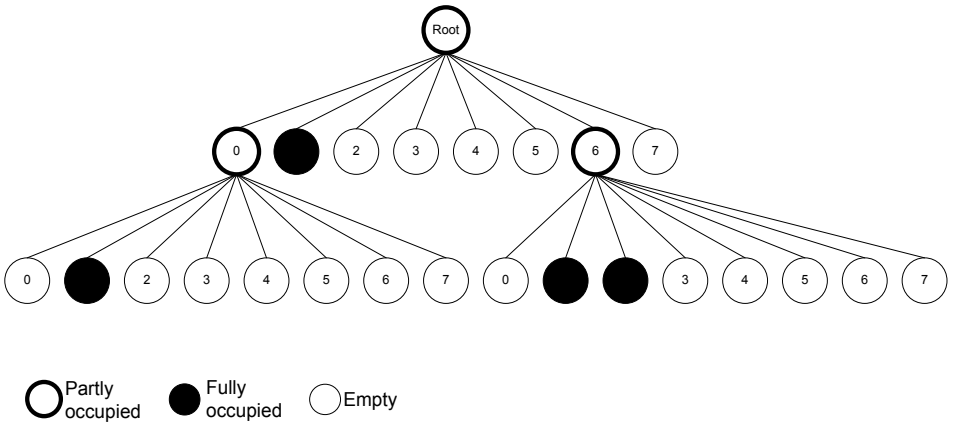


Figure 20: Octree data structure representation.

Thus, for the example above, the cells were stored in an array with the octal code indexing scheme entries as $\{01, 10, 11, 12, 13, 14, 15, 16, 17, 61, 62\}$. Because of the strict order of the indices, a fully occupied parent cell can be combined by encoding it with X. The array has therefore been shortened to $\{01, 1X, 61, 62\}$, and is denoted as a mixed-octal representation of octal digits $\{0, .., 7\}$ and X. Only fully and partly occupied cells of the octree need to be stored. From the left to right, the octal digits within those indices determine the path from the root to certain leafs of the octree, respectively.

6.1.2 Neighbour and Parent-Child Relations

A position is added to the octree by converting it to the octree indexing scheme and adding the cell. Each cell may store additional information, for example to indicate the occupancy probability. Each parent of an added voxel is created with the correct occupancy value derived from its children. However, if the parent already exists with a collision probability value, the highest collision probability of its children is applied. In this way, a parent cell always has the highest collision probability value of its children. When a non-existing voxel is selected by neighbour relationships, this neighbour will inherit the collision probability of the next existing parent node.

Each voxel has neighbour relations to adjacent voxels if they do not exceed the boundary of the robot world, that is, the borders of the root cell. Neighbours exist in perpendicular and diagonal directions at each subdivision level.

In a uniform grid, the transition between cells may be considered to occur at edges within a graph. This may be utilized to find the shortest path from a start to a goal cell, for example with the A* search algorithm (Likhachev *et al.*, 2005, Russell and Norvig, 2002).

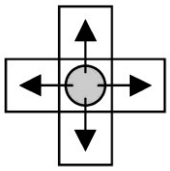


Figure 21: Four neighbours.

The neighbourhood relationships define the connectivity. The Manhattan metric in Figure 21 defines four neighbours in two-dimensional space. The chessboard metric in Figure 22 defines eight neighbours, which are also in diagonal directions.

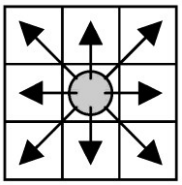


Figure 22: Eight neighbours.

The robot path-planning scenario demonstrated in Figure 23 results in a wrong connectivity since the robot may always have physical dimensions, and therefore, direct diagonal movements through the cells P to Q (left) have to be forbidden, although it is mathematically correct.

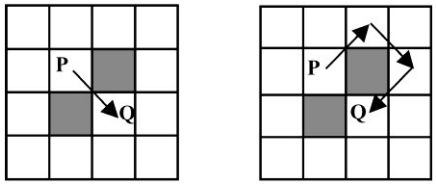


Figure 23: The problem with eight neighbours.

In three-dimensional space, 26 directions are possible from the middle cell, leading to Figure 24. The special case demonstrated in Figure 23 also applies in three dimensions.

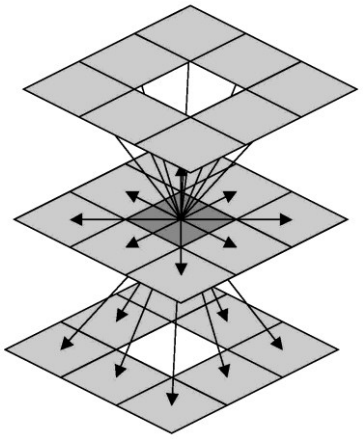


Figure 24: Spatial space neighbour relationship of an octree cell, shown by the arrows.

6.1.3 Digitalization of the Robot Environment

The octree midpoint was defined to the robot base position, which is shown in Figure 25, and which may be arbitrarily positioned in the world space. Therefore, points in the world space have to be converted to the local coordinate system of the octree. This was accomplished using a simple shift operation since the orientation of the world and local coordinate systems are identical.

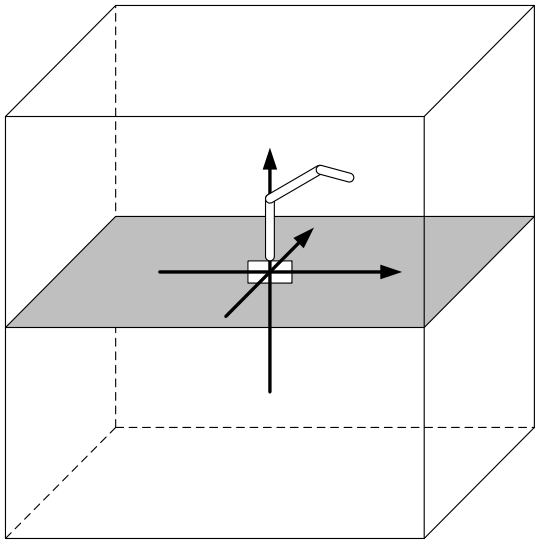


Figure 25: The robot environment and relation of world and octree representation.

Cells may be represented by their world coordinate, (I, J, K) coordinate or index scheme representation. All types may be converted into each of the other types, although the conversion from world to (I, J, K) or index scheme representation leads to a loss of accuracy. The reason for this is the fixed voxel sizes and the defined octree accuracy.

The octal point class is a sophisticated and intelligent data structure that was used internally. This class provides a wide scope of knowledge about its environment, and encapsulates a representation of the octal number as array structure. Within this structure, child relations of the encapsulated points and all its neighbour relations in every direction are stored by their respective octal representations. The necessary calculations were executed during the creation of the octal point to minimize computational costs. The neighbour relations are stored for all neighbours independent from their existence.

6.2 Robot Joint Position Storage

The octree stores spatial Cartesian coordinates, but a robot arm position is an n-dimensional vector of joint angles. A single Cartesian world coordinate may be reached using multiple robot arm positions. Collisions of the robot arm with obstacles may occur anywhere at the robot arm. Therefore, the whole robot arm position (not only the Cartesian world position) has to be considered within an n-dimensional storage system, where n is the number of robot arm joints.

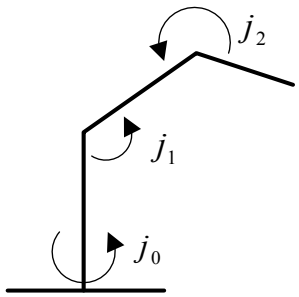


Figure 26: Example robot.

A high-performance and memory-efficient storage system was implemented, and allows information to be requested on a specified accuracy level. Each position consists of the angle values of each robot joint, as illustrated in Figure 26. Robot arm joints are usually limited to a specific range, which is given by a minimum and maximum value. An example is given in equation (1). Mechanical sensors are often installed to check the robot arm ranges.

$$\begin{aligned} & j_0 : [-200 \dots 200] \\ & j_1 : [0 \dots 100] \\ & j_2 : [-180 \dots 180] \end{aligned} \tag{1}$$

The joint angle ranges may also overlap, which is the case here for j_0. The storage system implements a binary tree with an accuracy that is defined by its depth t. The absolute angle range is subdivided by two on each depth level, as illustrated in Figure 27.

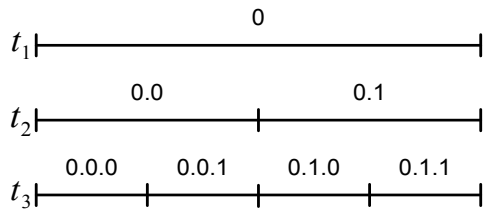


Figure 27: General joint angle binary tree for a joint j with depth $t = 3$.

The depth t for each joint n is calculated based on the required minimum accuracy G and the absolute joint range d in equation (2). t is rounded up to a natural number, which increases the accuracy G.

$$\mathrm{G} = \frac{d}{2^{t_n}} \tag{2}$$

Therefore, the depth t for a joint n is calculated using the absolute range length in equation (3).

$$\mathrm{t_n} = \left\lceil log_2\left(\frac{d}{G}\right)\right\rceil \tag{3}$$

For joint j_0, an accuracy of $G = 0.1°$ and an absolute range length $d_{j_0} = \|j_0\| = 400°$, $d_{j_1} = 100°$ and $d_{j_2} = 360°$ may be given as an example in (4).

$$\begin{aligned} \mathrm{t_{j_0}} &= \left\lceil log_2\left(\frac{400°}{0.1°}\right)\right\rceil \approx \lceil 11.96\rceil = 12 \\ \mathrm{t_{j_1}} &= \left\lceil log_2\left(\frac{100°}{0.1°}\right)\right\rceil \approx \lceil 9.96\rceil = 10 \\ \mathrm{t_{j_2}} &= \left\lceil log_2\left(\frac{360°}{0.1°}\right)\right\rceil \approx \lceil 11.81\rceil = 12 \end{aligned} \tag{4}$$

An illustration may be given in equation (5) using the calculation of an example point $P_1 = (-180,40,0)^T$ in a binary tree with a simplified example accuracy of $G = 100°$ for the joints $j_0..j_2$.

$$\mathrm{t_{j_0}} = \left\lceil log_2\left(\frac{400°}{100°}\right)\right\rceil = 2 \tag{5}$$

$$t_{j_1} = \left\lceil log_2\left(\frac{100°}{100°}\right)\right\rceil = 0$$

$$t_{j_2} = \left\lceil log_2\left(\frac{360°}{100°}\right)\right\rceil = 2$$

The resulting binary tree position Pos_{tree} of the example positions P_{1-3} are graphically shown in Figure 28.

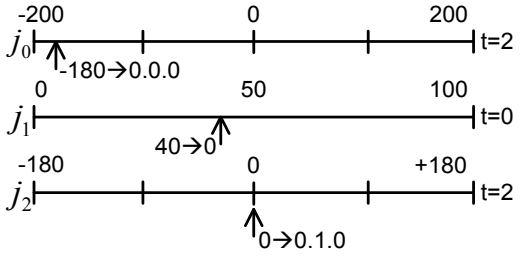


Figure 28: Joint angles binary tree.
(j_{1-3} binary tree for $P_1 = (-180,40,0)^T$ and accuracy $G = 100$)

The resulting binary tree index positions Pos_{tree} for P_{1-3} are stated in equation (6).

$$\begin{aligned} &Pos_{tree}\{P_1 = (-180,40,0)^T\} = [000;0;010] \\ (6)\qquad &Pos_{tree}\{P_2 = (-50,10,40)^T\} = [001;0;010] \\ &Pos_{tree}\{P_3 = (-101,90,80)^T\} = [000;0;010] \end{aligned}$$

Because every position Pos_{tree} represents a collision point with an occupancy value O, these values are stored and updated along the position using the update rule provided in equation (7).

$$(7)\qquad O_{Pos_{tree_{new}}} = max(O_{Pos_{tree}};\ O_P)$$

The external interface to this component defines methods to obtain and store robot arm positions, including their occupancy values in the requested accuracy. If the requested position does not exist within the binary tree, an occupancy value of zero is returned. Positions are stored when they do not yet exist in the binary tree. Existing positions update their occupancy values with the formula given in (7).

Storing joint positions in the presented way reduces the number of joint positions stored per octree cell and allows storing the joint positions in a ‘natural’ way. Thus, joint positions that are near together, and also their occupancy information can be summed up to one binary tree cell. The joint positions are normalized.

6.3 Model Data Storage

Although a Cartesian and a joint storage have already been implemented based on space partitioning, a model storage was implemented to additionally store the modelling elements as objects. The model storage was implemented as a Java3D scene graph (Sun-Microsystems, 2012), which is structured as a tree containing several elements that are needed to display the objects. It can be directly visualized, as illustrated in Figure 30, using an implemented Java3D viewer. The user is able to interact with the viewer using the mouse and the keyboard. Information about the visualized objects can be obtained by clicking on the objects. Storing moving obstacles within the Cartesian storage requires the processing of intensive octree transformations. Therefore, models were stored within the model storage.

The geometric models were imported from files with the Drawing-Exchange-Format (DXF), which is a widely accepted format utilized by many computer-aided design programs. This file consists of pre-existing static model information, which may either be imported into the world model or directly be used within it. The import has been accomplished with the help of Java3D by using collision test methods and storing each position within the world model.

The model information was imported using a rasterization step with a predefined raster size r_s with $0 < r_s < s$. The raster size r_s was set to $r_s = 0.2 \cdot s = 3.126cm$. Although pre-processing was not necessary, it was employed to reduce the running times of the algorithm. Modelled obstacle data does not need to be complete, and it has been employed to add already existing information to the in-memory world model.

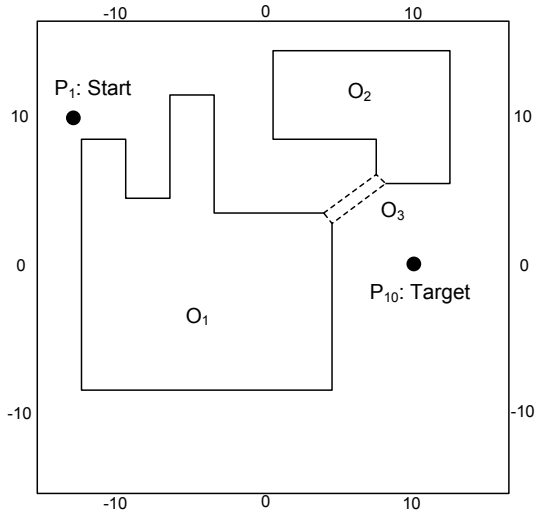


Figure 29: Experimental scenario (2D example in 3D world), with obstacle O_3 being unknown.

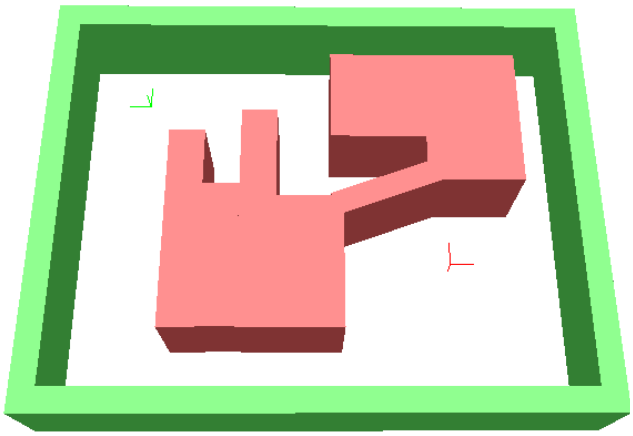


Figure 30: Illustration of the experimental scenario in the 3D world.

The Java3D scene graph also supports collision detection, but only within the visualized frames, and does not detect interpenetrating objects between two frames. The Open-Dynamics-Engine (ODE) physical simulation engine supports collision detection between frames (Smith, 2012). ODE is a free, industrial-quality library that is used for simulating articulated rigid body dynamics in virtual reality environments. It was used for collision detection of basic geometric objects, but collision detection with complex CAD data is only supported at a basic level. The detection can be manually enhanced by implementing the calculation of collision points and vectors. Nevertheless, for this work, the requirements are fulfilled since only basic geometric objects are required. The Java binding ODEJava (Comunity, 2012) was employed to implement a graphics engine to combine ODEJava with Java3D.

6.4 Data Fusion Framework

The world model handles Cartesian and n-dimensional collision positions as well as model data. The employed data sources provide information about the position of the robot, the obstacles and the collision positions. Data fusion was required to acquire consistent information to allow accurate representation of the in-memory world model. Sensors tend to deliver imprecise data, such as the occupation of the robot working space, which is required during the trajectory planning process. The general sensor fusion architecture is represented in Figure 31.

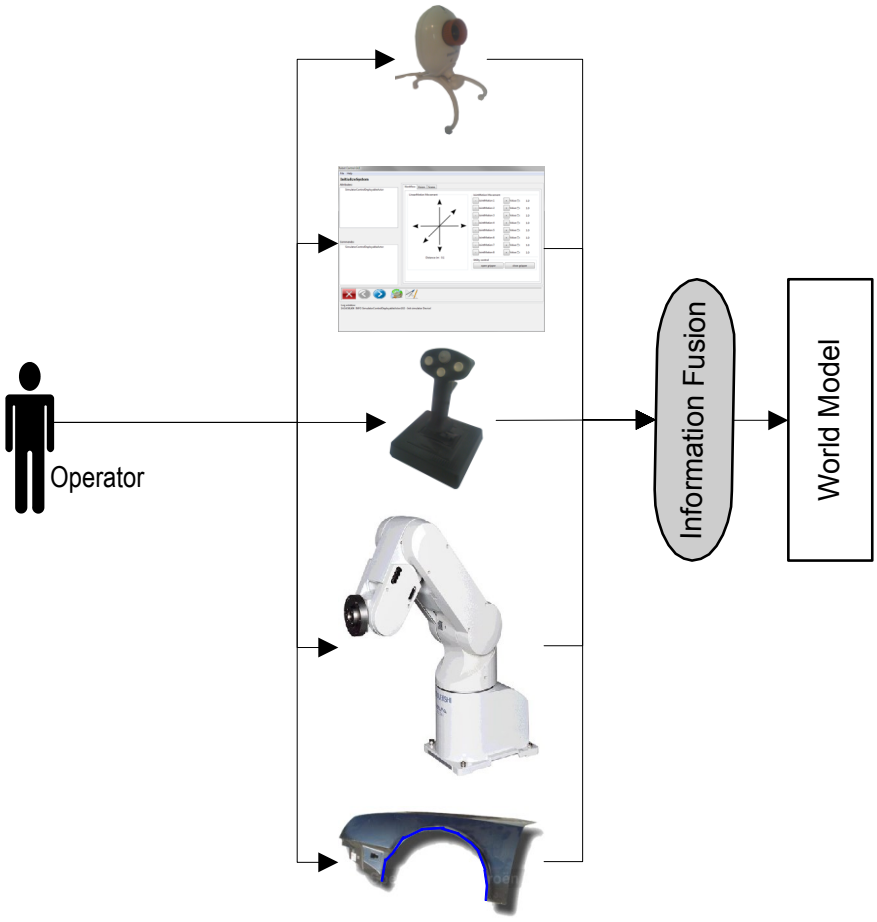


Figure 31:Data sources of the information fusion system.

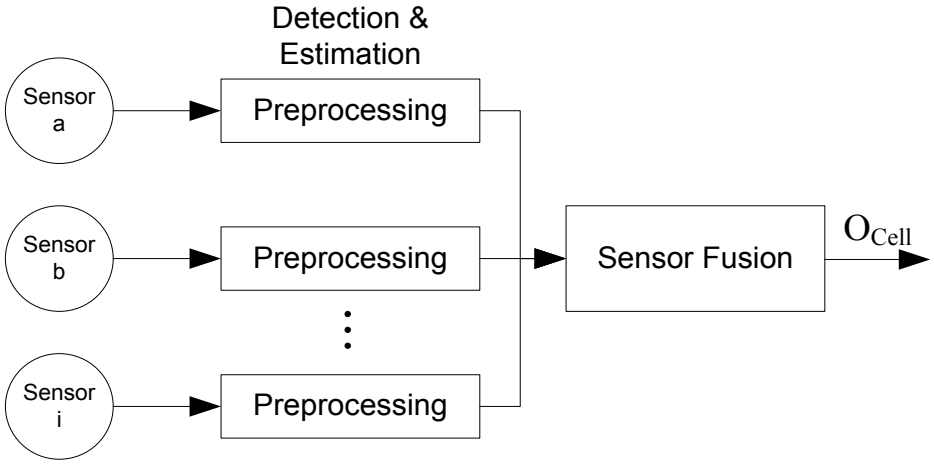


Figure 32: Sensor fusion architecture.

The raw data of commensurate sensors (that is, when the sensors measure the same physical phenomena, such as two visual image sensors) can be directly combined. Unfortunately, the sensors used in this work are not commensurate. Thus, data fusion is required on a higher level.

The interpretation of the raw data of the applied sensors results in a single value, the sensor occupancy O_{Sensor_i}, which is normalized between $0 \leq O_{Sensor_i} \leq 1$. Information about the occupancy is directly obtained using two sensor types, namely the modelled obstacle data and the collision indication button.

Each sensor has a manually defined reliability value, $0 \leq \mathrm{R}_{\mathrm{Sensor_i}} \leq 1$. For example, modelled data may be less reliable than collision indications of the operator. The reliability values of the applied sensors, modelled obstacle data and the collision indication button have been predefined based on experience.

The employed data fusion strategy calculates the averaged weighted sum of the sensor occupancy values O_{Sensor_i} according to their reliability, and applies the history of the so-achieved values O_{Cell_0} with an SMA. The advantage of the applied strategy is the fusion of multiple sensors with different reliabilities by averaging and smoothing of the sensor measurements. The fused sensor values are persistent in the in-memory world model, and are ready for subsequent reuse.

O'_{Cell_0} in equation (8) represents the cell occupancy at the actual time step, and it is the averaged weighted sum of the sensor occupancy values with a given number of sensors n.

$$O'_{Cell_0} = \frac{1}{\sum_{\mathrm{i}=1}^{\mathrm{n}} R_{Sensor_i}} \cdot \sum_{i=1}^{n} \left(R_{Sensor_i} \cdot O_{Sensor_i}\right) \tag{8}$$

The cell occupancy probability O'_{Cell} $(0 \leq O'_{Cell} \leq 1)$ is calculated by the equation given in (9). The history of the cell is considered by calculating the SMA with an order of $m = 3$. The experimentally chosen order of the SMA filter defines the window size.

$$O'_{Cell} = \frac{1}{\mathrm{m}} \cdot \sum_{\mathrm{j}=0}^{-\mathrm{m}+1} O'_{Cell_{\mathrm{j}}} \tag{9}$$

The index ‘0’ always belongs to the actual values, ‘-1, -2 ...’ etc. to former values. Sensor values are centred on the mean for static obstacles, and the lag behind the latest sensor value may therefore be neglected.

Sensor values and the corresponding fused sensor values are illustrated in Figure 33 and Figure 34. Three sensors were measured, with sensors 1 and 3 having a low (false) value in

measurements 3 and 4, respectively. In Figure 34, the fused average values for the cell values were compared with the unfiltered sensor values.

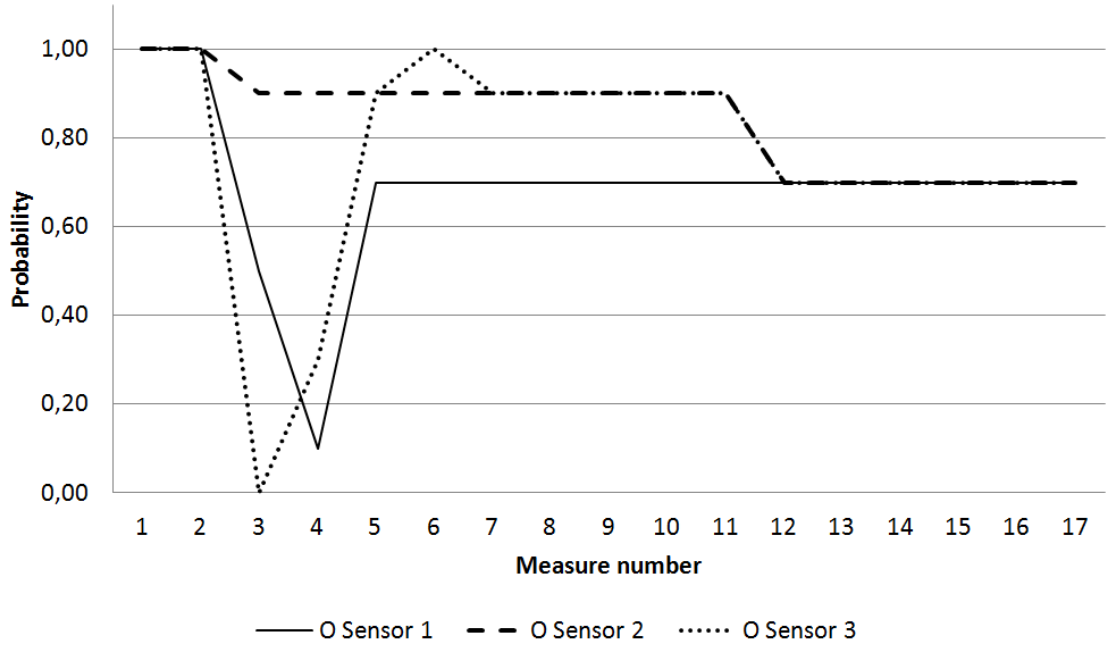


Figure 33: Sensor values derived from real sensors.

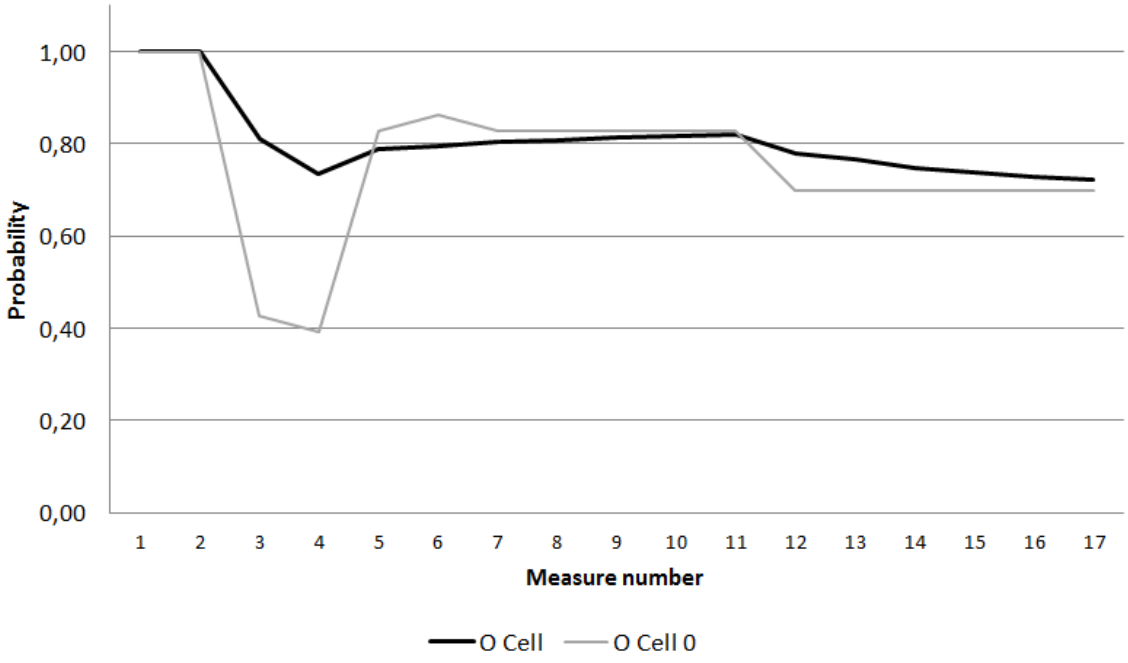


Figure 34: Fused sensor data.

The reliability of the fused sensor data for static obstacles was computed by the similarity of the fused sensor data values, as described in equation (10) and equation (11). Similar occupancies result in a probability of $R_O = 1$, where $0 \leq R_O \leq 1$.

(10) $$R_{O_0} = 1 - \left|O'_{Cell_{-1}} - O'_{Cell_0}\right|$$

(11) $$R_O = \frac{1}{m} \cdot \sum_{j=0}^{-m+1} R_{O_j}$$

Altogether, the cell occupancies O'_{Cell} and the reliabilities R_O of the cell occupancies were applied as a product $O'_{Cell} \cdot R_O$ of those cells, and equation (9) is redefined as equation (12). The impact of the cell occupancies on path planning will be shown in Chapter 8.

(12) $$O_{Cell} = O'_{Cell} \cdot R_O$$

Attention has to be given to the sensor information type, which can be in configuration space or world space. World coordinates are only relevant for the cell occupancy while configuration space coordinates are additionally stored within the cell.

6.5 Vision System

Vision was employed to recognize markers in picture coordinates. The pointing device and the manipulator tool-centre-point (TCP) were equipped with those markers to execute first tests with visual servo control using a neural network. Subsection 8.3.2 is dedicated to the pointing device as part of the human-machine-interface.

Active and passive marker types were evaluated for recognition. It was expected that active ones would deliver good recognition results. Therefore, the luminescence emitter diodes in the visible wavelength range and in the infrared wavelength range were evaluated. The recognition of markers in the visible light range was difficult because of interferences in the background which had to be filtered. Infrared markers showed promise with respect to simplifying the recognition, but the camera required an additional infrared filter to be able to detect only the infrared markers. The tested infrared filters also reduced the intensity of the infrared light range, and therefore required strong active infrared markers. The light emission of the luminescence emitter diodes is often directional for both luminescence-emitter-diode types that emit infrared and visible light. Lampshades were tested to produce a diffuse light source, but did not improve the recognition capabilities.

Therefore, passive markers have been further evaluated, and wooden balls with the colours red, green, blue and yellow delivered acceptable results, even with background interference, which was filtered.

The filter required image stream processing implemented with Matlab/Simulink (see also Subsection 6.5.3) to generate a dynamic link library (DLL). The implemented image stream processing chain is illustrated in Figure 35. The image stream source was a web-camera.

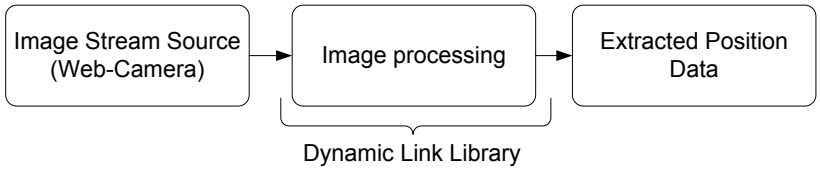


Figure 35: Image stream processing chain.

6.5.1 Colour Recognition

Colour recognition requires an in-memory representation of colours to encode a series of images into an image stream. The representation of colours is defined through the colour space, which may vary according to its purpose, that is, some colour spaces may encode and compress the colour information based on measurements of human colour perception. The colour spaces RGB, YCbCr and HSV are often used by Matlab.

The RGB colour space describes each colour as a combination of the base colours red, green and blue. Each base colour value ranged from 0 to 255. The YCbCr colour-space also has three values, but ranges from 0.0 to 1.0. The Y defines the luma component, and Cb and Cr define the blue-difference and red-difference chroma components. The HSV colour space encodes colours in a cylindrical space, as shown in Figure 36. As hue H varies from 0.0 to 1.0, the corresponding colours vary from red through yellow, green, cyan, blue, magenta, and back to red. As the saturation S varies from 0.0 to 1.0, the corresponding colours (hues) vary from unsaturated (shades of grey) to fully saturated (no white component). As the brightness value V varies from 0.0 to 1.0, the corresponding colours become increasingly brighter.

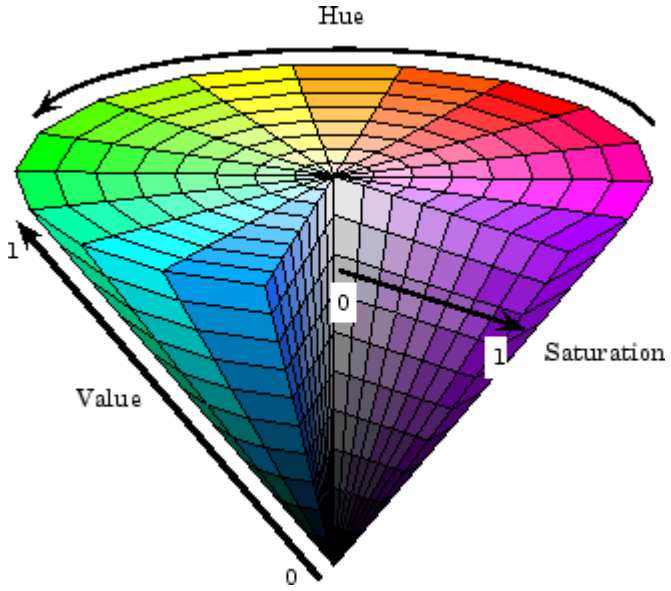


Figure 36: The HSV colour space.
(Source: Matlab documentation)

Therefore, the HSV colour space defines colours using only the hue and saturation. The brightness influences the maximum saturation of a colour, but it was shown that this effect may be omitted when a minimum brightness is achieved.

An implemented colour calibration allowed the definition of the colours to be recognized manually. The definition of the colour area in the hue-saturation space is stored as minimum and maximum values of hue and saturation. This was accomplished using a preview image of the employed camera. An elliptical space was selected on the preview to create an image mask and to crop unimportant image regions. The selected region was analysed pixel by pixel to store the minimum and maximum hue and saturation values.

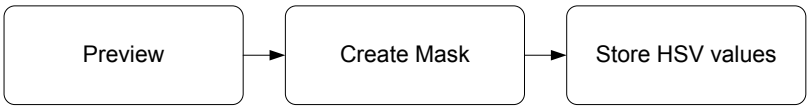


Figure 37: Colour calibration process.

6.5.2 Image Stream Source

The Simulink image acquisition block illustrated in Figure 38 (left block) acquires image and video data streams from devices, such as cameras and frame grabbers, in order to deliver image data within a Simulink model. The block directly previews the acquisition in Simulink, and opens, initializes and configures the resolution of the input device. The output signal is an array with the width and height of the image size in pixels. Each array

element identifies a colour value in the device-dependent colour space, such as RGB, HSV and YCbCr.

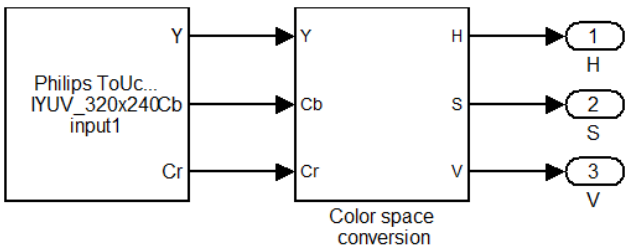


Figure 38: Simulink image acquisition block and colour conversion.

The developed image processing chain uses the HSV colour space. Because the image acquisition block provides the stream in the YCbCr colour space, a colour space conversion was required to convert the image stream from the YCbCr to the HSV colour space.

6.5.3 Marker Recognition

An image stream processing chain was implemented to segment, detect and track the position of markers with a specific colour in an image stream. This processing was performed for red and blue coloured markers. Segmentation was also applied to select regions in the image which comply with the calibrated colour values. Detection recognizes blobs of the selected regions which were further utilized for tracking with a Kalman filter. The tracking block delivers the extracted position data of the markers.

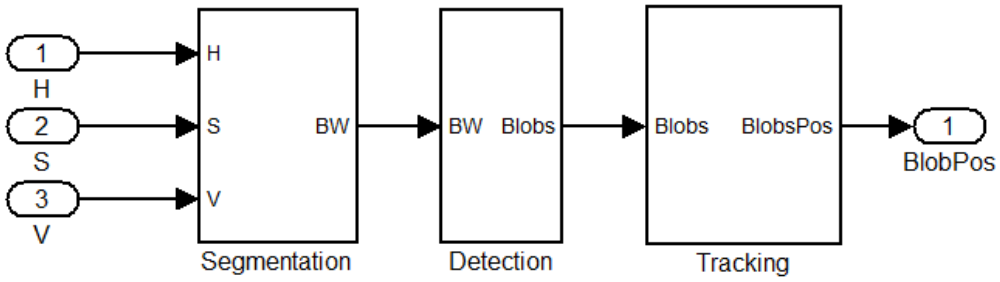


Figure 39: Image stream processing chain.

Segmentation

Segmentation is realized by filtering the images of the image stream regarding their hue, saturation and brightness colour-space component. Each pixel that complies with the calibrated colour component ranges for hue, saturation and a minimum brightness are labelled. Pixel labelling sets labelled pixels in the binary image stream output to 1 and

unlabelled pixels to 0. Pixel labelling leads to a conversion of the coloured image stream to a binary image stream containing a single calibrated colour.

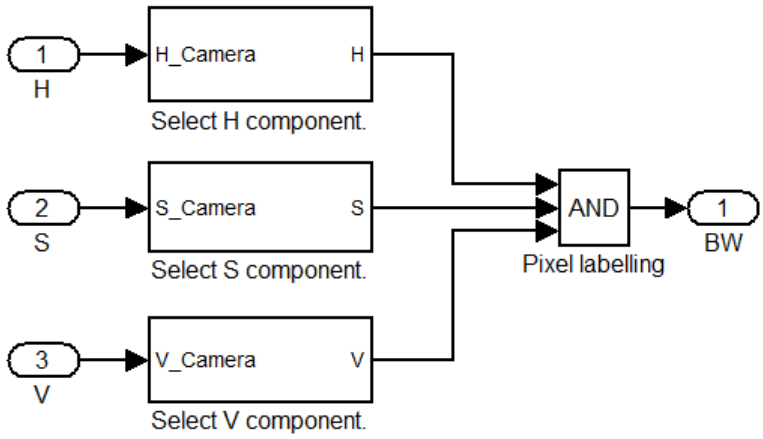


Figure 40: Image stream segmentation.

Detection

The binary image stream (BW) contains noise that is filtered by a median filter. The filtered image stream may still contain gaps within objects that are closed with the closing algorithm. The resulting image stream allows blob analysis to detect objects, for example balls. It takes a given filtered binary image stream as the input, and outputs quantities such as the Centroid, major and minor axis. The Centroid signal is a 2-by-N matrix, where the columns represent the coordinates of the centroid of each blob and N is the number of blobs.

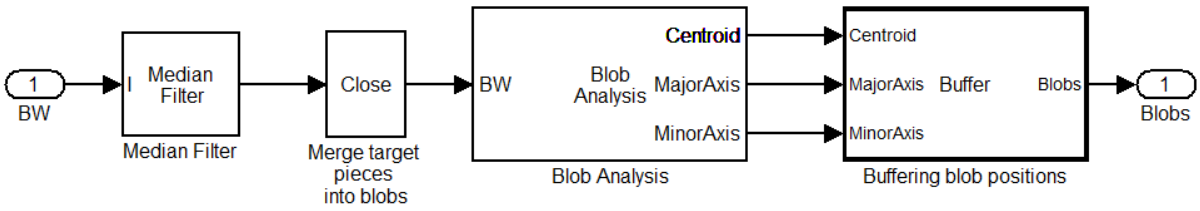


Figure 41: Blob analysis block.

There is still noise in the image stream that leads to false recognitions. Further improvements were realized by utilizing an additional property of the ‘ball‘ markers. Their projection onto the picture plane results in a circular shape from any direction and has been taken into account. The major and minor axes of the blob analysis for each blob were utilized to calculate the circularity $c = \mathrm{MajorAxis}/\mathrm{MinorAxis}$ of each blob, where a value of $c \leq 1.2$ was used as a threshold to indicate the circular shape of the blob.

Tracking

The identified blobs were sorted within the indexer block of Figure 43 to match the blob positions of the previous iteration. This was realized by a distance measurement of each new blob to all previous blobs to find its previous matching blob. Two blobs with the shortest distances are assumed to be the same blob. It was shown that this method is only valid when the movement of the blobs in each frame is sufficiently slow. In addition, blobs were buffered so that missing blobs always keep their last position within a maximum period of 1 second. This smoothed the recognized marker positions, especially when they were not detected in several frames.

The sorted and buffered blob positions were sent to a Kalman filter, which reduces the noise of the measurement data and outputs a vector with position and velocity information in the x and y directions. Only the position information is utilized, and the selector block therefore rebuilds the output vector. The indexer block may also utilize the predicted position output X_prd of the Kalman filter to sort the blobs and to optimize the results.

Figure 42: Blob analysis block.

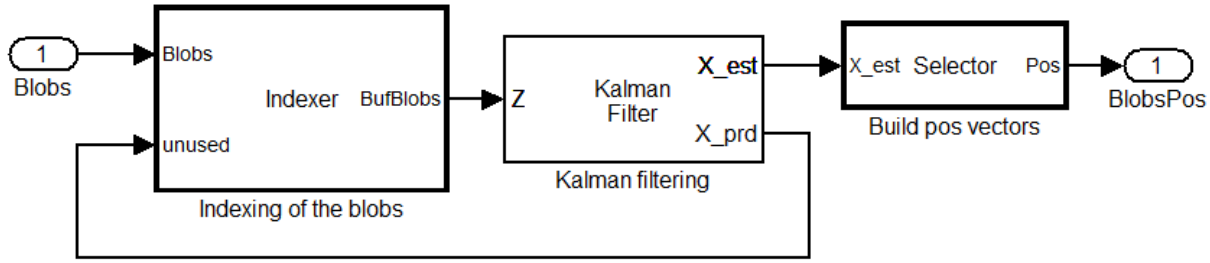


Figure 43: Blob analysis block.

6.6 Summary

This chapter addresses objective three and it presented an in-memory world model that stores fused collision information regarding the collision indication button, the model data and the robot. Collision points may be delivered in the Cartesian or robot joint space, which are both handled by the world model.

The implemented SMA filter for the data fusion algorithm may lead to over-smoothing of the sensor values, and there may therefore be a recognition delay for sudden events. This depends mainly on the order of the SMA filter, which can be set individually for each information source. Important sensors with a high reliability have a low SMA order.

Because the operator uses a collision indication button with a low order of the SMA filter, collisions are always detected.

The image processing chain recognizes markers in an image stream. The colours to be recognized were manually chosen during a pre-processing step. Coloured balls were used as markers to differentiate the markers, e.g. to distinguish the pointing device and robot-arm markers. The implementation in Matlab/Simulink allowed further improvements of the algorithms without any necessary modifications to the remaining software system. The image processing algorithms were compiled into a DLL for system integration. The developed algorithm may deliver false results when the markers are moved too fast or when the markers leave the camera view. Nevertheless, the marker recognition capabilities are sufficient for the implementation of a prototypical robot-programming assistant.

The presented probabilistic data fusion world model was utilized as data basis for the enhanced robot programming system, especially for the path and trajectory planning algorithms. It was established to turn relevant information about the physical environment into a cohesive and processible information source.

The outcome of this chapter was subject for various publications and the addressed objective three has been met.

7 Research of the Robot Kinematics Model and the Robot Control Capabilities

Research into the enhanced online robot programming approach was accomplished using two types of robots. The first type is the industrial articulated manipulator, which is described in Section 7.1, and which is also the intended target system for the enhanced online robot programming system. The second robot type, which is detailed in Section 7.3, is an autonomous mobile robot. The hardware of both robots is described in Appendix B. Simulation of the two robot types and of a free-flying point robot is described in Section 7.5.

The trajectory generation algorithm of the robot manipulator uses a “free flying point robot” in one of the first steps to calculate the motion (see also Section 7.2.3 for the robot model). The autonomous mobile robot can also be seen as a free flying robot in two dimensions and it has therefore been used to test early implementations of the first calculation steps of the algorithm.

The investigation shows that remote control of the industrial manipulator Mitsubishi RV-2AJ and the mobile robot Festo Robotino is possible and has been published at a conference. Forward and inverse calculations with the robot kinematics were analysed. This chapter corresponds with objective two.

7.1 Mitsubishi RV-2AJ Manipulator Control

The industrial articulated manipulator Mitsubishi RV-2AJ is well documented, and communication with a personal computer is possible. Its commercial viability has already been industrially proven in the manufacture of car sub-assemblies, semiconductor memories and other industrial/consumer goods (Mitsubishi-Electric, 2008). The main areas of application are assembly, manufacture, pick & place and handling. The ability to use industrial robots without the need to modify the robot and its controller is important to facilitate its rollout in industry.

A robot control framework described by Kohrt *et al.* (2008) was developed to control the Mitsubishi RV-2AJ robot manipulator and to exchange information such as sensor data and the robot arm position. The framework enables direct robot control, serial/Ethernet connection, robot parameter editing/reading/writing, program uploading and downloading, real-time movement control, robot system backup/restore, external control over user datagram protocol (UDP) and equipment control. The initial configuration of the robot was automated on start-up of the system.

This section discusses the built-in communication modes Real-Time External Control Mode, Controller Link Mode and a Data Link Mode. An additional, extended communication mode was implemented with the Data Link Control Mode. The development was based on the built-in communication modes to overcome the real-time control limitations now being discussed.

The Data Link Control Mode allows bi-directional communication for control commands and sensor information exchange at any time. Usually, the robot system allows the sending of motion commands which have to be executed and finished before the next command can be processed. Therefore, applications such as real-time joystick control of a robot are not possible. This framework overcomes this limitation by installing a communication server on the CR1 controller, which manages the communication to the personal computer.

7.1.1 The Built-In Robot Control Modes

The built-in communication modes Controller Link Mode, Data Link Mode and Real-Time External Control Mode were utilized to create the extended Data Link Control Mode that is described in this subsection.

Controller Link Mode

The Controller Link Mode was used to set parameters, send robot control commands and read the robot status. Receiving status information during movement of the robot and controlling the robot in real-time is not possible. The data is sent in plain text over an Ethernet, and it was therefore possible to monitor the Ethernet communication between the controller and the personal computer. The protocol format for sending commands is shown in Listing 1.

```
[<Robot No.>];[<Slot No.>];<Command> <Argument>
```

Listing 1: Command protocol format.

Each command is followed by a message that is sent by the controller, and contains status information and the result. Table 3 states the pattern of the returning status information, where each star stands for one digit. The framework verifies the correct transmission of the robot command with the returned status information.

Commands	Contents
QoK****	Normal status
Qok****	Error status
QeR****	Illegal data with error number
Qer****	Error status and illegal data with error number

Table 3: Status of sent commands.

Real-Time External Control Mode

Real-Time External Control of the robot was employed for direct robot control, where the trajectory is calculated manually by the personal computer. The real-time external control mode is based on the UDP networking protocol (Flanagan, 2002), which is a simple and low-level network communication protocol that sends arrays of bytes over the network. Even though UDP transmission is not reliable, the low protocol overhead allows quick datagram transmission. The sending and receiving of packets is monitored, and a timeout exception is triggered if the communication does not meet the cycle-time requirement. Runtime is crucial, since every communication cycle has a period of 7.1ms (Mitsubishi-Electric, 2002a), depending on the robot hardware. It was discovered that a plain Java port is not capable of communicating with the robot controller in the required time, and leads to a loss of UDP packages. Thus, movement of the robot was no longer smooth. A DLL written in C solved the cycle-time issue. This library may also be used in Matlab/Simulink to build a ‘hardware-in-the-loop‘ low-level robot control application. However, the library was not utilized because on the one hand, the tested Java robot control component had already been implemented and tested, while on the other hand, the component required to execute the DLL function had not yet been completed.

Data Link Mode

The Data Link Mode connects a controller to a personal computer. Usually, it is utilised to send robot status information from internal robot sensors to the receiver.

7.1.2 Overview of the built-in Communication Modes

The three built-in communications modes are outlined in Table 4, and in Table 5, actual case results are identified to highlight their usage. Because it was not possible to send control commands and information requests over one connection, a second connection was always required to receive actual status information during robot motion.

Mode	Phys. Layer	Command type	Feed back type	UC1	UC2	UC3	UC4	UC5	UC6
RTEC	ETH	SDO	SDO	X	-	-	-	-	-
DL	ETH	SD	SD	-	-	-	X	X	X
DL	RS232	SD	SD	-	-	-	X	X	X
CL	ETH	Robot command		-	X	-	-	-	-
CL	RS232	Robot command		-	-	X	-	-	-
CL	ETH	Robot program		-	-	-	X	-	-
CL	RS232	Robot program		-	-	-	-	X	-

(RTEC – Real Time External Control; DL – Data Link; CL – Control Link; ETH – Ethernet; SDO – Serialized Data Object; SD – Serialized Data; UC – Use Case)

Table 4: Built-in robot communication modes.

Use-case	Description
1	Direct robot control over Ethernet with feedback. Either the mentor or the path planning system may move the robot manually. No controller calculations are involved.
2	Robot operation with single movement commands over Ethernet. The controller calculates the path. Feedback data may be retrieved by Ethernet connection after finishing movement.
3	Robot operation with single movement commands over serial port. The controller calculates the path. Feedback data may be retrieved by serial port connection after finishing movement.
4	Robot operation with robot programs over Ethernet. The controller calculates the path. Feedback data may be retrieved either by Ethernet or by serial port connection.
5	Robot operation with robot programs over serial port. The controller calculates the path. Feedback data may be retrieved either by Ethernet or by serial port connection.
6	Robot operation with two data-link channels. One sending channel over serial port and one receiving channel over Ethernet. The robot has to be programmed so that it is possible to send movement-type and data.

Table 5: Use cases.

The most important requirements are the reception of the robot sensor information during robot movement and the real-time controllability of the robot, mentioned in use case 1. The extended Data Link Control mode explained in Subsection 7.1.3 was developed to provide the required functionality defined in use case 1.

7.1.3 The Extended Data Link Control Mode

The Data Link Mode was extended through a control component, which gives the opportunity to control the robot and simultaneously receive status information. The personal computer and the robot controller were arranged in a cascaded control system, where the robot controller calculates the trajectory given by the personal computer in the form of piecewise ‘MoveTo‘ commands. This allowed to control the robot manipulator along the trajectory without stopping. Commands are sent over the Ethernet or the serial port.

Multitasking was employed to run the Data Link Control Mode programs in parallel, placed in program slots of the CR1 controller. Communication between the programs

running in parallel was realized using program external variables and user defined external variables.

The main control program MULTITASK in Listing 2 is executed first in slot 1. It sets the variables M_01 and M_02 to zero and starts the programs DATALINK and CONTROLLINK in slot 2 and slot 3. The program waits for the variables *M_01* and *M_02* to be set from the other programs to stop execution in lines 80 and 90.

```
10 RELM
20 M_01=0
25 XLOAD 2,"DATALINK"
30 XRUN 2,"DATALINK"
40 WAIT M_RUN(2)=1
50 M_02=0
55 XLOAD 3,"CONTROLLINK"
60 XRUN 3,"CONTROLLINK"
70 WAIT M_RUN(3)=1
80 WAIT M_01=1
90 WAIT M_02=1
100 XSTP 2
110 WAIT M_WAI(2)=1
120 XSTP 3
130 WAIT M_WAI(3)=1
140 GETM 1
180 HLT
190 END
```

Listing 2: Multitask management program.

The DATALINK program in slot 3 (Listing 3) sends the timestamp, current joint position, current speed of the tool centre point and current Cartesian position. Sending is looped over lines 100 to 130, and is executed until a zero value is received. After closing the communication port, the program notifies the MULITASK program by setting the external variable M_02.

```
10 WAIT M_02=0
20 M_TIMER(1)=0
30 OPEN "COM2:" AS #2
35 INPUT #2,DATA
40 IF DATA = "0" THEN 160
100 PRINT#2, M_TIMER(1), "|", P_CURR, "|", J_FBC, "|", J_CURR, "|",M_RSPD(3)
130 GOTO 100
160 M_02=1
170 WAIT M_02=0
180 END
```

Listing 3: Datalink communication.

The CONTROLLINK program moves the robot manipulator by receiving and executing movement commands. This program runs in cycle mode, and no user interaction, such as moving the robot with the teach pendant, or by robot commands in controller communication mode, is possible. Communication control is performed over the RS232 port, which results in a slow connection. However, it was still fast enough to directly send and execute robot control commands. The movement control program is shown in Listing 4. The CNT command enables the robot to move to multiple movement positions continuously without stopping at each movement position.

```
10 WAIT M_01=0
20 OVRD 100
30 GETM 1
40 CNT 1, 300
50 SERVO ON
60 OPEN "COM1:" AS #1
70 DEF JNT JNTPOS
80 INPUT #1, JNTPOS
90 MOV JNTPOS
100 GOTO 80
```

Listing 4: Control link communication.

7.2 Mitsubishi RV-2AJ Kinematics

According to Kucuk and Bingul (2006), kinematics is described as the motion of bodies without consideration of the forces or moments that cause their motion. Robot kinematics refers to the analytical study of the motion of a robot manipulator. The formulation of the kinematics model for the employed robot is crucial for robot position calculation. The Cartesian space is often employed in kinematics modelling of manipulators, and the transformation between two coordinate systems may be decomposed into a rotation and a translation. Homogenous transformations based on 4x4 orthonormal matrices are most

frequently applied in robotics. Denavit and Hartenberg (1955) showed that a general transformation between two joints requires four parameters. These parameters, known as the Denavit-Hartenberg (DH) parameters, have become the standard for describing robot kinematics. Kinematics is classified as forward and inverse kinematics. The forward kinematics problem is straightforward, and it is not complex to derive the equations. Hence, a manipulator always has a forward kinematics solution. The calculation of the inverse kinematics is computationally difficult, and generally takes a long time when compared to real-time control contexts. Singularities, nonlinearities and multiple solutions render the calculation more difficult. Thus, only a small class of manipulators with a simple kinematics have complete analytical solutions (Kucuk and Bingul, 2004). The relationship between forward and inverse kinematics is illustrated in Figure 44.

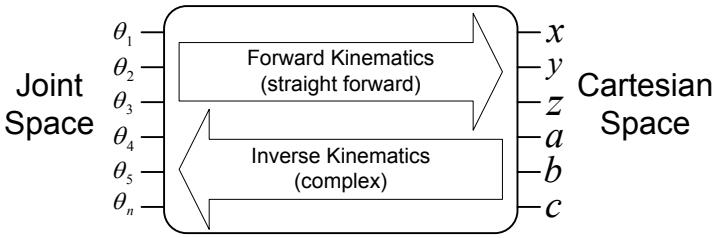


Figure 44: Schematic representation of forward and inverse kinematics

The two main solution techniques for inverse kinematics calculations are analytical and numerical methods. In the first type, the joint variables are solved analytically according to given configuration data. In the second type, the joint variables are obtained on the basis of numerical techniques.

Craig (2003) states that due to mechanical design considerations, manipulators are generally constructed with joints which exhibit just one degree of freedom. Most manipulators, like the employed Mitsubishi RV-2AJ, have revolute joints or have sliding prismatic joints.

Figure 45: Revolute (left) and prismatic (right) joints

The analytical solution of the employed manipulator in terms of geometric and algebraic solutions was applied throughout this study. The geometric approach was applied

to simple robot structures such as the arm segments 1-3 of the employed robot in Figure 46 and Figure 47. The arm segments 4 – 6 of the most industrial articulated robots require algebraic solutions. The joint axes cross at a single point, and geometric solutions are therefore difficult.

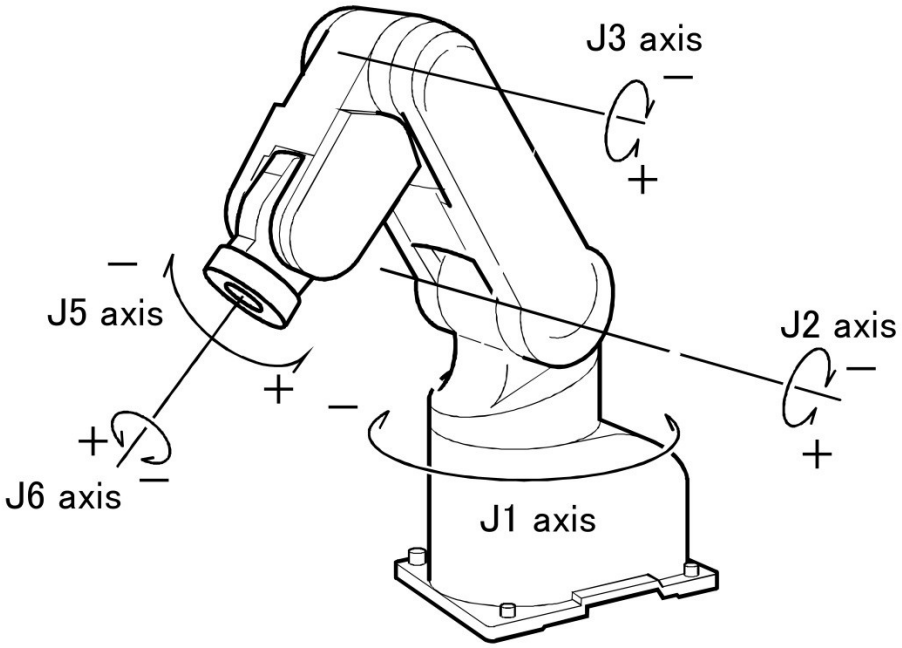


Figure 46: Mitsubishi RV-2AJ joints (from Mitsubishi documentation).

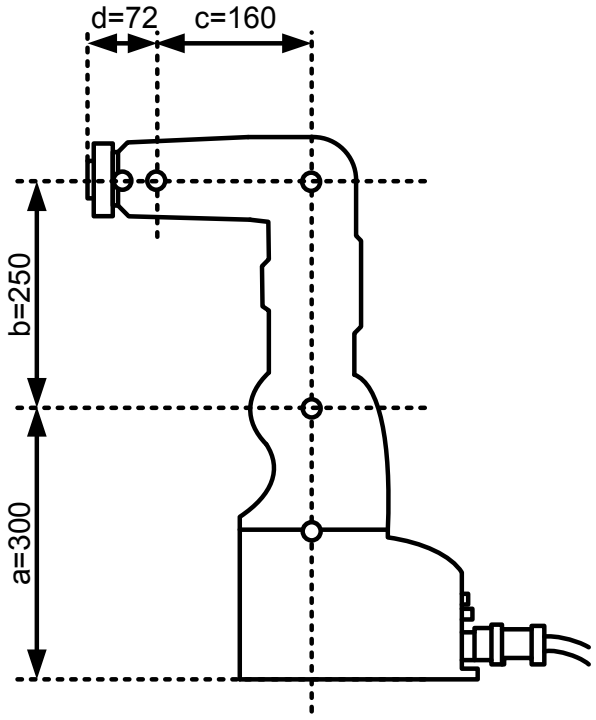


Figure 47: Mitsubishi RV-2AJ dimensions

The DH-parameters and the corresponding coordinate systems are shown in Table 6 and Figure 48, respectively. The robot flange is the mechanical interface used to mount tools. The tool centre point defines the application point of the tool. For example, a mechanical hand may have its tool centre point in between its grippers. In the absence of tools, the tool centre point is usually located in the middle of the flange surface. All calculations in this subsection are executed without tools. The rules to derive the DH-parameters from the robot geometry and variable explanations are stated in Appendix C.

Robot Arm Link Number	d [mm]	Θ [rad]	a [mm]	α [rad]
1	300	π	0	π/2
2	0	π/2	250	0
3	0	0	160	0
4	0	π/2	0	π/2
5	72	π/2	0	0
Tool t	0	0	0	0

Table 6: DH-parameters (see also Appendix C).

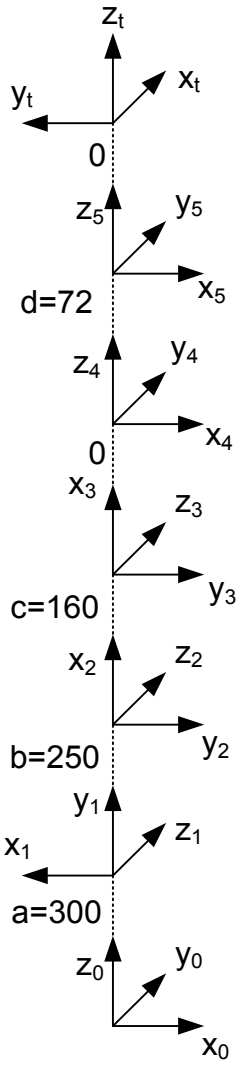


Figure 48: Robot coordinate systems.

(Home position: $\theta_1 = \theta_2 = \theta_3 = \theta_4 = \theta_5 = 0^\circ$)

7.2.1 The Geometric Solution

The geometric solution was applied for the manipulator arm joints 1-3. The trigonometric functions $atan2$ and the cosine law were employed to solve the geometric calculations analytically, as illustrated in Figure 49 and equations (13) and (14). $atan2$ is generally applied using the sin/cos function of an angle to increase the accuracy of angle calculations, instead of calculating angles with $asin$ or $acos$ directly.

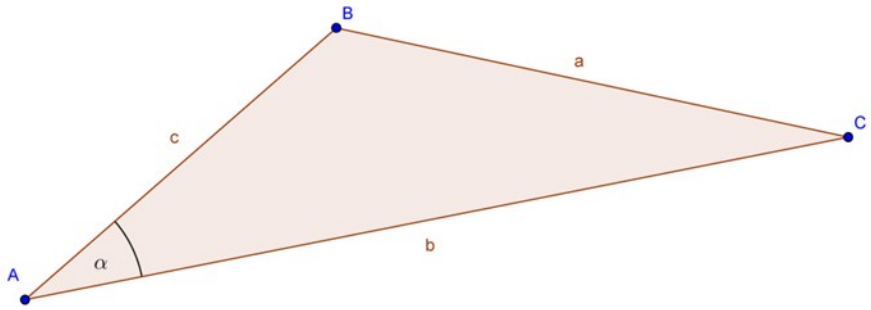


Figure 49: Law of cosine

(13) $$a^2 = c^2 + b^2 - 2cb \cdot \cos(\alpha)$$

(14) $$\cos(\alpha) = \frac{1}{-2cb} \cdot \left(a^2 - (c^2 + b^2)\right) = \frac{1}{2cb} \cdot (-a^2 + c^2 + b^2)$$

Calculation of θ_1

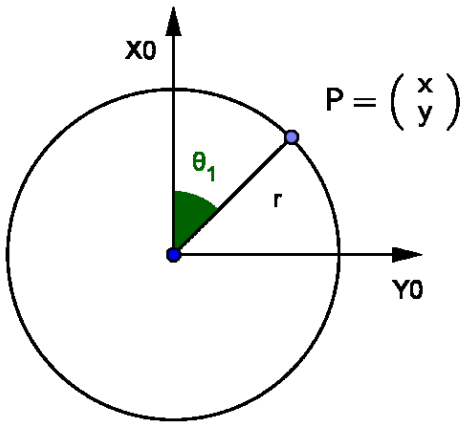


Figure 50: Geometric inverse calculation for joint 1

From the robot dimensions in Figure 47, the values $a = 300, b = 250, c = 160$ and $d = 72$ are given. As illustrated in Figure 50, the coordinate of the tool centre point on the

x/y layer is $P = (x \quad y)^T$. θ_1 is calculated in equation (18) using the $atan2$ function and the sin/cos of θ_1 to improve the accuracy.

(15) $$r = \sqrt{x^2 + y^2}$$

(16) $$\sin(\theta_1) = \frac{y}{r}$$

(17) $$\cos(\theta_1) = \frac{x}{r}$$

(18) $$\theta_1 = \text{atan2}\,(\sin(\theta_1), \cos(\theta_1)) = \text{atan2}\left(\frac{y}{\sqrt{x^2 + y^2}}, \frac{x}{\sqrt{x^2 + y^2}}\right)$$

Calculation of θ_2

The angle θ_2 of joint 2 is calculated by considering θ_3, respectively arm segments b and c.

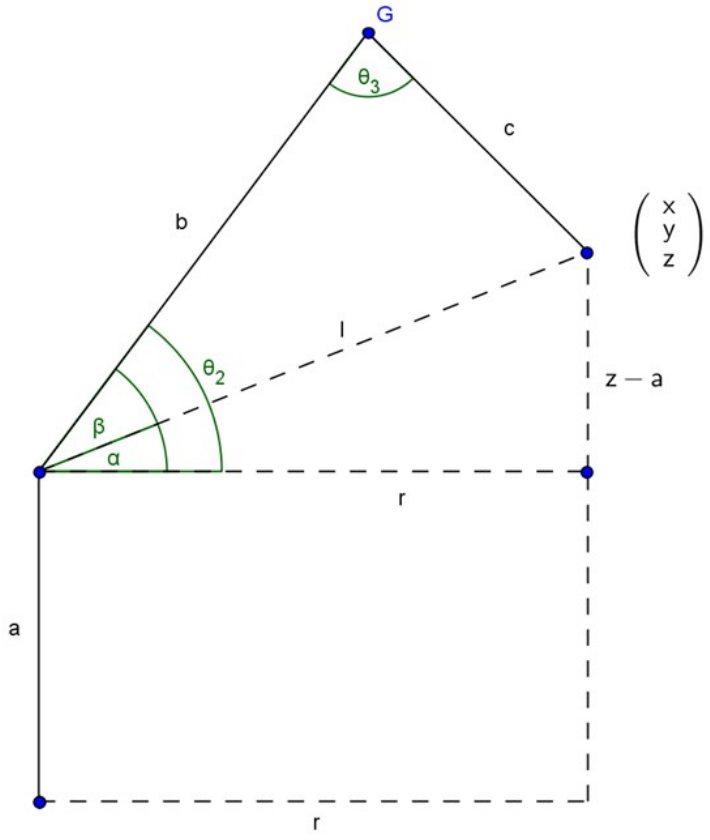


Figure 51: Geometric inverse calculation for joint 2 and 3

Additional factors c_1 and c_2 with $c_{1/2} = \{-1, 1\}$ are introduced to incorporate the unconsidered sign of the square root. The combined factor k was applied to the addition theorem for angles in equations (25) and (26).

(19) $$l = \sqrt{r^2 + (z-a)^2}$$

(20) $$\cos(\alpha) = c_1 \cdot \frac{r}{l}$$

(21) $$\sin(\alpha) = (z-a) \cdot \frac{1}{l}$$

(22) $$\cos(\beta) = (b^2 + l^2 - c^2) \cdot \frac{1}{2 \cdot b \cdot l}$$

(23) $$\sin(\beta) = \sqrt{1 - cos^2(\beta)}$$

(24) $$k = c_1 \cdot c_2$$

(25) $$\sin(\theta_2)' = \sin(\alpha) \cdot \cos(\beta) + k \cdot \cos(\alpha) \cdot \sin(\beta)$$

(26) $$\cos(\theta_2)' = \cos(\alpha) \cdot \cos(\beta) - k \cdot \sin(\alpha) \cdot \sin(\beta)$$

(27) $$\theta_2{}' = atan2(\sin(\theta_2)', \cos(\theta_2)')$$

The joint angle θ_2 must also consider the home position of the robot, as illustrated in Figure 48. Thus, the angle θ_2' must be subtracted from the angle value of 90° to comply with the defined home position of the robot.

(28) $$\theta_2 = 90° - \theta_2' = 90° - \text{atan2}(\sin(\theta_2)', \cos(\theta_2)')$$

Calculation of θ_3

θ_3 is calculated by applying the $atan2$ function. The angle $\theta_3{}'$ has to be subtracted from the angle value of 180° to comply with the defined home position of the robot.

(29) $$\cos(\theta_3{}') = (b^2 + c^2 - l^2) \cdot \frac{1}{2bc}$$

(30) $$\sin(\theta_3{}') = k \cdot \sqrt{1 - \cos^2(\theta_3{}')}$$

(31) $$\theta_3{}' = \text{atan2}(\sin(\theta_3{}'), \cos(\theta_3{}'))$$

(32) $$\theta_3 = 180° - \theta_3{}'$$

7.2.2 Algebraic Solution

The algebraic solution is based on the common transformation equation (33), which considers the rotation and translation between two joints. The DH parameters identified in Table 6 are applied to the common transformation equation (33). The common transformation ${}^{0}A_{\mathrm{t}}$ for all joints is stated in (34).

$$(33) \qquad {}^{i-1}A_i = \begin{pmatrix} \cos(\theta_i) & -\cos(\alpha_i)\cdot\sin(\theta_i) & \sin(\alpha_i)\cdot\sin(\theta_i) & a_i\cdot\cos(\theta_i) \\ \sin(\theta_i) & \cos(\alpha_i)\cdot\cos(\theta_i) & -\sin(\alpha_i)\cdot\cos(\theta_i) & a_i\cdot\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$(34) \qquad {}^{0}A_{\mathrm{t}} = {}^{0}A_1 \cdot {}^{1}A_2 \cdot {}^{2}A_3 \cdot {}^{3}A_4 \cdot {}^{4}A_5 \cdot {}^{5}A_{\mathrm{t}} = {}^{0}A_3 \cdot {}^{3}A_5 \cdot {}^{5}A_{\mathrm{t}}$$

The tool coordinate system equals the coordinate system of the robot flange since no tool is attached. It is given by equation (35).

$$(35) \qquad {}^{5}A_{\mathrm{t}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The angles θ_1 to θ_3 are already known from the geometric calculations above, and the tool transformation ${}^{5}A_t$ is also known. Therefore, the transformation ${}^{3}A_5$ may be calculated using equation (36) to achieve the angles θ_4 and θ_5. M is the computed target matrix, which is also known. Generally, ${}^{3}A_5$ is given by equation (37).

$$(36) \qquad {}^{3}A_5 = {}^{0}A_3^{-1} \cdot {}^{0}A_{\mathrm{t}} \cdot {}^{5}A_{\mathrm{t}}^{-1} = M$$

$$(37) \qquad {}^{3}A_5 = \begin{pmatrix} \cos(\theta_4) & 0 & \sin(\theta_4) & 0 \\ \sin(\theta_4) & 0 & -\cos(\theta_4) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta_5) & -\sin(\theta_5) & 0 & 0 \\ \sin(\theta_5) & \cos(\theta_5) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \cos(\theta_4)\cdot\cos(\theta_5) & -\cos(\theta_4)\cdot\sin(\theta_5) & \sin(\theta_4) & 0 \\ \sin(\theta_4)\cdot\cos(\theta_5) & -\sin(\theta_4)\cdot\sin(\theta_5) & -\cos(\theta_4) & 0 \\ \sin(\theta_5) & \cos(\theta_5) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = M =$$

$$= \begin{pmatrix} \mathrm{m}_{11} & \mathrm{m}_{12} & \mathrm{m}_{13} & \mathrm{m}_{14} \\ \mathrm{m}_{21} & \mathrm{m}_{22} & \mathrm{m}_{23} & \mathrm{m}_{24} \\ \mathrm{m}_{31} & \mathrm{m}_{32} & \mathrm{m}_{33} & \mathrm{m}_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The joints θ_4 and θ_5 were found through comparison in equations (38) and (39).

$$(38) \qquad \theta_4 = \mathrm{atan2}(m_{13}, -m_{23})$$

(39) $\theta_5 = atan2(m_{31}, m_{32})$

The angles θ_4 and θ_5 are independent of the manipulator position, and they are only dependent on the orientation of the tool centre point. Thus, those angles have to be set correctly in order to reach a specified target location.

To calculate the reachability of a location, its orientation must be known. Otherwise, the solution space may be large and an appropriate manipulator configuration must be chosen. This is application dependent, and will be further discussed in Subsection 8.5.3.

7.2.3 Application of the Dubins Airplane Model

In the geometric formulation of the movement problem, the robot has been reduced to a point on a two dimensional surface with similar behaviour to Dubins car (Dubins, 1957). This car is able to drive only forward and the radius of steering is bounded. An extension of the Dubins car is given with the Dubins airplane, which applies to $\mathbb{R}^3$ spaces (Chitsaz and LaValle, 2007). The robot position is uniquely defined by the position and orientation. The quadruple $(x, y, z, \theta) \in \mathbb{R}^3$ and $\theta \in [0,2\pi[$ represents the configuration and the coordinates (x, y, z) represent the midpoint, while θ represents the orientation of the airplane, as shown in Figure 52. θ is the angle between the x-axis of the frame and the airplane's local longitudinal axis in the x/y plane. Thus, the Dubins airplane is the Dubins car with an additional configuration variable for altitude z. This is a simplified model of a real airplane.

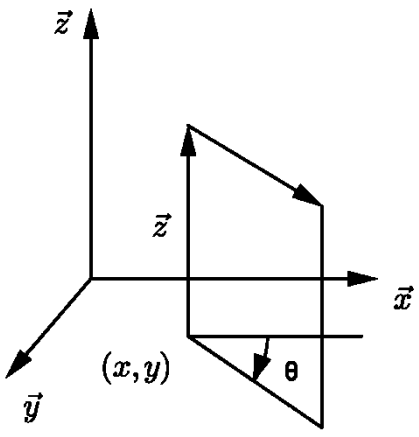


Figure 52: Dubins airplane model.

An industrial manipulator can 'fly' curves in any direction, thus, a second parameter λ was added for the orientation. The 5th-tupel $(x, y, z, \theta, \lambda) \in \mathbb{R}^3$ with $\theta \in [0,2\pi[$ and $\lambda \in [0,2\pi[$ represents the configuration, while θ and λ represent the orientation, as shown

in Figure 53. λ is the angle between the x-axis of the frame and the airplane's local longitudinal axis in the x/z plane. The orientation is equal to spherical coordinates (Papula, 1998).

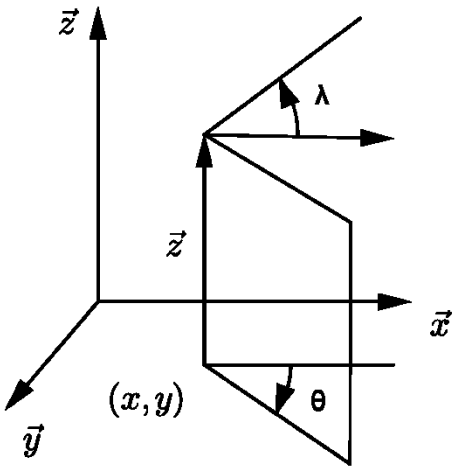


Figure 53: Industrial manipulator 'free flying' model.

Industrial articulated robots do not have good movement capabilities when compared to the industrial manipulator model. They most often provide joint, linear and circular movement primitives. The circular movement is restricted to have a static radius during circular movement. The restrictions on the steering angle are higher than on the classic non-holonomic movement constraint.

Nevertheless, the industrial manipulator model, coupled with the restriction on the static curvature radius were applied. Equally, the autonomous mobile robot has to meet the constraints of the Dubins car coupled with the restriction to the static curvature radius to allow direct comparison with the manipulator movements.

7.3 Robotino Mobile Robot Control

The autonomous mobile robot Robotino allows research on trajectory planning in a two-dimensional world space without the restrictions of the robot arm. The developed path planning algorithms were first tested on this robot before they have been applied to the industrial robot arm.

The provided robot control framework supports wireless local area network connections to command the robot and to obtain sensor information. Commands, for example driving commands, are generally sequentially executed until the end of the robot movement. Driving commands allow the speed of each wheel of the Omni drive to be controlled. The

Omni drive controller also supports interpolated movement types such as linear and circular movements by setting the linear speed in the plane, the $\vec{t}$ and $\vec{n}$ direction, and a rotational speed about the plane normal $\vec{e}$, as illustrated in Figure 56.

Figure 54: A Robotino robot from the company Festo.

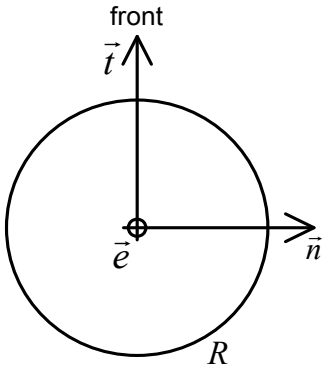


Figure 55: Local coordinate axes of the Robotino robot.

7.4 Robotino Kinematics

The kinematics of a car-like robot is also valid for the employed mobile Robotino robot, although a car-like robot has two rear wheels and two directional front wheels. The movement controller of the Robotino imitates this behaviour through circular interpolation. The robot moves on the plane $\mathbb{R}^2$ and its configuration is uniquely defined by the position and orientation. The triple $(x, y, \theta) \in \mathbb{R}^2$ and $\theta \in [0,2\pi[$ represents the configuration, where (x, y) are the coordinates of the midpoint and θ is the orientation of the robot, as depicted in Figure 56.

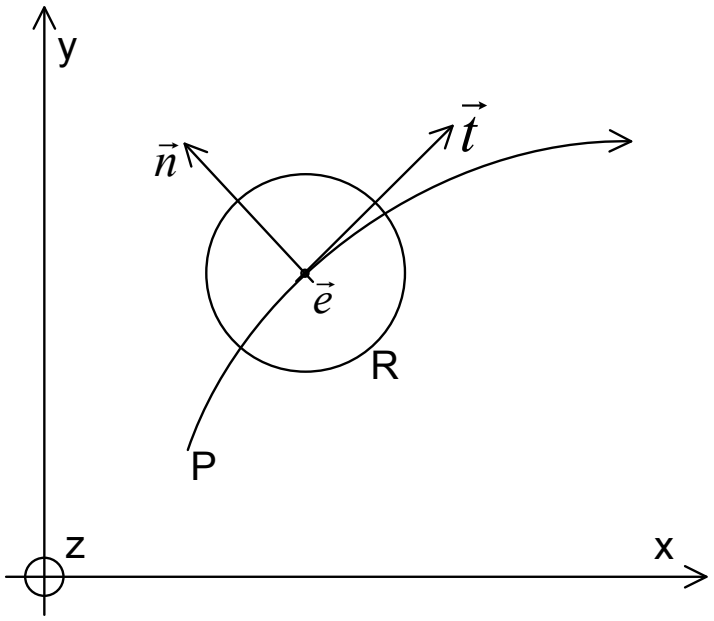


Figure 56: Kinematics of a car like robot.

As illustrated in Figure 57, the simulator is a point R in three-dimensional space, where the local robot coordinate system $\vec{t}$, $\vec{n}$ and $\vec{e}$ is given with the origin R, which is the centre of the robot. Linear movements can only be executed along the $\vec{t}$ axis of the local robot coordinate system, although an Omni drive may also be able to move in the $\vec{n}$ axis direction. Because the trajectory of the industrial manipulator was compared to a free-flying car-like robot, sideway movements were forbidden and set as a constraint.

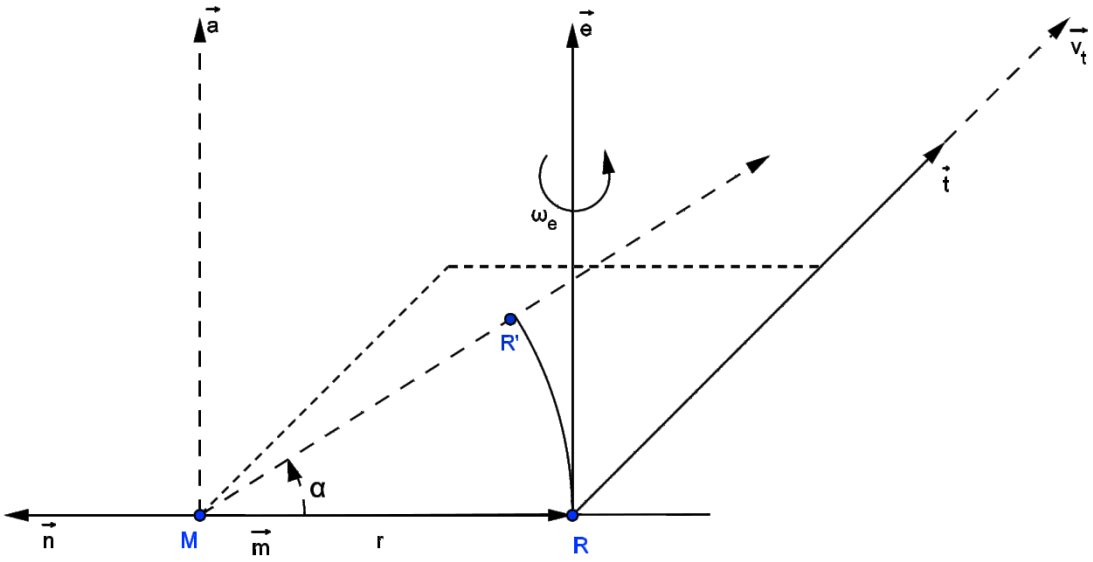


Figure 57: Robotino calculations.

Circular movements may be executed using an angular speed ω_e that results in a circular speed v_a. The robot turns around the given local $\vec{e}$ axis and moves forward along the $\vec{t}$ axis at the same time. It drives linearly forward when $v_t > 0$ and $\omega_e = 0$. Equations (41) and (42) are obtained with the given parameters Δt, ω_e, R, v_t, $\vec{a}$ and M, which are further described in (40).

(40) $$\omega_e = 0{,}2 \cdot \frac{1}{s}$$

$$v_t = 1 \cdot \frac{m}{s}$$

$$\vec{a} = (0 \quad 0 \quad 1)^T$$

$$M = (0 \quad 0 \quad 0)^T$$

(41) $$\alpha = \omega_e \cdot \Delta t$$

(42) $$r = \frac{v_t}{\omega_e}$$

The orientation calculation of R to the new orientation R' was carried out by computing equation (43). The parameter R is the actual orientation and position, Rot is the Rotation α around the given axis $\vec{a}$, $Trans$ is the translation of R, so that $Pos(R) = (0 \quad 0 \quad 0)^T$ and $Trans^{-1}$ is the back translation.

(43) $$R' = Trans^{-1} \cdot Rot \cdot Trans \cdot R$$

Additional constraints are given in equations (44) and (45). The tangent direction is continuous and the turning radius respects a minimum constraint. These paths may be followed by a real vehicle without stopping, and therefore have a continuous curvature profile in their motion.

(44) $$\tan \alpha = \frac{\dot{y}}{\dot{x}}$$

(45) $$r \geq r_{min}$$

These relations are non-holonomic (Barraquand and Latombe, 1989) and restrict the shape of the paths of the mobile robot. Autonomous mobile robots with these constraints applied are called Dubins car in $\mathbb{R}^2$ (Dubins, 1957).

7.5 Robot Simulation

Simulation of the employed robot types was introduced to speed up algorithm test and development. The simulation of the robot arm is restricted to forward and inverse kinematic calculations. The mobile robot simulation utilizes an extended kinematics of the Robotino robot, and allows linear and circular movements in $\mathbb{R}^3$. Now, the local $\vec{n}$ and $\vec{t}$ axes are used to rotate the robot, which leads to the industrial manipulator model. In addition, the simulator supports linear movements.

7.6 Summary

Robot control applications require a connection to the real robot system. Sending robot control commands as well as receiving information from the robot, such as the position, speed and orientation, is necessary, especially for path-planning applications that focus on algorithm development. This framework enables the utilization of a standard industry robot system, an autonomous mobile robot, and a simulated robot. The kinematics computation for each supported robot, including the simulated robot, was implemented.

The framework extends the Mitsubishi CR1 controller family robot system and employs a new communication mode. It receives robot information during movement, and sends robot commands during movement of the robot manipulator without stopping between the commands.

8 Investigation into a Trajectory Planning Algorithm to Support Intuitive Use of the Robot Programming System

This chapter corresponds to objective four, which is the research and development of an enhanced online robot programming support system that generates static robot programs for industrial robot manipulators. The most important findings have been published in one journal paper and one conference paper.

From the requirements described in Chapter 5, a method was researched to combine the maintainability of the robot program and the shortness of the robot trajectory. In terms of the clarity and changeability of the generated robot program, the maintainability is important in industry, and enables the flexibility to modify existing robot programs manually. The system provides the connection to external devices such as the robot, the vision system, the joystick and the pointing device, and also integrates the required software components.

Section 8.1 explains the usage scenarios of the robot programming system which has to be supported by the developed system. An overview of the main components of the system is described in Section 8.2. The probabilistic world model and the robot kinematics and control framework have already been introduced in Chapters 6 and 7. The interaction with the operator required assistance leading through the necessary steps to generate the robot program. This assistance is based on a suitable HMI, which is described in Section 8.3, to enable inexperienced operators to work with the system. The mission defines the overall aim of the robot task, which can include gluing, handling or pick-and-place tasks. The mission planner presented in Section 8.4 controls the trajectory planner, enabling it to fulfil the given mission. Existing trajectory planning algorithms often execute path smoothing after path finding, although these tasks are competitive. The proposed trajectory planner in Section 8.5 allows the simultaneous execution of both tasks. In subsequent steps, these trajectories have to be generated to a robot program file, which can be directly employed to the industrial production system. Section 8.6 describes a geometric approach to accomplishing this transformation step. The most important findings are summarized in Section 8.7.

8.1 Usage Scenarios

The usage scenarios are described as use cases (Balzert, 1999). A use case is a list of steps that defines interactions between the operator and the system to achieve the goal of generating a robot program. All of the use cases were realized with the aim of an intuitive

execution to reduce the need for special knowledge. The diagram in Figure 58 shows a graphical representation of the considered use cases.

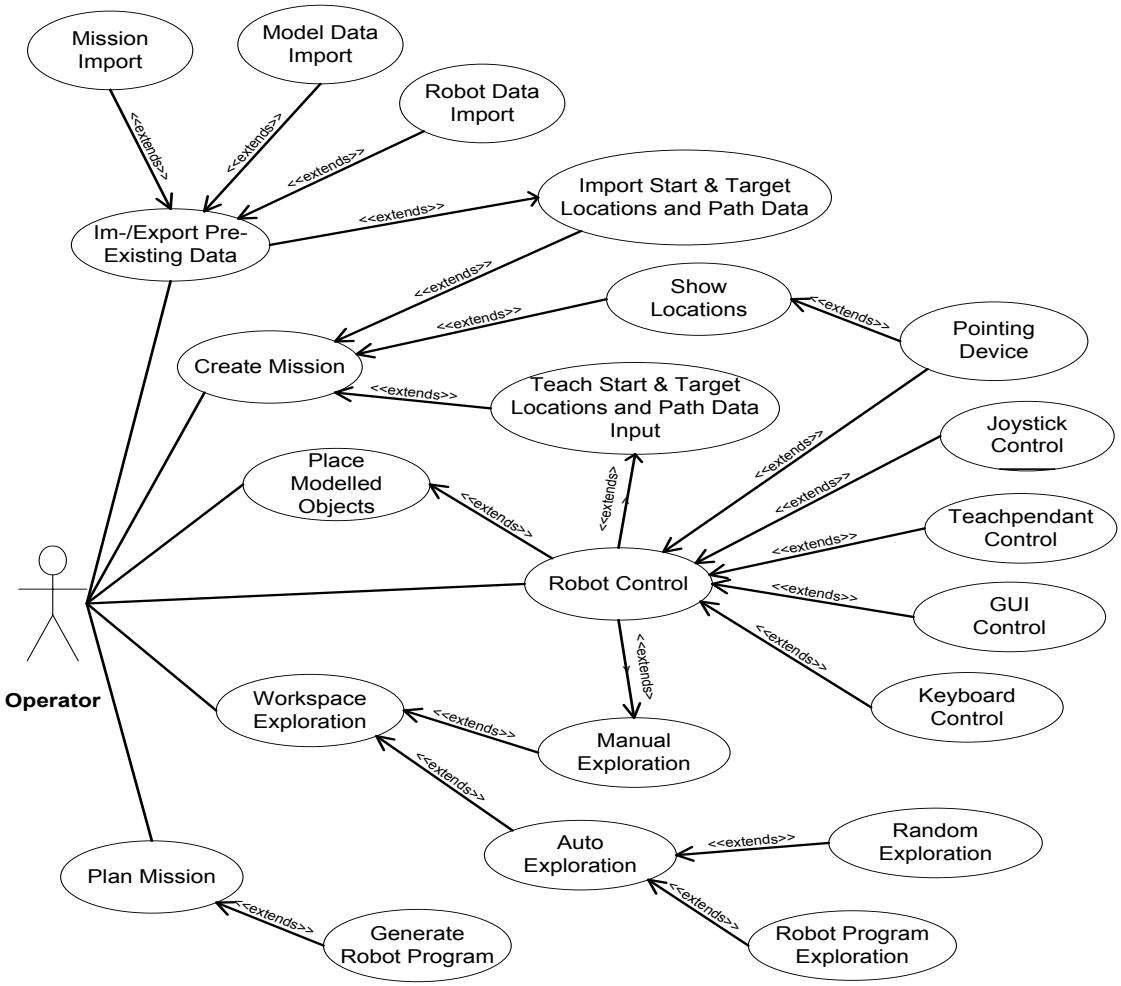


Figure 58: The use cases of the support system.

Use Case 1: Import/Export Pre-Existing Data

During data import and export, data of models, the mission and the robot, including the robot specific kinematics, are loaded from or saved to a disk. The mission parameters contain mission application paths and locations, as well as other planning parameters.

Use Case 2: Create Mission

Missions including start and target locations can be created manually or by importing the model data. Information for the application type is entered within the GUI or is also included within the model data.

Use Case 3: Place Modelled Objects

Modelled objects are required to be placed within the workspace so that the world model can be updated. This positioning process may be done with the robot or directly with the pointing device. The applied four-point method uses four locations to define the position and orientation of the modelled object.

Use Case 4: Robot Control

The robot is controllable with input devices such as a joystick, pointing device, teach pendant, keyboard, and a GUI.

Use Case 5: Workspace Exploration

Exploration of the workspace may help to reduce the time taken to generate the trajectory. This can be done either manually or automatically. Manual exploration utilizes manual robot control, while automatic exploration is done by random movements or by the execution of pre-existing robot programs.

Use Case 6: Plan Mission

Planning the mission includes mission and path planning as well as robot program generation to a textual robot program file. The output represents the result of the enhanced online robot programming system in the form of a file, and it may be exported directly to the robot or as a text file to the hard disk.

The use cases have to be subsequently executed in order to generate the robot program. Thus, the workflow in Listing 5 has been defined to summarize the use cases.

1. Set up an online path planning and the enhanced online robot programming system including hardware.
2. Importation of pre-existing data such as robot geometry and CAD data.
3. Create a mission using import, robot movements, CAD locations, pointing devices or simulations.
4. Execution of the support system.
5. Robot program generation.
6. Uploading of the robot program file to the robot.
7. Removal of the support system.

Listing 5: Summary of the use cases.

8.2 System Overview

The overview of the enhanced online robot programming support system illustrated in Figure 59 shows the involved software components, devices and sensors. The software components were developed to support the defined operator use cases stated in Section 8.1.

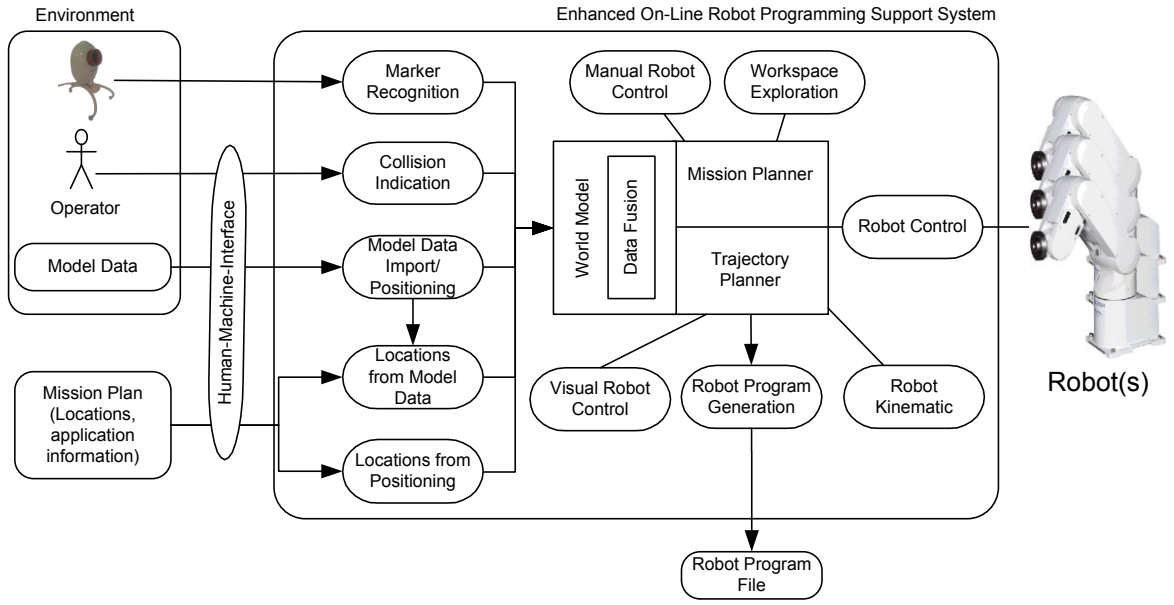


Figure 59: Support system overview.

In general, the system provides an HMI that consists of a GUI, a joystick and a pointing device. The main task of the system is to generate a robot program file from a given mission. The definition of the mission still relies on the operator, who provides knowledge of the application such as painting, gluing or pick-and-place tasks. A mission consists of application locations and paths that include application information, such as the colour for painting. Both can be provided within the model data or may be amended within the support system. The application locations can be manually determined.

The robot control component controls the manipulator, the mobile robot and the simulated robot. The robot kinematic component provides forward and inverse calculations. Both components are described in Chapter 7. The robot system is equipped with a teach pendant to control the robot movement manually. Additional input devices, e.g. joystick, GUI of the robot programming system, visual servo-control, mouse and keyboard, have been connected to simplify manual robot control.

The visual servo-control applies a pointing device to indicate the target location to the robot using the marker recognition component. The robot moves automatically towards the given location with the help of a neural network, and stores the position. Subsequently, the network transforms picture coordinates into robot control commands, as described in Subsection 8.3.2.

The importation of model-based CAD data was employed to represent the world model more accurately. CAD data from simulation systems, such as RobCAD (Tecnomatix, 2011), can be exported as DXF files including all locations attached. Usually, CAD data already exist in simulation tools and modelling software. They are taken from laser-scan- or construction-processes. This model data was placed within the real robot cell, hence improving the accuracy of the world model. In addition, these objects allow the use of physically unavailable objects.

Data fusion combines all information sources to deliver cohesive data to the world model. The data sources also include the robot positions from existing robot programs to explore the working space. It was also explored by random or manually controlled movements. Collisions are always processed during exploration so that free and occupied areas of the workspace are explored throughout its movements by manual collision indications. Thus, the world model becomes more accurate during the exploration process.

The mission and path planner presented in Sections 8.4 and 8.5 together handle the planning of a motion in real-time, including shortest-path calculation and collision avoidance. Finally, the entire robot motion is stored within the support system in the form of a trajectory that consists of connected particles. Its transfer to a robot-specific program file is achieved in two steps: first, the translation into a robot program of solely the provided trajectory; secondly, the generation of the specific robot program enriched by additional configuration commands and specific linguistic syntaxes. The two-step generation, described in Section 8.6, also supports other robot types and languages.

All software components were created with the developed code generation toolchain presented in Chapter 9. Each of those components is an independent component with a clear interface to the software framework. This simplified the use of third party work, such as for DLL integration. All components may be developed independently as soon as the interface and information exchange are specified. Each component provides life-cycle and

message-based communication functionalities as well as a run-time behaviour, which is called an execution model and it is detailed in Appendix E. A graphical model specifies the communication flow and message types, which were subsequently generated into source code, including additional libraries that are required to execute the software system.

8.3 Human Machine Interface

This section gives a detailed overview regarding the concept, functionality and structure of the HMI of the system. The interface was kept simple and it provides a GUI in addition to external robot control devices and a pointing device. The pointing device delivers its position information, which was processed together with the robot arm position to realize visual servo control.

8.3.1 Graphical User Interface

The GUI was created in a generic and flexible way to guide the robot-program-file generation process. A finite state machine was exploited to ensure valid system behaviour and to improve the system robustness as it relates to wrong user entries and undefined states. The GUI adapts its toolbar and workflow to the state machine, which acts as a controller. It provides a defined set of states, which help to reduce the risk of GUI errors.

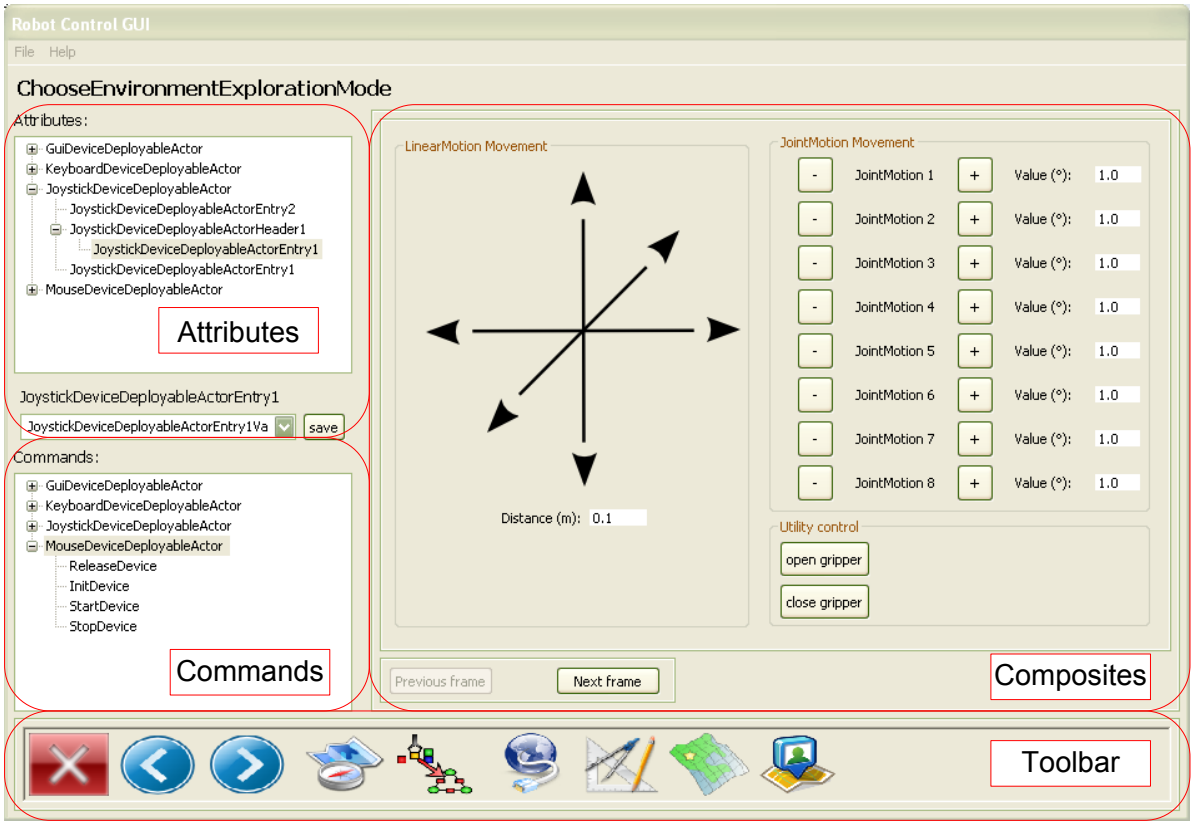


Figure 60: Design of the Graphical-User-Interface.

The GUI is structured into five main components, as shown in Figure 60. During the development phase of the system, debugging was required to implement the user interaction. The debugging-related widgets, the attributes, the commands, and the log window are not shown in the final version.

The robot program generation-related widgets are placed within the ‘Composites’ area in Figure 60. A composite is a Java class containing SWT widgets that are required by a software component to allow user input with a GUI. Each individual software component that requires user input encapsulates its own composite, which is dynamically integrated into the GUI. The fixed tab ‘Workflow’ dynamically displays all state dependent composites. For each state change, the appropriate composite is displayed. The ‘Vision’ tab shows the camera views and the ‘Scene’ tab contains a graphical representation of the world model, which is visualized by a Java 3D viewer.

Each connected software component may provide attributes which are displayed within the ‘Attributes’ area. The message-based communication of these components allows the

display of all allowed messages within the ‘Commands’ area, and can be directly executed during debugging.

The Finite-State-Machine

The Finite-State-Machine was realized using UniMod (eVelopers Corporation, 2011), which is an open-source application. It allows the developer to design an application logically with the help of state-chart diagrams and the generation of Finite-State-Machine Extensible Markup Language (XML) description files. The XML-description files are executed using a Finite-State-Machine runtime framework.

Figure 61 provides an overview of the communication structure of the GUI software component. Messages to other software components may be sent during an event of a widget, such as a button click. The Event Provider is the interface for incoming messages from other software components and from internal messages.

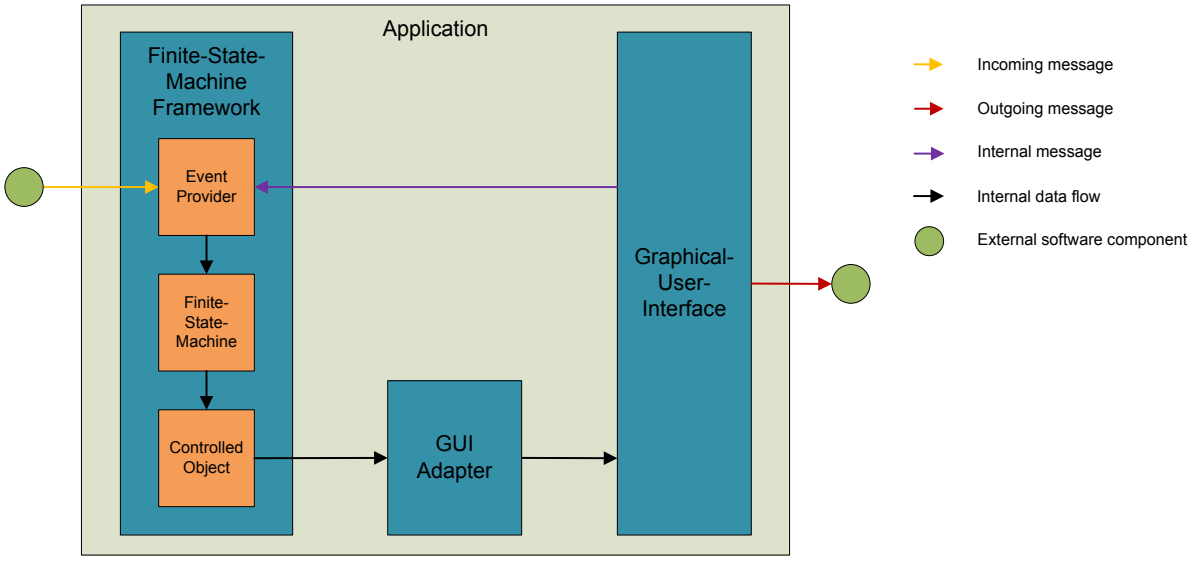


Figure 61: Communication system of the Graphical-User-Interface.

The UniMod resource of the GUI consists of three parts. The event provider transforms received messages into events that can be processed. The controlled object connects the Finite-State-Machine with the ‘GUI Adapter’ to control the GUI. The ‘GUI Adapter’ is a Java class that was required to decouple the GUI source code from the user code. The Jigloo SWT/Swing GUI builder (Cloudgarden, 2011) was utilized to create the user interface.

As shown in Figure 62, the finite state machine was employed to control the GUI. Buttons on the GUI send messages to the finite state machine and cause a state change. This leads to a GUI change in the toolbar and the composites.

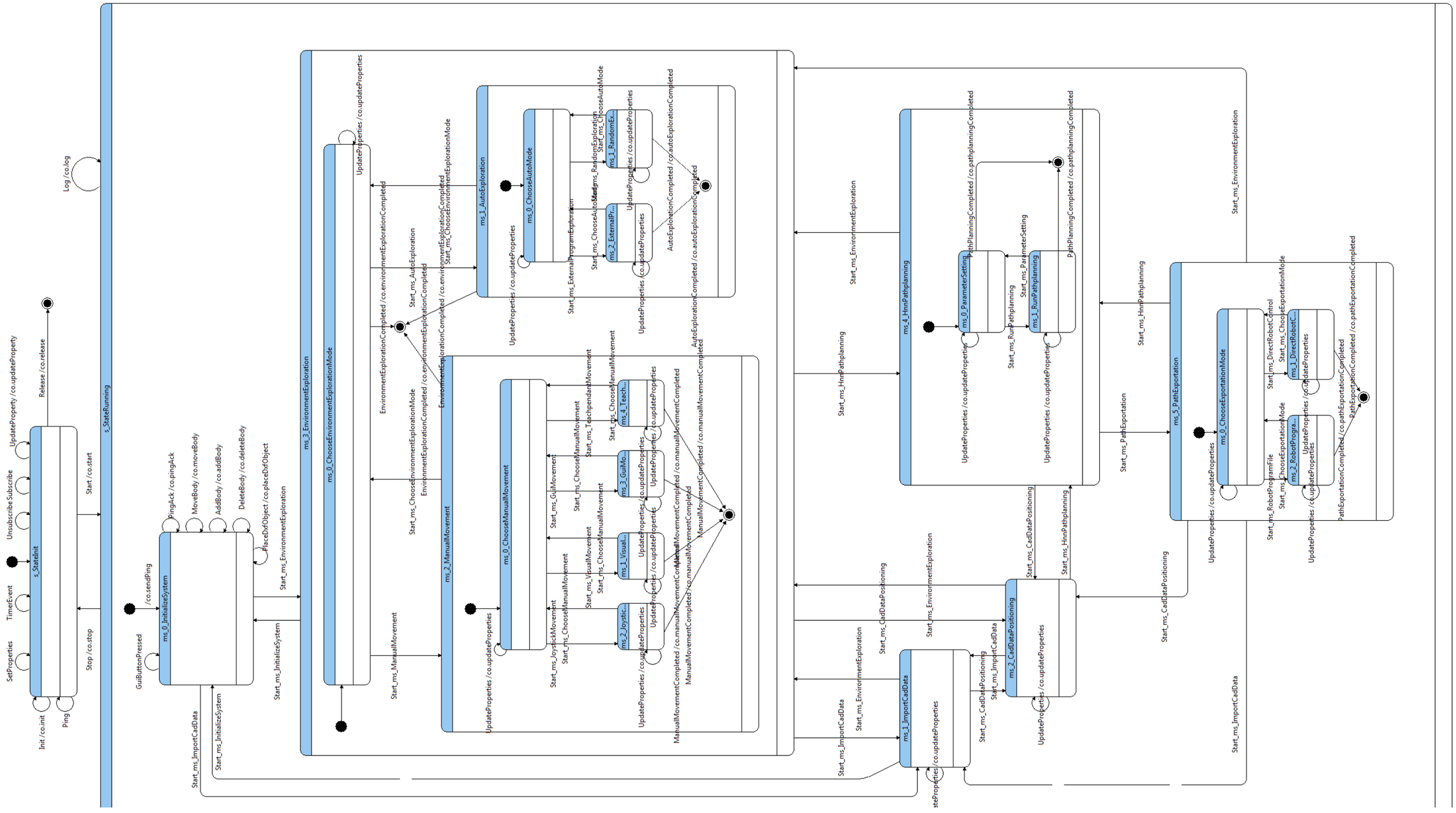


Figure 62: Graphical-User-Interface controller Finite-State-Machine.

Dynamic Toolbar

The toolbar is dependent on the underlying finite state machine and presents three fixed elements on its left: ‘Exit’, ‘Previous state’ and ‘Next state’. These elements are fixed for all states. The buttons ‘Previous state’ and ‘Next state’ lead the user through predefined usage guidelines. The ‘Exit’ button shuts down the robot programming system and closes the GUI.

State dependent buttons are dynamically added to the toolbar. They represent the possible state transitions to connected states from the current state. If one of these buttons is clicked, the trigger activates the transition to the desired state. All buttons are represented by a symbol and the associated name of the state as tool tip text.

An important aspect of the finite state machine concept is the parsing of the state machine for connected main states to display the workflow in the toolbar, as illustrated in Figure 63.

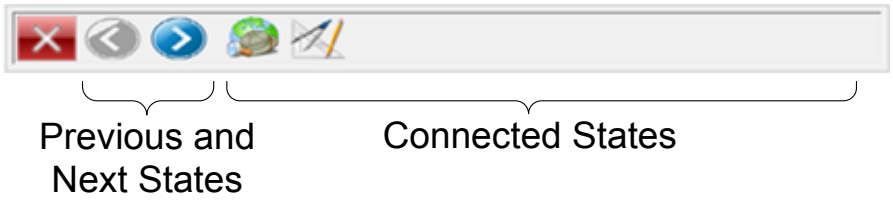


Figure 63: The dynamic toolbar.

The finite state machine is separated into main states (‘ms_’) which represent individual composites, and general states (‘s_’) which have internal functions. Every main state has a state number that is defined by the standard path through the finite state machine, which is proposed to the user as a standard workflow. Within each parent state, its child states are numbered starting from zero, as demonstrated in Figure 64. The operator may leave the proposed workflow, for example by following the dashed path. The next and previous states along the proposed workflow have to be calculated in order to lead the operator along the proposed path.

This was done for the previous state by obtaining the previous state with the highest number which is smaller than the current state number. For the calculation of the state prior to the first state within a parent state, the previous state of the parent state is calculated in the same manner.

The next state is calculated by obtaining the subsequent state number that is greater than the current state number. For the next state calculation of the last state within a parent state, the next state of the parent state is calculated.

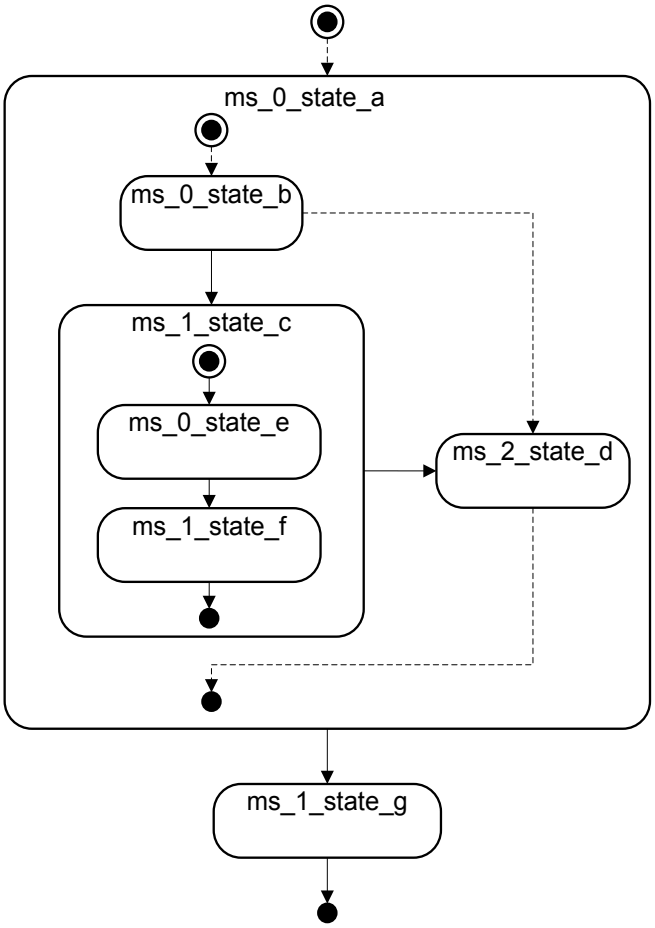


Figure 64: The dynamic toolbar.

8.3.2 Visual Servo Robot Control

The visual servo-control component employs the pointing device to indicate the target locations in space. Both the pointing device and the robot-arm are equipped with markers. To achieve information regarding the position of the objects in space, the robot cell is equipped with a vision system which monitors the space within the robot cell. The vision system recognizes the picture coordinates of the markers to enable the visual servo-control component to control the robot-arm towards the pointing device. Subsequently, a neural network transforms the picture coordinates into robot control commands, as described by Ritter (1994) with a Kohonen network and Lenz and Pipe (2003) with a radial basis

function network. The robot moves automatically towards the given location. An overview of the visual servo control application is given in Figure 65.

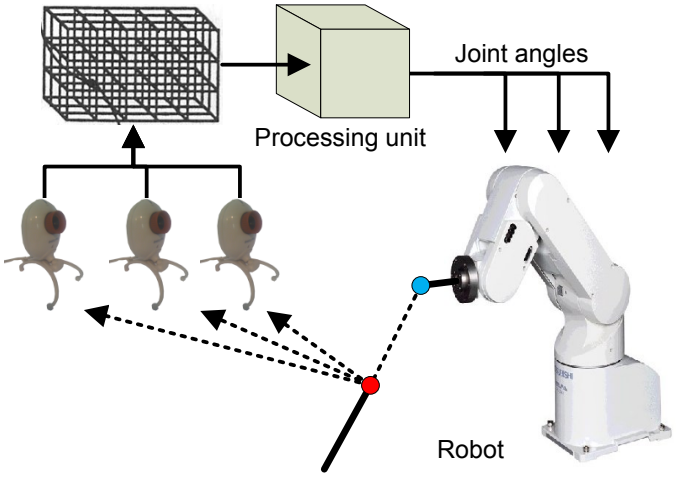


Figure 65: Visual servo control application.

The neural network is trained with randomly chosen target locations. The target location is monitored from the cameras and their signals are applied to the neural net. Each neuron is responsible for a subspace of the robot cell. An activated neuron provides control signals to the robot controller. Each camera of the vision system delivers a two-dimensional image coordinate of the viewing pane, and the neural network learns the transformation to control signals for the five robot joints. The robot moves to incorrect robot positions at the beginning of the learning process, but the accuracy is improved with each learning position and the difference between the robot positions and the target positions. There was no need for more information about the robot, the cell, the cameras or its positions in space. This is the typical behaviour of an autonomous learnable system.

8.4 Mission Planner

Robot programs in industrial settings often include several application tasks for the robot, which are summarized in a mission. The order of the task execution can be limited by tasks of other robots or humans, and results in interaction between the participants. In this study, the planning of the mission tasks resulted in the well-known ‘travelling salesman problem’ (Russell and Norvig, 2002). The movement distance of the robot was utilized to define a cost and optimization function for path planning. The distance information of the mission task is extended during path planning using sensor data. Thus,

the mission planner and the path planer are interconnected to exchange mission planning-related data.

8.4.1 The General Path Planning Control Loop

The mission and path planning control loop depicted in Figure 66 shows the interaction of the mission and path planner with the robot system and its environment. The mission and path planner component computes trajectories for a given mission. The mission and path planner subsequently calculates a path and controls the robot manipulator along that path. The robot interacts physically with the environment, and its movement is monitored by internal and external sensors which provide its position and velocity to the path planner. Local obstacles and workpieces may also exist within the workspace and have to be recognized. Collisions were detected by sensors or by the in-memory world model. Their positions and velocities are provided to the mission and path planner and to the world model, which creates an in-memory map of the environment. All of these components are executed in real-time.

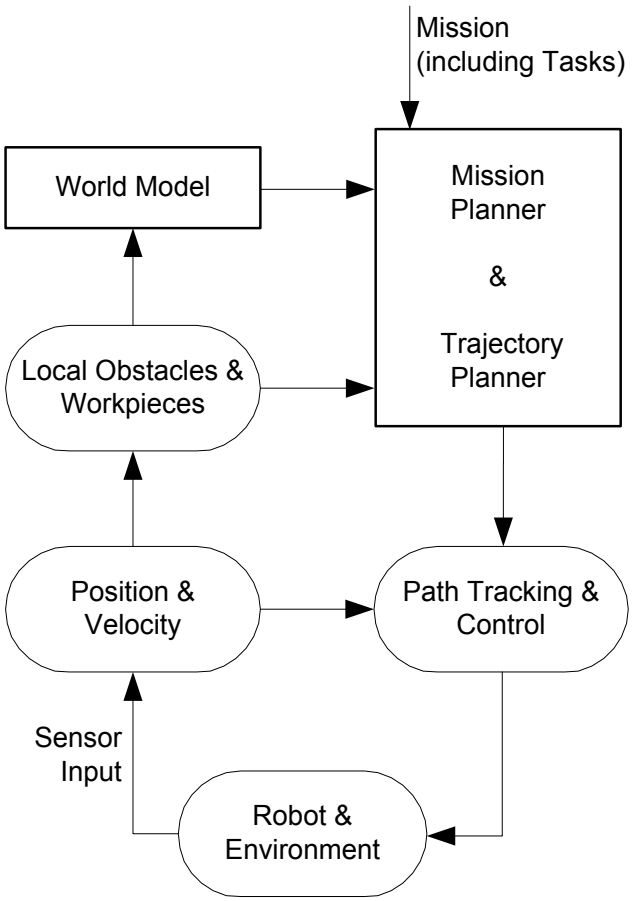


Figure 66: The Mission and path planning control loop.

Path planning in robotics considers model-based and sensor-based information to capture the environment of the robot. Perception, which is initiated by sensors, provides the system with information about the environment and interprets them. Those sensors include cameras or tactile sensors that are often used for robot manipulators. The application of the control loop to the real environment results in the general overview given in Figure 67.

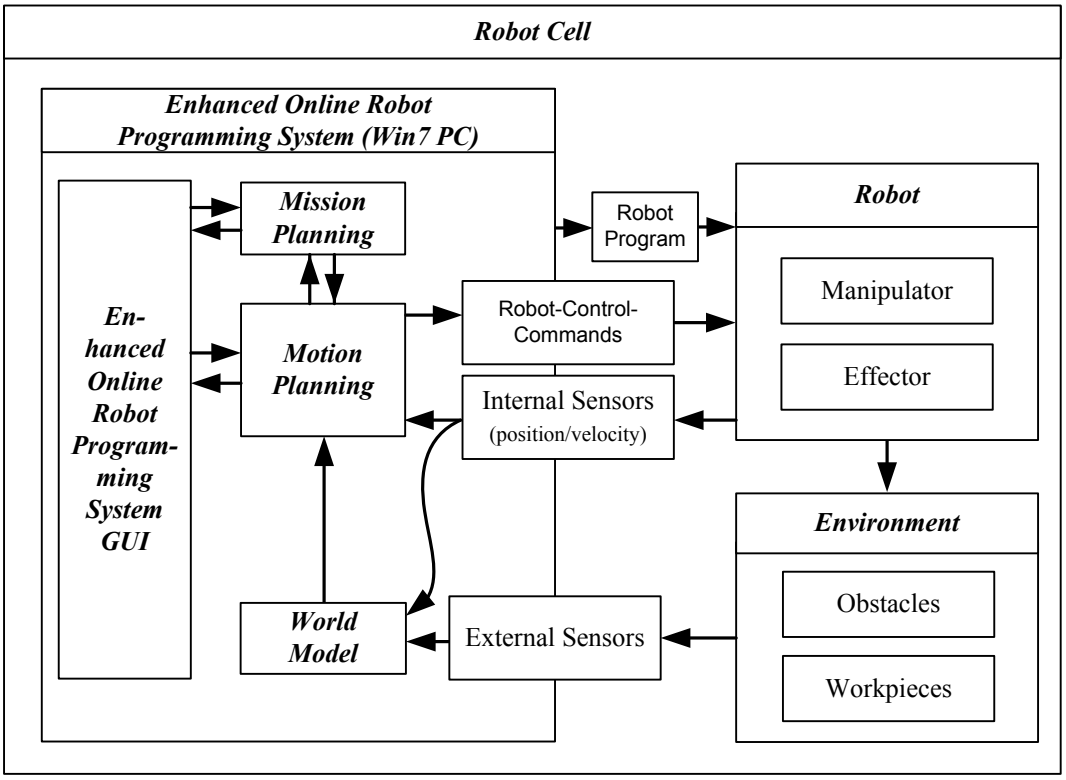


Figure 67: Logical view of the support system.

8.4.2 Mission Planning

For a given mission, the mission planner plans multiple application trajectories. Any algorithm that is used to solve the travelling salesman problem (Russell and Norvig, 2002) may be utilised to calculate the order in which each application path is processed. The mission planner delegates the task of trajectory planning to the path planner. Both the mission and path planner have to establish an interconnection for the exchange of information which is the length of the actual planned path, as shown in Figure 68.

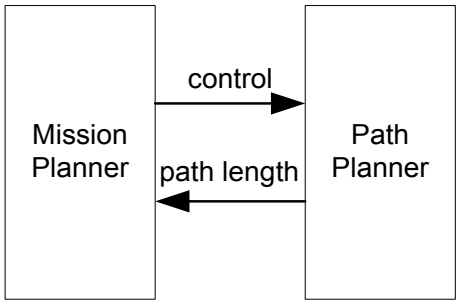


Figure 68: Path length information exchange.

The applied mission-planning algorithm has to be capable of handling path length information during path planner execution, and it has to react by instructing the path

planner. In the proposed system, a brute-force algorithm was used, and as such, it is desirable for a demonstration system, although it is limited to operation with few application paths.

Mission and path planning was based on the object model presented in Figure 69. A mission consists of one start and one target location as well as a number of application paths. A path is subdivided into roads that connect the start and the target, in addition to crossing locations and application locations. The final trajectory is the result of the path and trajectory planning calculations. An application path may also contain application information, e.g. movement type, application type, colour and other information required for spraying, painting or other tasks.

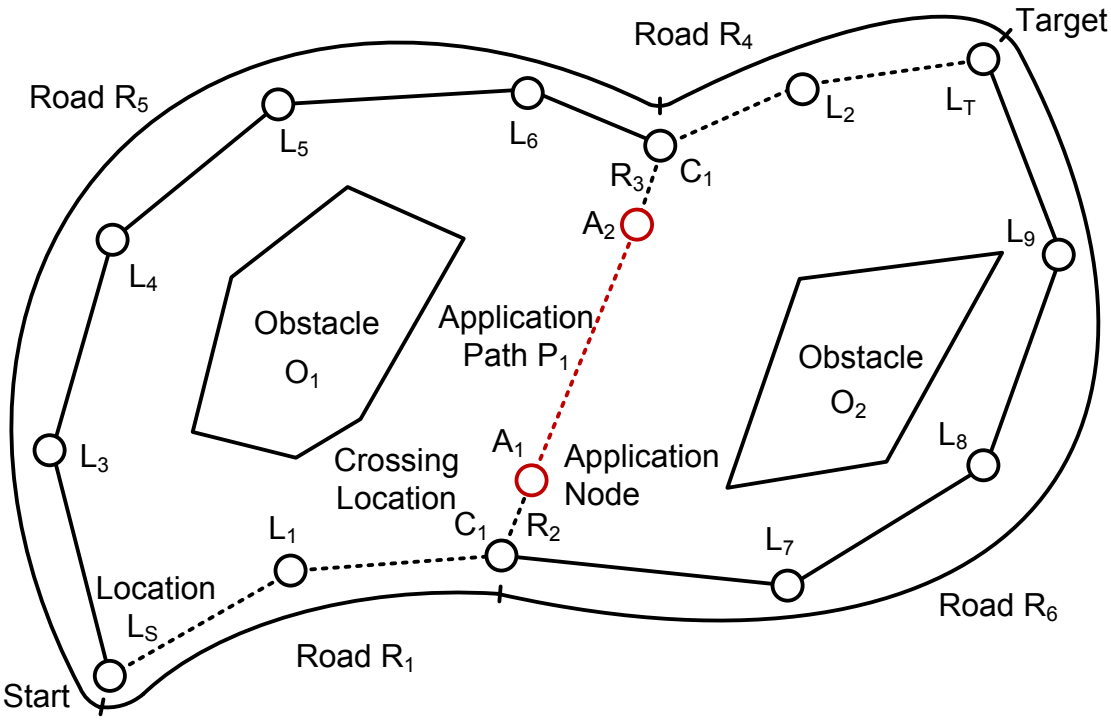


Figure 69: Definition of the roadmap elements. A_1 and A_2 set the start and end location of the application path.

To accomplish a mission, the optimal route must be found that connects each application path from the start to the target location, as illustrated in Figure 70.

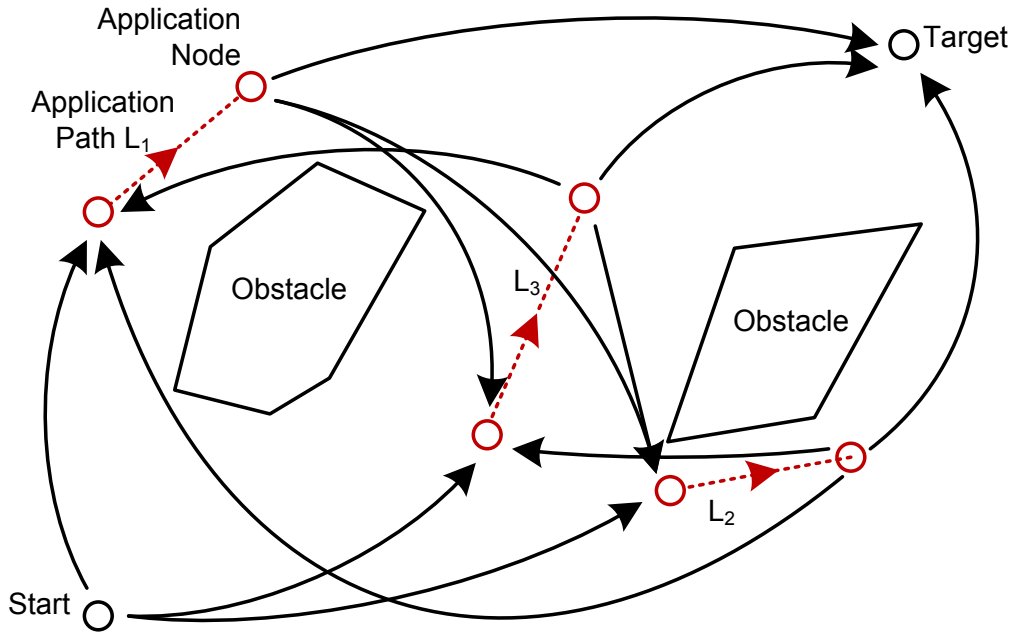


Figure 70: Illustration of possible path connecting three application path for mission task planning.

Because no exact data is known beforehand, the path distances between each sub-goal are not known, and may be estimated and subsequently calculated by trying to connect each sub-goal with each other using robot movements. In the case of lines, the end of the application lines must be fully connected to the target location and starting locations of other application lines. An example can be given with a mission that consists of three lines for a welding application (L_1, L_2 and L_3). The resulting combinations (in this case 12 connections) have to be planned to achieve the connection length for mission planning, as illustrated in Table 7.

to / **from**	**Goal**	$\mathbf{L1}_{Start}$	$\mathbf{L2}_{Start}$	$\mathbf{L3}_{Start}$
Start	O	X	X	X
$\mathbf{L1}_{End}$	X	O	X	X
$\mathbf{L2}_{End}$	X	X	O	X
$\mathbf{L3}_{End}$	X	X	X	O

Table 7: Path combinations.

The lengths of linear and circular application paths are known beforehand. The connection length is the length of the trajectory that connects two locations. This is found with the path planner, which tries to connect these locations. Once a path is connected, complete path length information is available, which can be utilized for mission planning.

8.5 Trajectory Planner

The industry requirements, which were defined in Chapter 5, specify a path-planning system that produces readable and changeable robot programs. Industrial robot programs often consist of circular, linear and joint movement primitives. The robot program itself consists of a minimum number of locations and movement primitives, as well as changeovers of movement primitives. Usually, the joint movement is the most desired one because it represents a short and fast movement type. All axes are in motion at the same time, and the motion is coordinated so that the movements of all axes end at the same time. Although this is the most favourable form of movement, it has a disadvantage in that it is not predictable for the operator during robot programming. The joint movement primitive has not yet been considered, and has been included in future work. Therefore, focus has been given to the circular and linear movement types.

Path planning generally relies on inexact data of the robot and the environment, which are stored into the in-memory world model with the help of sensor information. Vision may help to increase the knowledge of the world model. The world model employed in this study was introduced in Chapter 6. The mathematical treatment of forward and inverse kinematics, as well as the control of the employed robots was presented in Chapter 7.

The interaction of the path planner and the mission planner are described in Section 8.4. While the path planner focuses on the creation of the trajectory, the mission planner handles a higher level of path planning. The path planner calculates a path, and controls the robot along that path until a collision is detected, the kinematics constraints are not met or until the target is reached. In each case, the updated path length information is delivered to the mission planner, which re-plans the mission on a higher level.

A robot trajectory is a path in the working space of the robot. Each point on the path is described as a vector with the position and the time. The trajectory planning task is to find a collision-free movement of the robot from the start to the target location, considering the motion constraints of the robot (e.g. a car that cannot move sideways), while also satisfying the requirements for readability, maintainability and changeability of the derived robot program.

The presented algorithm is executed in three steps. First, it analyses the topology of the working space to create a roadmap with the Voronoi-based approach described in

Section 8.5.5, which also considers obstacles and the reachability of the utilized robot. This roadmap is employed in the second step to find the shortest path connecting the start and target locations. At this stage, the found solution path does not fulfil the requirements of the robot program features. Thus, in the third step, the solution path is adapted, modified and smoothed to represent a trajectory with basic circular and linear movement primitives.

The general trajectory planning workflow is presented in Subsection 8.5.1. The robot manipulator reachability and discretization of its configuration space are discussed in the Sections 8.5.1 and 8.5.3, respectively. Path planning with exact search algorithms are generally time consuming, and approximation methods have therefore become more important. Neural networks have demonstrated good approximation capabilities and are analysed in Section 8.5.4 to be employed for path planning. Neural network path planning results have shown that the principles identified in this way may also be employed for a cell-based path planning approach, which is detailed in Section 8.5.5. For path planning with dynamic obstacles, the state time space was considered, and is detailed in Section 8.5.7. The transformation of a given path to a trajectory by concatenating circular and linear movement primitives with the help of particles is explained in Section 8.5.8.

8.5.1 The General Trajectory Planning Workflow

The robot manipulator was considered based on the manipulator model, which is described in Subsection 7.2.3. The robot is steered from the start to the target location by real robot movements along trajectories.

The trajectories were generated by calculating the shortest path within the roadmap joint positions from the start to the goal. In a subsequent step, the identified path was transformed to a trajectory consisting of movement primitives. Transformation into a trajectory was achieved by applying equidistance, rotation and shrink forces on the joint space positions (Kohrt *et al.*, 2006b). This lead to a trajectory formed by canonically ordered movement primitives, which had linear and circular movements. The trajectory generated in this way avoids obstacles and reduces their clearance.

A linear octree (Gargantini, 1982b) was utilized to represent the working space of the robot and a roadmap in a spatial $\mathbb{R}^3$ in-memory world representation. Information concerning the environment in which the robot operates, including obstacles, was captured within the octree. The octree was improved during trajectory planning with real sensory

information, which is delivered in the form of collision locations. The improvement resulted in an adaptation process of the octree, which was primarily aimed at the generation of a roadmap approximating the Voronoi form.

Finally, the robot was moved along the found trajectory until either a collision or a robot kinematic constraint violation occurs, a shorter path is found by the search algorithm, or the target is reached. This often triggers a re-planning of the trajectory if a shorter path is recognised. Because real robot movements are involved, this should not happen too often. To prevent this, a hysteresis is applied. The hysteresis was also utilized to employ an additional exploration of the workspace, which improves the knowledge of the world model.

The general workflow of path planning is illustrated in Listing 6.

1) Create connectivity in the form of an approximated Voronoi form
2) Explore the workspace and update the world model
   a) Automatic random exploration
   b) Exploration by existing robot programs
   c) Exploration by following the Voronoi lines to the target without path smoothing
3) Apply the path-searching algorithm in joint space
4) Apply the elastic net algorithm to generate the trajectory
5) Move along the trajectory from the start to the target until a constraint violation occurs (collision or robot kinematic constraint), a shorter path is found by the path searching algorithm or the target is reached
   a) On collision or kinematic constraint violation
      i) Update the roadmap and generate new roads
      ii) Take back the last movement to the last common trajectory position that is unchanged
   b) On the shorter path found in the roadmap
      i) Continue the movement to explore the workspace along the possible trajectory solution until the path length difference is larger than a hysteresis value
      ii) When the path length difference is larger than the hysteresis value, do an automatic random exploration
      iii) Take back movement to the last common trajectory position of the old and new trajectory and continue with 5)

Listing 6: The support system execution tasks.

8.5.2 Discretization of the Configuration Space

The configuration space of an articulated robot is often discretized in order to execute a path-searching algorithm on the discretized search space. The discretization plays an important role since the accuracy of the search algorithm is often coupled with the accuracy of the discretization.

An example graph of a discretized configuration space is shown in Figure 71. The discretization of the movement range without constraints of the axes are practically feasible only for robots with a low number of axes, for example less than four joints.

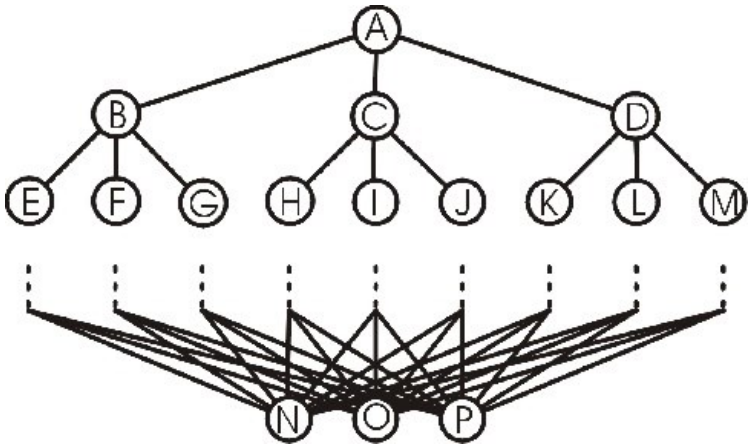


Figure 71: Graph of a discretized configuration space.

Various approaches for discretizing the configuration space have been reported in literature. The approaches reported by Reif and Wang (2000), Yang *et al.* (2011) and Zacharias *et al.* (2007) use hierarchical structures, capability maps or non-uniform discretization to optimize the search space to enable efficient searching. Thus, optimization can generally be reached by minimizing or ordering the search space specifically for the applied search algorithm.

The planning algorithm described here was executed in the constrained configuration space to improve the search algorithm. The reachability of the robot was required to calculate these constraints. In addition, the mechanically valid positions were utilized to minimize the discretized space.

Henrich *et al.* () and Reif and Wang (2000) describe an optimal discretization approach that sets the resolution along each configuration coordinate (robot axis) according to the maximum movement of the robot end-effector for each step that the robot moves along this coordinate. The discretization resolution is determined with $\Delta \mathrm{q} = (\Delta q_1, \dots, \Delta q_D)$ of a D-dimensional configuration space. A uniform discretization for all joints of the robot manipulator can be defined with $\Delta q_i = c$ for some constant c.

With a reasonable joint resolution of 1°, the uniform discretization results in very large configuration spaces. For example, a discretization of the joints of the Mitsubishi RV-2AJ with $\Delta q = (1°, 1°, 1°, 1°, 1°)$ results in a configuration space with $89.42 \cdot 10^{10}$ states.

The algorithm presented in this work is based on equation (46), where l_i is the distance between the centre of joint i to the farthest point to which the end-effector can reach, and $MaxMove$ is a pre-set distance that the robot may move for one step along the coordinate.

$$\Delta q_i = 2 \cdot \arcsin \frac{MaxMove}{2 \cdot l_i} \tag{46}$$

The optimal discretization results in Cartesian movements Δx_i of joint i, which meet the condition $\Delta x_{max} \leq MaxMove$, where $\Delta x_{max} = max\{\Delta x_i, \forall i\}$. For $MaxMove = 10mm$ of a Mitsubishi RV-2AJ, the optimal discretization equals to equation (47).

$$\Delta q = (0.80°, 0.98°, 1.73°, 3.33°, 5.73°) \tag{47}$$

The size of the corresponding configuration space obtained considering the mechanical constraints of the utilized Mitsubishi robot is $3.42 \cdot 10^{10}$ states. This is 1.89 times less than the uniform discretization with $\Delta q = (0.8°, 0.8°, 0.8°, 0.8°, 0.8°)$ and $264.99 \cdot 10^{10}$ states.

To apply an octree with a length l of $5\,m$, a depth of $d = 5$, and a cell size of $s = 15.63cm$ on the highest accuracy level, the $MaxMove$ parameter must be set accordingly. Because each cell should have at least $n = 5$ points, the $MaxMove$ parameter was set to $MaxMove = s/n = 3{,}126cm$. The calculation results in $1.14 \cdot 10^8$ states for the optimal discretization, as opposed to $88.88 \cdot 10^8$ states for the uniform discretization. Table 8 compares the non-uniform and uniform discretization values.

Robot Arm Link Number i	Link Length [m]	l_i [m]	Optimal Discretization [°]	Uniform Discretization [°]
1	0.13	0.712	2.51	2.51
2	0.25	0.582	3.08	2.51
3	0.16	0.332	5.40	2.51
4	not available			
5	0.072	0.172	10.42	2.51
6	0.1	0.1	17.98	2.51

Table 8: Optimal discretization compared to uniform discretization.

8.5.3 Reachability Calculation

The reachability of a robot in world space can be calculated by transforming the robot configurations from the tool centre point coordinates to world coordinates, or vice versa. This transformation can be applied with forward or inverse calculations of the robot kinematics. An efficient inverse calculation can only be achieved for world coordinates

with given information regarding its position and orientation. Because the orientation can be arbitrarily chosen, inverse calculations lead to intensive computation.

This problem was studied in (Yang *et al.*, 2011, Zacharias *et al.*, 2007), and a simple pre-calculation step was proposed to generate and to preserve the required information in a look-up table by performing forward calculations of the robot arm configuration to the points in space. The look-up table may generally be used if the robot kinematics is static and known beforehand. However, since this algorithm is used in an industrial environment, both statements are fulfilled. The aim of the look-up table is to represent the reachability using a limited number of joint positions P_{RJ} to reduce the search space for a path-searching algorithm. The number of joint positions P_{RJ} has a direct impact on the running time of the path-searching algorithm and the required pre-calculation time of the look-up table. The limitation exists because of the employed search algorithm described in Subsection 8.5.6.

The implemented linear octree O - the world model - has a defined depth d, which enables the calculation of the smallest octree cell size. This can be further employed to estimate the robot link dependent accuracies a_n, which have to be carefully chosen. To guarantee that the path-searching algorithm will successfully complete the search task, a sufficient number of discretized positions P_{RJ} are ensured to be stored for each octree cell on the deepest level.

The octree accuracy does not need to be very high because the employed trajectory planning approach discussed in Subsection 8.5.6 only applies to the octree for path searching. The trajectory generation algorithm actively requests additional positions, and operates almost independently from the octree.

8.5.4 The Neural Network Based Roadmap Approach

Neural networks have the ability to approximate, which may be utilized to produce a new path-planning system by combining roadmap generation and path finding algorithms. Problem dependent neural network types such as feed forward, self-organizing and Radial Basis Function networks were analysed (Russell and Norvig, 2002). However, the Kohonen map (see also Appendix F) was chosen because it is an unsupervised learning self-organizing map which directly maps the neurons to the configuration space of the robot, producing a similarity graph of input data. It represents the connectivity and the

probability distribution with its topology-preserving feature. The weight vectors are adapted and moved towards the input vector. This unfolds in an approximated robot configuration-space model represented by the neural network.

The Coloured Kohonen Map

An extension to the self-organizing map is proposed in literature with the Coloured Kohonen Map (Vleugels *et al.*, 1993), which approximates the obstacles and the free working space using two node types. One type of node approximates the obstacles while the other type approximates a roadmap in the free space. The free space is represented by connected points within the free space, and form a roadmap on which the robot may move along the connected edges. The roadmap is optimized to reduce the complexity and to compute a topological map in Voronoi form. Obstacles and non-reachable areas in the configuration space, which exist due to mechanical and geometrical constraints of the robot, were stored within the world model, and they are automatically considered in roadmap generation. The neural network consists of neurons that are generated at the beginning with an initial position distribution. During learning, the weight vectors are adapted and nodes are added. Thus, it is a growing neural net changing its architecture during runtime.

Extensions to the Coloured Kohonen Map

The aim of the neural network is to approximate the obstacles, generate a roadmap, find the shortest path and create a trajectory. This may be achieved by modifying the Coloured Kohonen Map algorithm, which also has to be extended to allow its application in multidimensional spaces for robot arms.

The visualization of the in-memory computer model of the workspace is shown in Figure 72. The outer box and the grey polygons inside the figure are assumed obstacles, and the dots are locations of the robot in the configuration space.

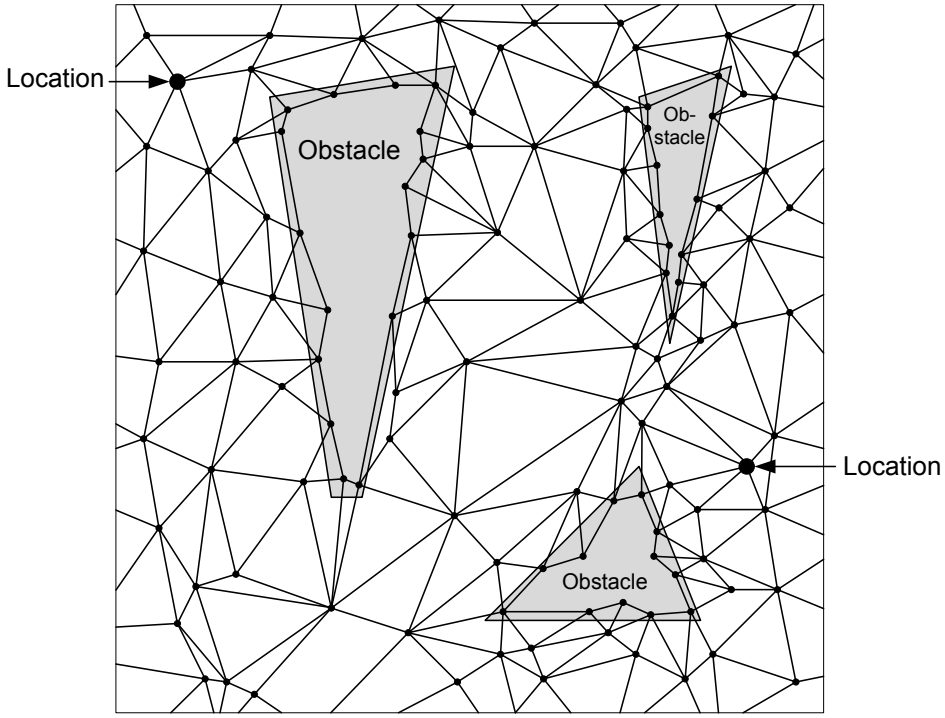


Figure 72: Workspace approximation of the obstacles and the free space with robot configuration locations.

Integration of Forces

In general, the weight adaptation rules of the Colored Kohonen Map are applied on the winner node including its neighbouring nodes (Blackmore and Miikkulainen, 1993, Cheng and Zell, 1999, Fritzke, 1995, Fritzke, 1991, Fritzke, 1993, Fritzke and Wilke, 1991, Ivrissimtzis *et al.*, 2003, Vleugels *et al.*, 1993).

The weight adaptation equation (48) (see also equation (120) in Appendix F) has been extended by Vleugels *et al.* (1993) to create a coloured version of the neural network. This has mainly been accomplished by modifying the weight adaptation term $(x - w_{bmu}^n)$ of equation (48).

$$w_{bmu}^{n+1} = w_{bmu}^n + c(t)h(s, w_{bmu}^n)(x - w_{bmu}^n) \tag{48}$$

This extension was further used to integrate additional weight adaptations which represent forces on the nodes. Thus, the first extension by Vleugels *et al.* (1993) is a force to generate a roadmap in the Voronoi form, which is used to find a shortest path from the start to the target location. The second extension is a force to approximate the obstacles.

As illustrated in Figure 73, the Coloured Kohonen map applies only forces on unsafe nodes if the input vector, which is illustrated as a cross in the figure, is safe, so that the

nodes move towards the safe position without violating the obstacle boundary. If the input vector is unsafe, forces on safe nodes are applied so that the particle moves away from the unsafe position.

An update mechanism iterates through the neurons in the original Kohonen-map learning algorithm, which was optimized by iterating only through nodes, where changes are noted. Because the calculation of all neurons was computationally intensive, further improvements may be achieved by local calculations to allow parallel processing of the nodes.

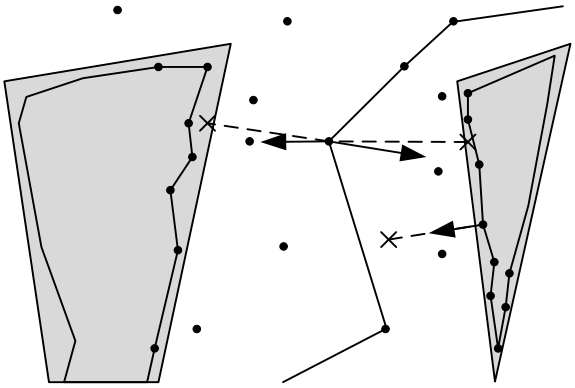


Figure 73: Forces on the safe and unsafe nodes for random inputs, marked as crosses.

Additional developed forces, explained in Subsection 8.5.8, were applied to transform the so derived shortest path to a trajectory. Trajectory generation is performed using the path, and by modifying this path to conform to non-holonomic movement constraints for the manipulator model described in Subsection 7.2.3.

Node Movement

Weight adaptation results in movements of the particles and may violate constraints, e.g. when a roadmap node collides with an obstacle. Care was taken for collisions of safe nodes with approximated obstacles, which are represented by unsafe nodes. Collision checks were performed by simple vector-vector (2D) or vector-polygon (3D) collision checks. The movement of a node does not violate the border of its neighbouring nodes. Those checks have only been applied on edges and polygons of adjacent nodes to reduce processing time. The movement vector that collides with an edge or polygon must be recalculated so that its direction is parallel to the edge or polygon surface, allowing a drift along the obstacle boundary. The calculations can be found in Appendix G.

Architectural Node Adaptations

Architectural adaptations within the neural net, following the rules given in literature (Fritzke, 1995, Fritzke and Wilke, 1991), change the connectivity and the number of neurons. Nodes are added in areas having low accuracy of environment approximation, and edges are adapted to fit the new architecture.

The architectural changes also include edge removal and addition. An edge is removed if two unsafe nodes are connected and have no common neighbour. If the safe node loses all of its edges, it is also removed. If new nodes are added, the connectivity to its neighbours is built by new edges.

The adaptations of the neural net were separated into scene-based and error-based modifications. Black nodes are generally unsafe nodes, white nodes are safe nodes and grey nodes are the new nodes. Error-based modifications are executed after k iterations during the neural network learning process. A new node is generally placed between the node with the highest error and its furthest safe neighbour. A second node is generally placed between the node with the highest error and its furthest unsafe neighbour.

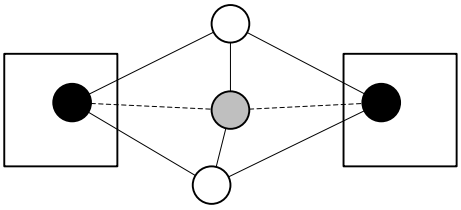


Figure 74: Error-based safe node addition.

A new node is placed on long edges between the node with the highest error and its furthest safe neighbour with two common safe neighbours (Figure 74). The new node is also connected to all common neighbours.

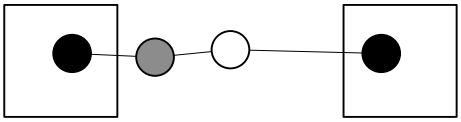


Figure 75: Error-based safe node addition.

If no such neighbour exists, a safe node is added on the edge to its furthest unsafe neighbour. It is connected to both unsafe and safe nodes (Figure 75). Nodes that are near to the boundary are not changed (Figure 76).

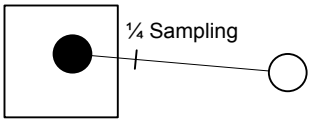


Figure 76: Unsafe nodes on the boundary.

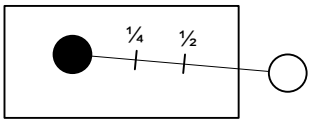


Figure 77: Error-based safe node addition.

An unsafe node is added at the ½ position and is connected to all neighbours when the ¼ and ½ samplings are both unsafe (Figure 77).

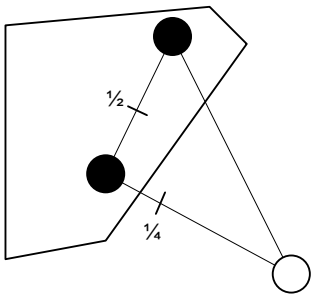


Figure 78: Error-based safe node addition.

A new unsafe node is added at ½ to the furthest unsafe node that has at least one common safe neighbour when the node is located on the boundary. Nodes are on the boundary when the ¼ sampling is safe (Figure 78). The new unsafe node is connected to all neighbours. If the node is not on the boundary, an edge is added halfway to the furthest safe neighbour when that configuration is located in the unsafe space.

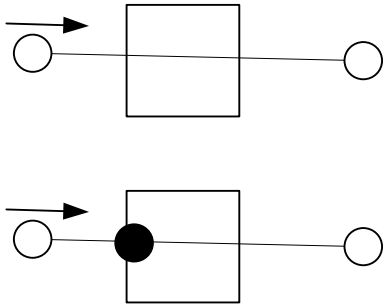


Figure 79: Scene-based unsafe node addition.

Scene-based unsafe node addition takes place at the collision configuration when a safe node is pulled by a safe node into forbidden space. The new unsafe node is connected to all neighbours.

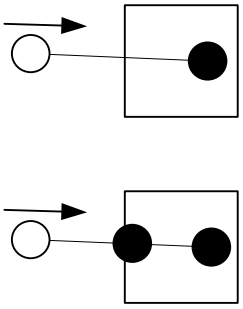


Figure 80: Scene-based unsafe node addition.

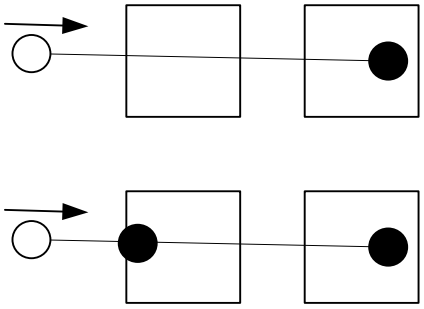


Figure 81: Scene-based unsafe node addition.

If a safe node is pulled by an unsafe node into forbidden space, an unsafe node is added at the collision configuration and it is connected to all neighbours.

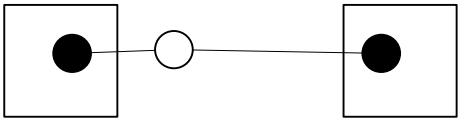


Figure 82: Scene based safe node addition.

If an edge between two unsafe neighbours is partially within safe space, a safe node is added. This is tested with three random tests along their connection.

Roadmap Simplification Forces

The resulting complex roadmap is illustrated in Figure 83. However, no robot constraints are considered, and it is assumed that the robot is a freely navigable point robot.

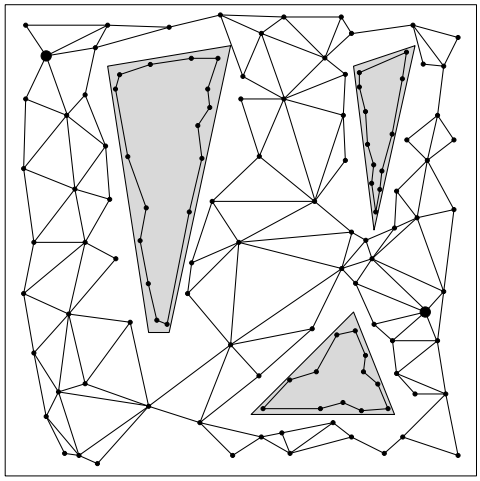


Figure 83: Workspace approximation of the obstacles and the free space.

Roadmap generation forces simplify the roadmap with the aim of reducing the number of nodes and straightening the roads. The roadmap then represents the connectivity of the space and forms the topological map, as shown in Figure 84. The Voronoi form was installed with the aim to maximize the clearance of the robot to all obstacles during robot movement.

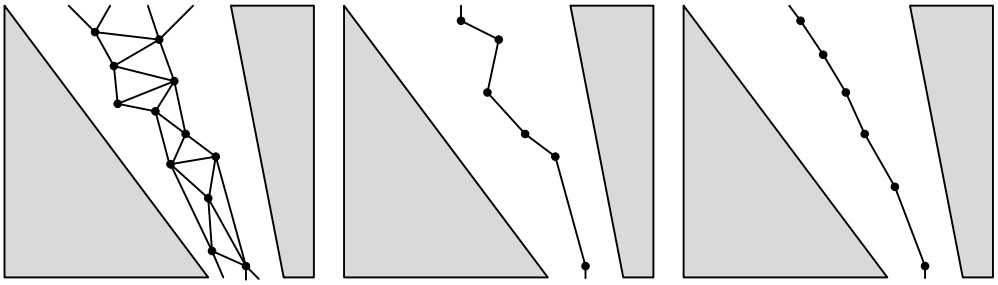


Figure 84: Simplification of a complex roadmap.

An improvement to the trajectory generation may be achieved by shrinking the connections of the nodes, and by application of the trajectory generation elastic net forces. Then, the path no longer follows the Voronoi diagram, and moves nearer to obstacles. Again, the nodes are not allowed to change their type (safe/unsafe).

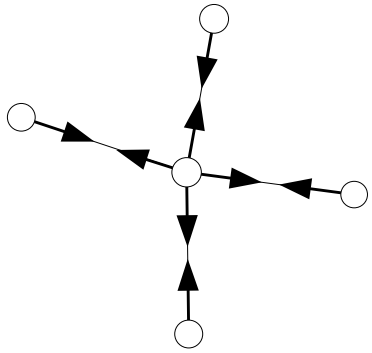


Figure 85: The shrinking forces.

Shrinking forces apply on all nodes with either more or less than two connections, but not exactly two connections. Nodes with two connections are in the desired form, and do not need to be further collapsed. This will shrink nodes that are either alone or individually connected, such as end points or multiple connected nodes. Simplification is realized by collapsing nodes until only two connections remain for every node. Nodes that represent mission locations such as the starting location of an application path are excluded.

For each safe node that has exactly two unsafe neighbours, the vector b is calculated with the equation (49) to move the particle in the middle of two unsafe particles.

$$b = \frac{\overrightarrow{v_2} + \overrightarrow{v_2}}{norm(\overrightarrow{v_2} + \overrightarrow{v_2})} \cdot \mu \tag{49}$$

The two vectors $\overrightarrow{v_1} = \overrightarrow{n_s n_{u_1}}$ and $\overrightarrow{v_2} = \overrightarrow{n_s n_{u_2}}$, which represent the vectors to the two unsafe nodes, were added and normalized, and finally multiplied with a small constant $\mu = 0.1$.

Summary

In fact, implementation tests of the algorithm presented by Vleugels *et al.* (1993) have shown that the Voronoi form is rarely reached. Adjustments of the parameters by trial and error, as suggested by the authors of (Vleugels *et al.*, 1993), have also not led to any improved results. In addition, real-time robot control with this kind of neural network requires processing of the neurons to adapt to the environment including the obstacles. Because random positions are not available in real environments, the proposed approach was no longer applied here.

8.5.5 The Cell Based Roadmap Approach

Roadmap methods generally identify a set of roads which may be safely travelled without incurring collisions with obstacles. The method adopted here was inspired by the approach presented in Subsection 8.5.4, which is based on the Voronoi form (Bhattacharya and Gavrilova, 2008, Garga and Bose, 1994). This choice was taken after considering two important aspects. First, the Voronoi form may be applied either in the world space or in the configuration space of the robot. Secondly, it maximizes the clearance of obstacles, so that the path-planning algorithms do not have to be particularly accurate. The second point may also be perceived as a negative characteristic, since the derived roads are not short, smooth or continuous enough to guarantee an enhancement (Bhattacharya and Gavrilova, 2007, Masehian and Amin-Naseri, 2004).

The octree stores its cells in a predefined maximum accuracy defined by the octree depth. Each cell stores a reachability value, which indicates whether or not the robot can move its tool-centre-point (e.g. the robot hand) into the cell area. The general reachability is stored in a pre-calculation step described in Subsection 8.5.3.

In addition, each cell also stores an occupancy value. Cells are defined as fully occupied, partially occupied or free, depending on the obstacles within the working space. This information is input by external sensors through the data fusion framework presented in Section 6.4. A collision button and CAD data for the construction process of the working cell were utilized in the test environment to detect obstacles. The choice was made because model data is often available, and the operator itself is a reliable source that can detect collisions. Additionally, more advanced sensors such as machine vision can also be applied to increase the recognition performance.

The occupancy and the reachability information are incorporated to create a roadmap within the reachable free space of the octree. The roadmap forms a Voronoi diagram, which is created by a cell-based algorithm within the octree.

Hence, the concept on which the Voronoi form is based was extended and applied to a grid-based algorithm. First, the obstacle and border cells are added to an open list. Then, all neighbour cells are iterated for all elements in the open list in order to mark them with the obstacle number based on the currently examined element of the open list. The

currently examined element is moved from the open to the closed list, and extended cells are added to the open list to be examined in the next iteration.

```
1. Store all border, obstacle and extended cells C in the open list
2. While open list element count > 0
   2.1.   Take first cell C_i from the open list
   2.2.   Inspect all neighbour cells of C_i and mark each extended neighbour
          cell according to the following conditions:
      2.2.1.    If the extended cell is located between two or more obstacles
         2.2.1.1.     If the cell is not reachable it is marked '0'
         2.2.1.2.     Else it is marked '-1'
      2.2.2.    Else copy the mark from cell C_i
   2.3.   Add all neighbour cells of C_i, which are not in the closed list, to
          the open list
   2.4.   Move cell C_i from the open list to the closed list
3. Wend
```

Listing 7: Cell extension algorithm.

The general grid-based algorithm described in Listing 7 produces the approximated Voronoi diagram shown in Figure 86. The primary aim is to approximate the Voronoi form between the obstacles and the border cells in configuration space.

Figure 86 represents several obstacles, unreachable configuration space cells, as well as start and target cells. The unreachable configuration space cells are equally treated as obstacle cells. The light grey cells '-1' represent the Voronoi approximation. The dark grey cells represent the unreachable configuration space. White cells denote expanded nodes, cells 1-7 denote expanded obstacle node cells, black cells denote border nodes, and cells 21-23 denote obstacles.

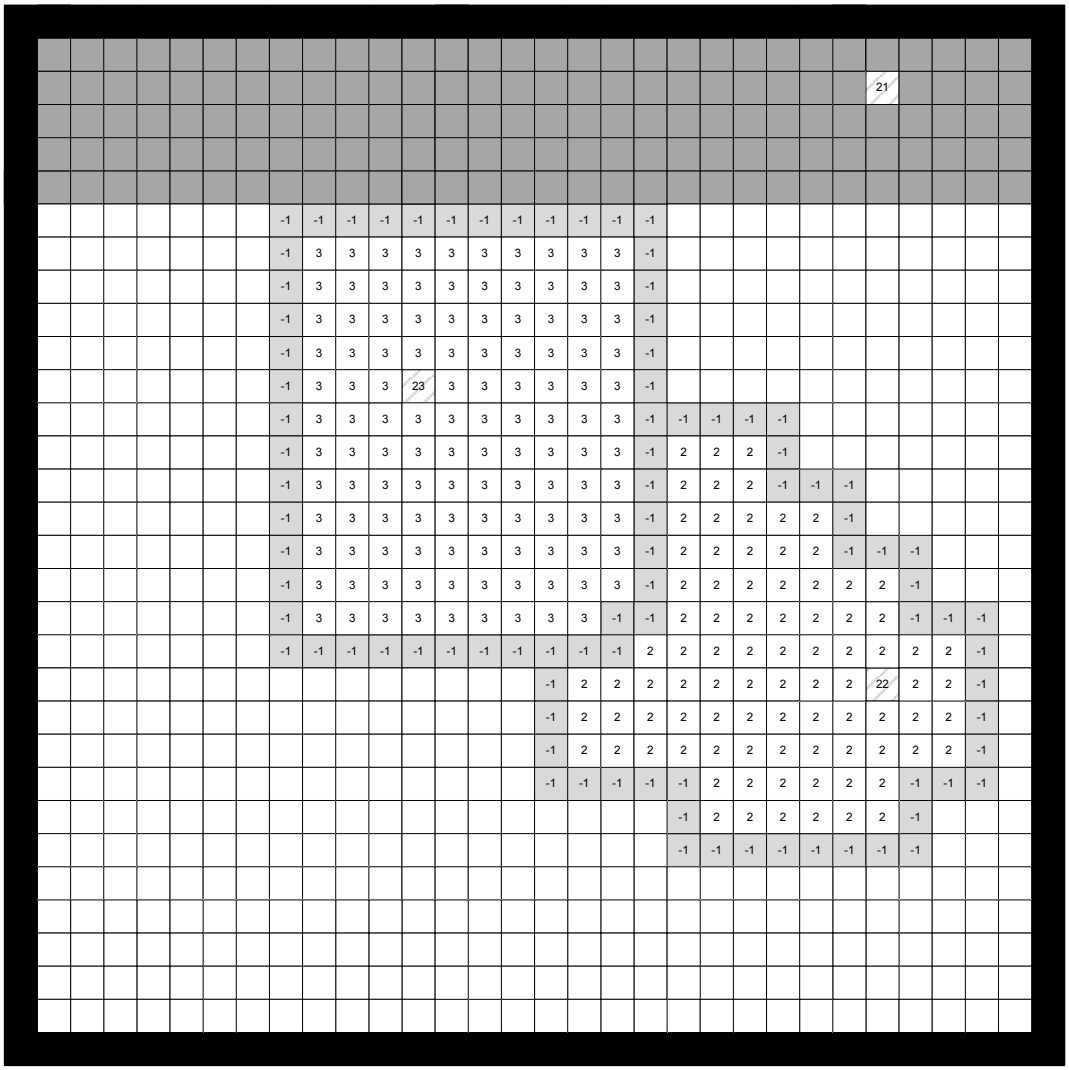


Figure 86: Voronoi approximation in a two-dimensional uniform grid.
With three obstacles 21- 23 and with the extended cells named 1-3.

The grid used in the implementation is a three-dimensional octree, which allows the addition of obstacles during runtime, while recalculation is only necessary for their neighbouring areas. The octree also provides the opportunity to use its hierarchy to speed up the algorithm. Application of this cell extension approach builds a roadmap that supports the real-time development of the topology and connectivity of the robot workspace.

This algorithm is applied to the tool centre point of the robot. The maximum clearance of the whole robot arm to the obstacles is indirectly considered because reported collision indication positions are stored as robot posture data in the cell. The cell occupancy is

always calculated based on all postures, and its occupancy value is therefore calculated accordingly.

As described in Chapter 6, the octree exhibited two limitations (Hwang *et al.*, 2003) in path planning. First, the detection of small passages requires a highly accurate octree/quadtree. Secondly, the shortest path is not always identified since the distance calculations of the cells always use the midpoints of the cells.

The first aspect requires the involvement of many cells; consequently, the planning stage may have a long processing time. Hwang *et al.* (2003) proposed the use of an obstacle dependent grid to overcome this limitation. However, the octree representation is used here to interface between world and joint space coordinates. The number of cells is reduced by the transition to the not occupied joint positions which are assigned to each cell, and by only subdividing needed cells.

The second aspect is solved using joint positions within a cell and the joint distance metric for the A* search. The joint positions deliver exact distance lengths, even on higher levels of the octree. The octree cell size is therefore decoupled from distance measurements.

Structure Based Performance Increase

The octree is a hierarchical data structure that allows the speeding up of the proposed cell extension algorithm. The cell extension algorithm is executed on each accuracy level, starting from the lowest resolution. Extended white cells on an accuracy level may be omitted for the next higher accuracy levels, and large areas of the working space are therefore quickly extended. Figures 5-9 represent the accuracy levels of each working space and the adopted cell extension method. The octree stores only the necessary cells, while the Voronoi approximation (‘-1’ cells), which is described in the following sections, is executed.

Figure 87: Level 1, edge length: $2^1 = 2$.

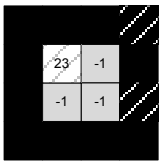


Figure 88: Level 2, edge length: $2^2 = 4$.

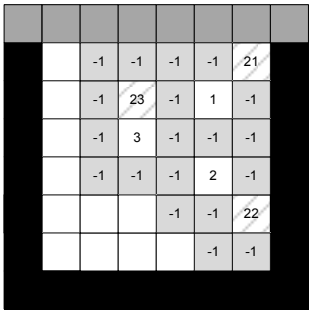


Figure 89: Level 3, edge length: $2^3 = 8$.

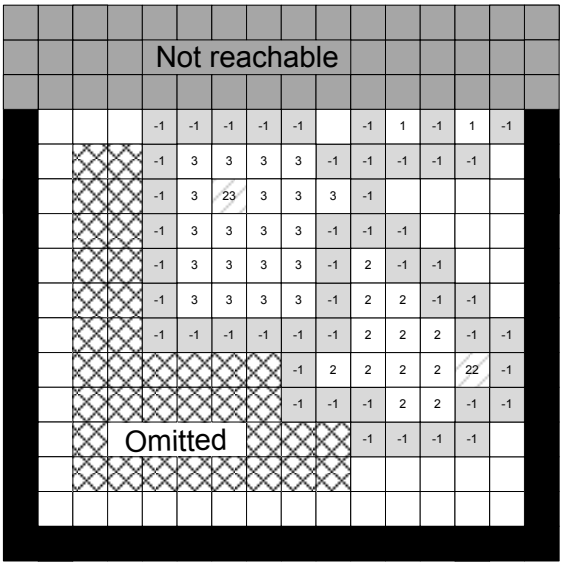


Figure 90: Level 4, edge length: $2^4 = 16$.

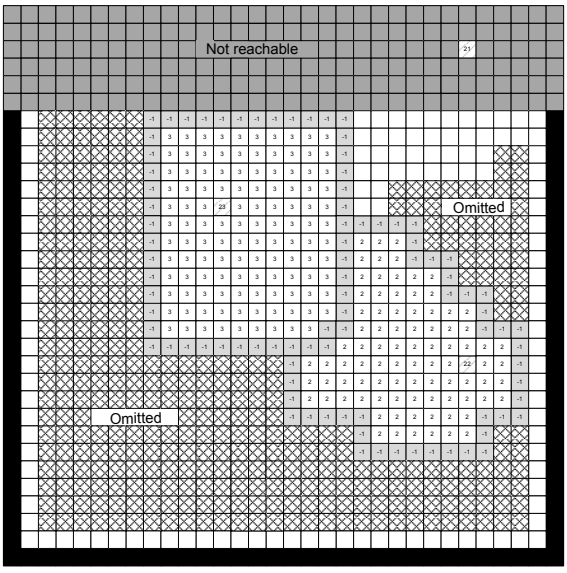


Figure 91: Level 5, edge length: $2^5 = 32$.

Obstacle Addition Mechanism

During the execution of the path-planning algorithm, new information regarding the working space and the obstacles is provided by the employed sensors and information sources, which are the collision button and the CAD model. New joint position information is added to the data structure in the steps described in Listing 8.

```
1. Get the robot posture for a collision indication
2. Execute forward calculation to get the world position
3. Store the joint position to the responsible octree cell binary tree
4. Calculate the occupation value for the cell
5. Update the parent cells
6. Recalculate the cell region d to obtain the updated Voronoi diagram
```

Listing 8: Obstacle addition algorithm.

The world coordinate of the position is determined by the subsequent forward kinematics calculation that stores the joint position into the octree cell that is responsible for the world position region.

The cell is marked by an occupation value according to the reported and fused sensor value O_{Cell}. A probability threshold of $t_P = 0.8$ is applied in equation (50) to transform the cell occupancy value to the binary value $O_{Cell_{Binary}}$ required by the Voronoi roadmap generation algorithm.

$$O_{Cell_{Binary}} = \begin{cases} O_{Cell} \geq t_P & 1 \\ else & 0 \end{cases} \tag{50}$$

Parent cells are either updated to partly or fully occupied, depending on the occupation of the child cells of the parent. Parts of the Voronoi roadmap have to be recalculated if new collision information is processed. A minimum distance d_{min} of the robot TCP is introduced to those obstacles, and is used to clear surrounding extended groups of cells within the distance d_{min}. An example is illustrated in Figure 92.

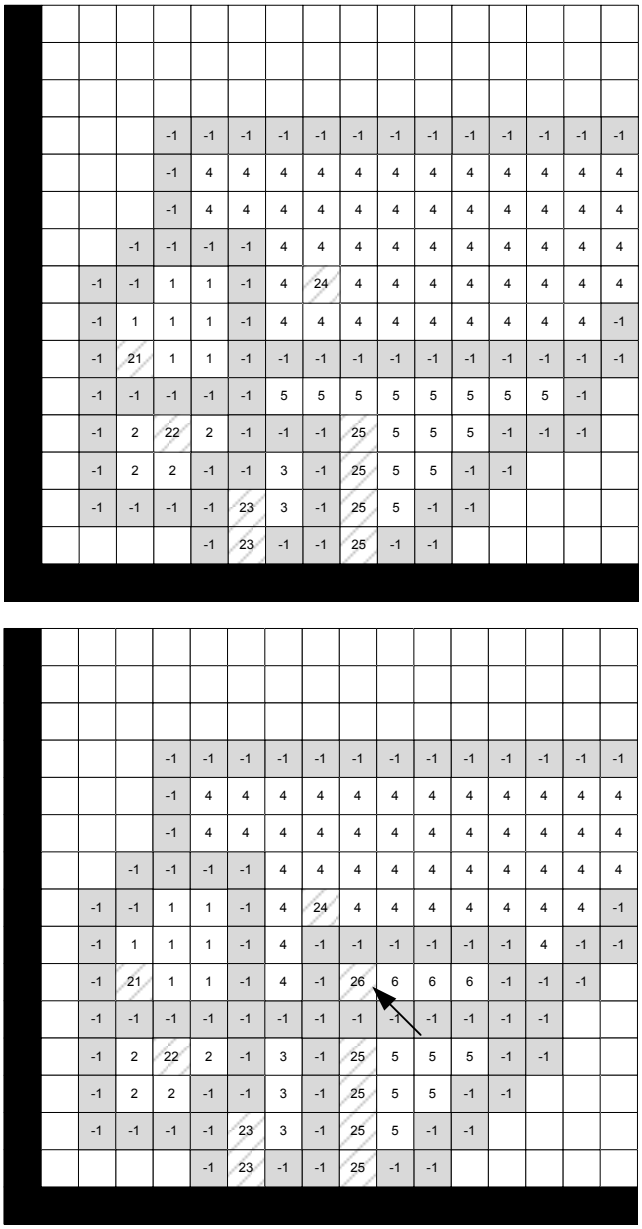


Figure 92: Dynamic and fast cell extension example (before and after update).

The cell in position (9, 6) is updated and marked as occupied (see second figure, cell number 26). A radius of $d_{min} = 5$ cells is considered. As a result, the group information and the Voronoi path are recalculated.

The second example in Figure 93 focuses on the defined distance and shows how the distance affects the Voronoi path generation. The distance to the occupied cells should be maximised within the given boundary of d_{min}. The occupied cell '27' (only its extended cells '7' are visible) is next to the newly added occupied cell '26', and the Voronoi path is therefore adapted. The guaranteed space between the Voronoi path and the newly added

cell is $d_{min}/2$ because the cell extension mechanism starts from the given distance and grows from both sides in order to meet in the middle of d_{min}.

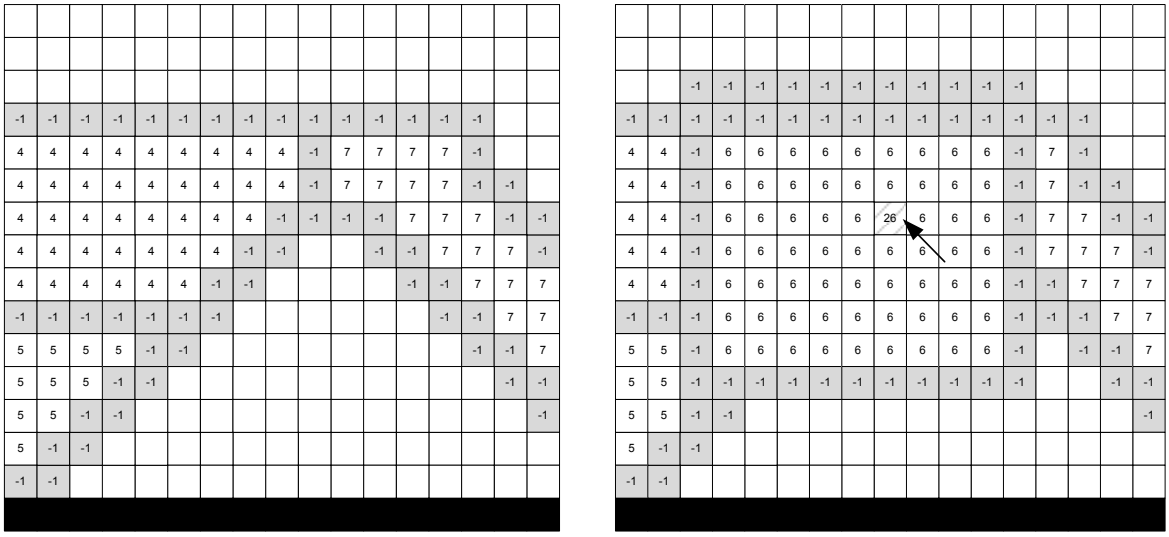


Figure 93: Defined distance influence on Voronoi path generation.

The algorithm is summed up in Listing 9, where the group information is updated for each obstacle addition.

1. Add new obstacle cell to open list
2. Reset and move cells within the distance d_{min} from the closed to the open list
3. Apply the algorithm from Listing 7

Listing 9: Cell addition for obstacles.

Roadmap Elements

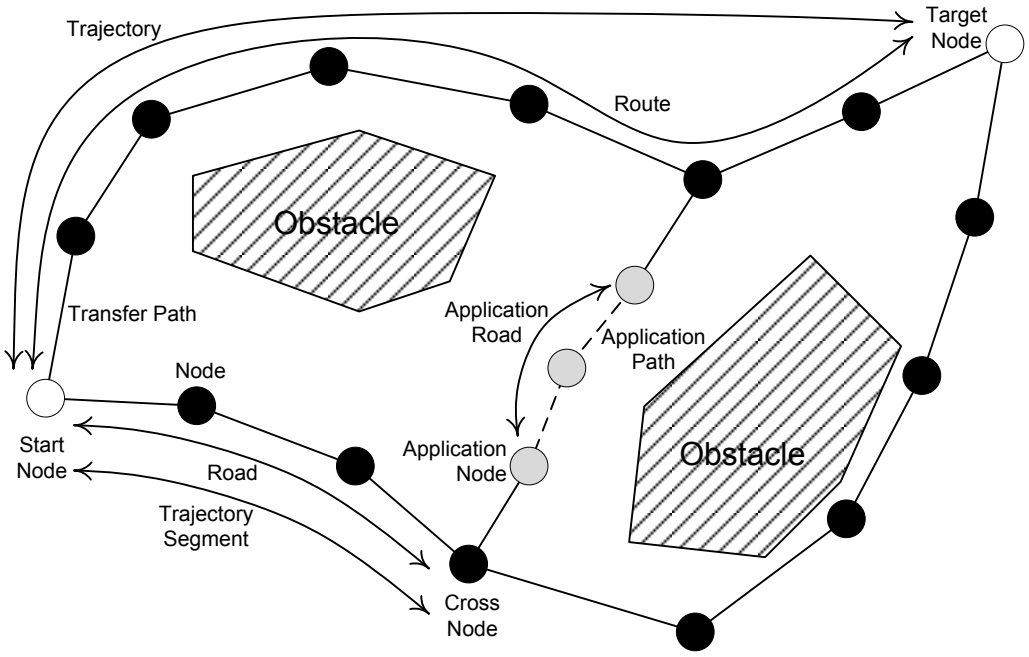


Figure 94: Roadmap elements.

The roadmap elements are represented in Figure 94. A mission provides specific mission data, such as start and target locations of application paths and additional application specific information. The roadmap consists of roads, paths and nodes. A road is a connection of two nodes that have to be start, target or cross nodes. An application node location is defined in the mission data, and is the start or target location of an application path. The connection between two nodes is a path, where two types of paths are possible: an application path and a transfer path, which is not a part of an application. An application road consists of application paths. A trajectory may be calculated from a route between two nodes. A route consists of roads. Trajectory segments are roads that are transformed into a trajectory. The roadmap was utilized to calculate the trajectory during the execution of the path planning system.

8.5.6 Search within the Roadmap

In robotics, the A* algorithm (Russell and Norvig, 2002) can be used to solve the given task of planning the shortest path in a graph. The A* always expands nodes that are considered to be the best nodes regarding its distance to the goal. It uses a heuristic that will not overestimate the distance to the target node. The A* finds the shortest path if there exists one at the given level of knowledge. The knowledge is expressed as the connectivity

of the working space. The calculation of the heuristic directly influences the calculation time of the algorithm.

The A* search algorithm is utilized to search within the joint positions of the Voronoi roadmap in order to connect the start to the target locations. During the planning of a trajectory, an improvement of the roadmap takes place with collision information to improve the approximation of the obstacles within the working space.

The start and target locations are handled as obstacles and the Voronoi roads are generated around them. The extended cells of the start and target cells are added to the search space to connect the location with the Voronoi roads.

The employed algorithm finds the shortest path with the help of heuristics to direct the search towards the target. The heuristic should not overestimate the distance to the goal. Therefore, the joint distance metric is utilized as the heuristic for the A* algorithm. The connectivity of the joint positions is given by the octree cell connectivity. All joint positions of one octree cell are connected to all joint positions of the neighbouring octree cells. This may result in high running search times if too many joint positions are stored within the octree cells. The discretization calculation described in Subsection 8.5.2 has to consider this by choosing the parameter $MaxMove$ within the equation (46) accordingly. This is highly dependent on the robot geometry.

As mentioned in Chapter 6, the occupancy probabilities of the cells and of the binary tree joint positions are considered as movement costs during path planning. Because the search is not executed within the cells, but within the joint positions, each joint position is allocated the probability given by $P = max\left(P_{Containment\ Cell}; P_{Joint\ Position}\right)$.

The connectivity of the octree cells includes direct and diagonal neighbours so that each non-boundary cell has 26 neighbours. The octree is an extension of the quadtree, which has highlighted two limitations (Hwang *et al.*, 2003) in path planning. First, the detection of small passages requires high accuracy of the octree/quadtree. Secondly, the shortest path is not always identified since the distance calculations of the cells always use the midpoints of the cells.

The first aspect requires the involvement of many cells. Consequently, the planning stage may take a great deal of processing time. Hwang *et al.* (2003) proposed the use of an obstacle-dependent grid to overcome this limitation. However, in the newly proposed approach, the octree representation is used to interface between world and joint space coordinates. The number of cells is reduced by the transition to the joint positions which are assigned to each cell, and by only subdividing the required cells.

The second aspect is solved using joint positions within a cell and the joint distance metric for the A* search. The joint distance between two joint positions is directly computed by the difference of these joint positions. The distance measurement is executed on the joint positions and not on the cells; therefore the octree cell size is decoupled from the distance measurements.

The roadmap itself is not changed during the trajectory calculation process, except for additional knowledge that has been gained during the exploration process of the robot. Exploration is always carried out when the robot moves within the working space, and additional information is stored within the world model.

The trajectory is calculated based on the found route, and it is followed by the robot. It is the most optimal trajectory based on the level of knowledge in the world model. The global optimality of the path is not yet assured, since forces are still applied to the nodes of the routes and obstacles may still be found, making the re-planning of the trajectory necessary. The system always tracks the estimated distances to the target.

Moreover, the application of the A* algorithm to a real robot results in the re-planning of the path itself each time a collision occurs. Collisions force the robot to undo its movement to the start location. Because real robot movements are involved, this should not happen too often. Therefore, an additional exploration of the working space is executed. Consequently, the system obtains environment information stored within the world model.

Together with the probabilistic occupancy map projected on joint positions, the A* path planning method always delivers the shortest roadmap Voronoi road, if one exists. The search space is reduced by the Voronoi form in world space, and the reachability calculation is dependent on the robot geometry. The joint positions are carefully distributed

along the roadmap paths. By applying this approach, good performance of the search stage is assured.

8.5.7 Obstacle Types

The proposed path planning system is based mainly on collision information from path planning during online programming with collision sensors, which could also include vision. Collision detection depends on the observed objects within the robot cell. Obstacle avoidance is based on the roadmap of the octree, which contains possible paths and trajectories to connect the start and target locations.

Kant and Zucker (1986) suggest the separation of obstacle types into static and dynamic obstacles. Dynamic obstacles within a robot cell were further subdivided considering their state-time within a production cycle. The state of an object describes its position and orientation. The state-time space is the combination of the time dimension, measured from the start of the production cycle, with the state of the object.

An object may have a predictable and defined trajectory, which may also be programmed. If this trajectory is controlled by the production control logic in a coordinated manner with the robot program, this object is timely synchronized. For example, such an object can be the door of the body of a car that is opened by the robot at a specified time in the program cycle. The production control logic normally takes input signals, e.g. when a robot escapes a defined robot cell space or from production devices, to control the workflow. These events are synchronization points, and are depicted in Figure 95.

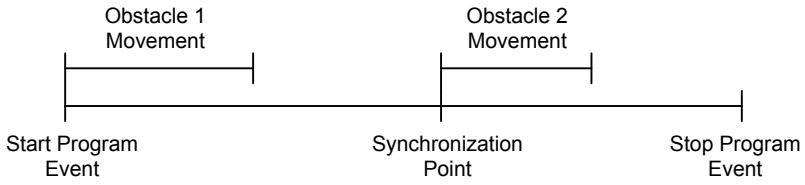


Figure 95: Obstacle synchronization.

The trajectory of an object may also be unsynchronized or not controlled by the production control logic. Together with non-deterministic obstacles, these objects are not synchronized with the robot program, and are therefore unpredictable.

Static obstacles are a special case, and are derived from dynamic, timely synchronized obstacles with predictable movements. An overall object-type definition is given in Table 9.

1. Predictable movement
    a. Timely synchronized
        i. Static obstacles
        ii. Dynamic obstacles
    b. Timely unsynchronized
        i. Dynamic obstacles
2. Unpredictable movement

Table 9: Object type definition.

The support of obstacles with unpredictable movement requires real-time collision avoidance and a permanent installation of the support system. The requirements of Chapter 5 define that the robot programming system should be removed after the generation of the program, which is static once it is generated. Therefore, obstacles with unpredictable movements are not the focus of this study.

The state time space is introduced by Fraichard (1999). It allows the transfer of the roadmap in state space into a graph in state time space by considering the time dimension. A reproducible movement can be transferred to the state-time space, and can be considered by the mission planner during path planning. Therefore, the static- and timely synchronized obstacles with predictable movements can be mapped into the state-time space. The static obstacles do not require the time dimension. The timely synchronized obstacles with predictable movements always occupy the same states in state-time space relative to the synchronization point. An additional collision indication button for dynamic obstacles could be added to the GUI for the operator to separate those two obstacle types. The state-time space may be further extended by multi-robot-operation support. Other robots may be seen as timely synchronized obstacles with predictable movements.

Timely synchronized obstacles with predictable movement require time synchronization, which may be performed automatically or manually by the operator during online programming.

8.5.8 Elastic Net Trajectory Generation

The transformation of the path to a trajectory is a necessary step that is carried out by the application of the elastic net. The path within the roadmap found by the A* algorithm consists of connected joint space positions. The transformation of the path into a trajectory was realized by applying equidistance, rotation and shrink forces on the joint space positions in world space. Thus, both the forward and inverse kinematics calculations were required.

The calculations for each particle were locally performed with no global knowledge of the trajectory. The generated result consists of canonically ordered movement primitives, which are linear and circular movements. The transformation automatically considers the reachability and obstacles.

The topology of the free working space is obtained and stored within the roadmap and its cells (including joint positions). The path-searching algorithm calculates a path that consists of particles, which are linked joint-space coordinates. Those particles have been transformed into world coordinates by simple forward kinematic calculations. The path of connected particles in world space forms the trajectory.

The Dubins car (Dubins, 1957) model (see also Subsection 7.2.3) of the robot with a bounded maximum steering angle e (see Figure 96) was employed for the two dimensional case. The robot is able to move around curves with a minimum radius of r, and along lines which represents a linear movement. No other manoeuvres are allowed. Furthermore, the robot moves only in a forward direction.

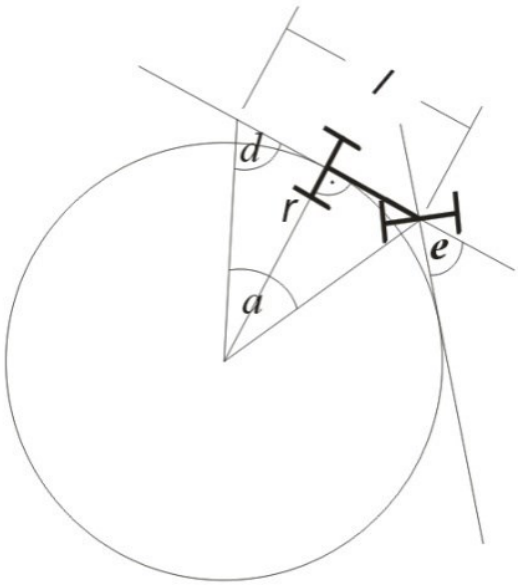


Figure 96: Correlation between e and the radius r in a circle (2D).

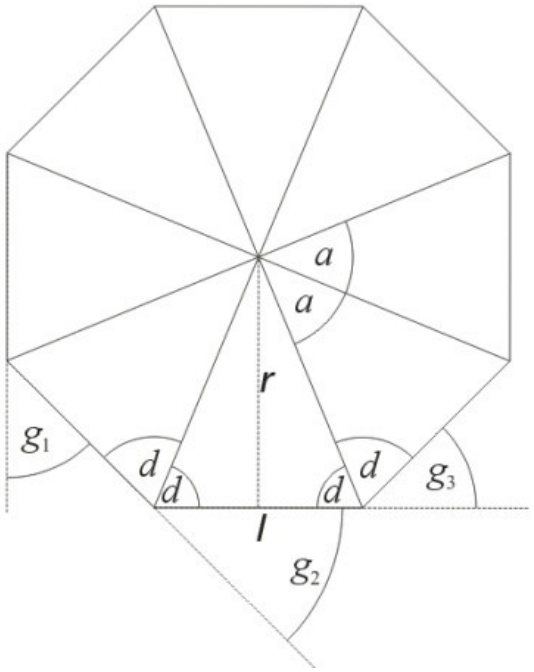


Figure 97: Correlation between e and the radius r in a polygon (2D).

Correlation between the radius of a curve and the steering angle e

The correlation between e and the radius r is shown for two cases involving a regular polygon and a circle. The former will be used later, where e corresponds to g_1, g_2 and g_3 in the ideal case. In Figure 97, the steering angle e of the real robot from Figure 96 may be compared. The formulas for the correlation of e and r are stated in (51), (52) and (53).

(51) $$a = \pi - 2 \cdot d = g_n = e$$

(52) $$e = g_n = 2 \cdot atan\left(\frac{l}{2 \cdot r}\right)$$

(53) $$r = \frac{l}{2 \cdot tan\left(\frac{e}{2}\right)}$$

Installed forces

As shown in Figure 98, three forces are installed on the particle path illustrated in Figure 99. The first force $F_{Equidistance}$ keeps the distances between the particles equidistant. The second force $F_{Rotation}$, which is actually the average of the four forces $F_{Rotation\ g_1}$, $F_{Rotation\ g_3}$, $F_{Rotation\ g_4}$ and $F_{Rotation\ g_6}$, moves the particles on a circle with the neighbouring ‘particle’ as the midpoint. The last force, F_{Shrink}, allows the path to

shrink in the direction of a straight line. The direction and value of the forces are influenced by the three neighbouring angles g_1, g_2 and g_3 (see Figure 97 and Figure 98).

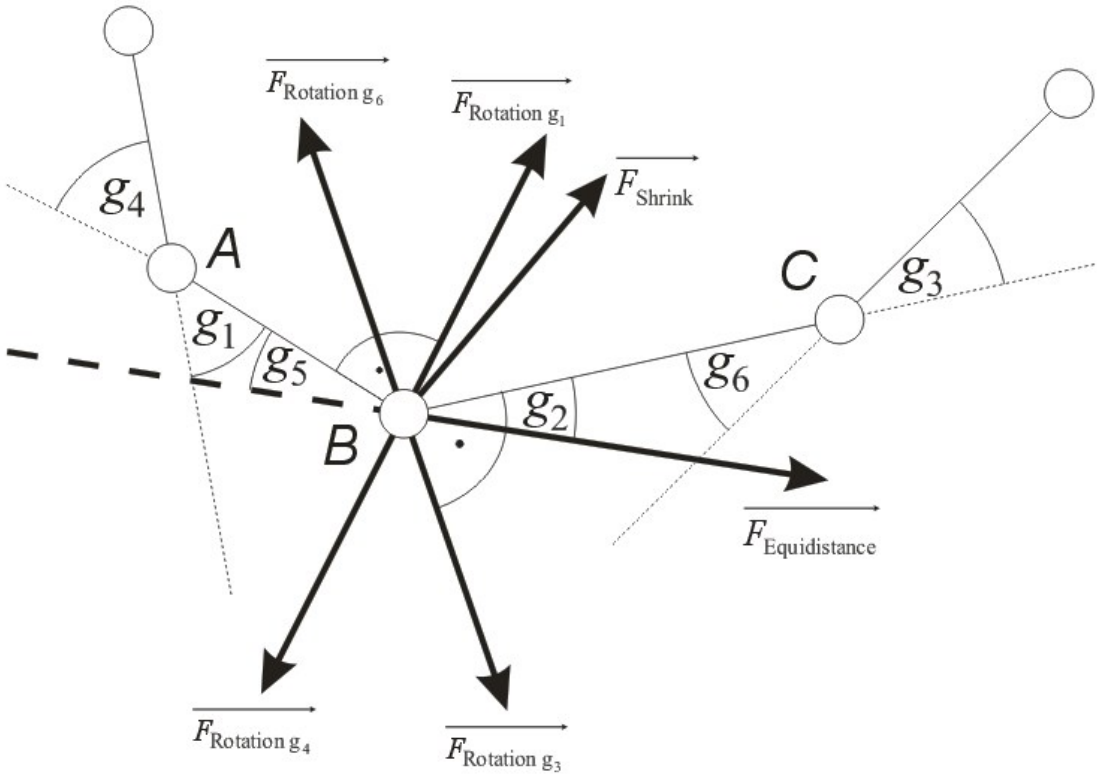


Figure 98: Installed forces.

Equidistance forces

These forces push the particles in a tangential direction. $F_{Equidistance}$ influences the other forces, especially the rotational forces, as little as possible. To reach the equidistance of all points, the tangential force is utilized. The absolute value of the force is the difference in the distance to the neighbouring points (54).

$$\overrightarrow{F_{Equidistance}} = (\overline{BC} - \overline{AB}) \cdot \frac{\overrightarrow{F_{Tangent}}}{\overrightarrow{F_{Tangent}}} \tag{54}$$

Rotational forces

The steering angle *e* (see Figure 96) may be changed at any time within its boundaries. Curves with a fixed *e* would result in circular curves. To build a circle of particles, it may be seen as a polyhedron, as shown in Figure 97. A polyhedron has straight lines between the neighbours, and a circle may be approximated by more particles. $F_{Rotation}$ attempts to

keep the angles of three neighbouring particles equal. Every line tries to minimize the difference of the angles g_1, g_2 and g_3 with a small rotation (see Figure 99).

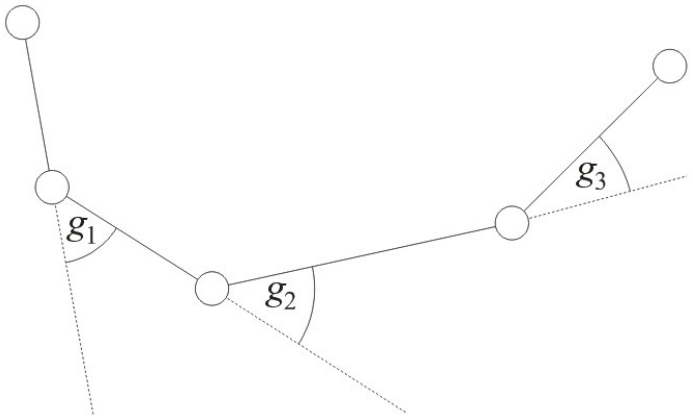


Figure 99: Angles of the rotational force.

The force of the rotation is orthogonal to its rotation axis. This leads to the formulas for the motion of point B in Figure 98:

(55) $$e_2 = \frac{g_1 + g_2}{2}$$

(56) $$e_2 = \frac{g_2 + g_3}{2}$$

(57) $$\overrightarrow{F_{Rotation\ g_1}} = \left(\frac{\overrightarrow{F_{g_1}}}{|\overrightarrow{F_{g_1}}|}\right) \cdot (e_1 - g_1)$$

(58) $$\overrightarrow{F_{Rotation\ g_3}} = \left(\frac{\overrightarrow{F_{g_3}}}{|\overrightarrow{F_{g_3}}|}\right) \cdot (e_1 - g_2)$$

(59) $$\overrightarrow{F_{Rotation\ g_4}} = \left(\frac{\overrightarrow{F_{g_4}}}{|\overrightarrow{F_{g_4}}|}\right) \cdot (e_2 - g_2)$$

(60) $$\overrightarrow{F_{Rotation\ g_6}} = \left(\frac{\overrightarrow{F_{g_6}}}{|\overrightarrow{F_{g_6}}|}\right) \cdot (e_2 - g_3)$$

(61) $$\overrightarrow{F_{Rotation}} = \overrightarrow{F_{Rotation\ g_1}} + \overrightarrow{F_{Rotation\ g_3}} + \overrightarrow{F_{Rotation\ g_4}} + \overrightarrow{F_{Rotation\ g_6}}$$

Shrink forces

F_{Shrink} is a constructed force at each ‘particle’ to build a straight line. This may be achieved by a simple vector addition of the two position vectors of the neighbours of each particle (see Figure 98) while considering the equidistance constraint.

(62) $$\overrightarrow{F_{Shrink}} = \overrightarrow{BC} - \overrightarrow{AB}$$

Forming lines

Every particle's position lies on an edge of the polyhedron. The overall force leads to a curved connection, where all 'particles' are ordered as equidistant and the steering angle *e* always lies within its boundaries. The path does not yet have straight lines. If the steering angle *e* is very small, the radius of the curve is very large and may be considered to be a straight line. The algorithm considers this to be a switch for the calculation of the positions of each particle. Shrink forces may be used to form a line. It is a simple vector addition of the two neighbouring lines of *B* to *A* and *C* (see (62) and Figure 98). A radius threshold r_{max} is introduced, which controls when the formulas for a line or a curve are used. r_{max} is the value for the maximum radius. The angle threshold $t_{a,min}$ was obtained from equation (52). If the statement $|g_n| \leq t_{a,min}$ is true for $n = 1,2,3$, the particles will be shrunk to a line. Otherwise, the rotational forces are applied.

Overall force

It is possible to construct a path from a start position to a target position with straight lines and curves with equal radius for each curve automatically. The threshold $t_{a,min}$ is the only parameter which is responsible for the decision of whether a line or a curve is to be built. If the formula (63) is applied, the path construction algorithm is divergent.

(63) $$\vec{F} = C_1 \cdot \overrightarrow{F_{Equidistance}} + \begin{cases} C_2 \cdot \overrightarrow{F_{Rotation}} & \text{if } |g_n| > t_{a,min} \\ C_3 \cdot \overrightarrow{F_{Shrink}} & \text{if } |g_n| \leq t_{a,min} \end{cases}$$

$F_{Rotation}$ is identified to produce incorrect results if the particles have not yet been ordered. The order may be measured in terms of particle movement, which is defined as a particle movement error. The overall elastic net movement error e of the elastic net was introduced. The term responsible for rotational forces is modified to order the particles dependent to the error e. The factor f, with $0 \leq f \leq 1$, calculated using equation (64) is dependent on the error e. For high error values, the factor f is near 1, while for low values, f is near 0.

(64) $$\mathrm{f} = 1 - \frac{1}{C_5 \cdot |\mathrm{e}| + 1}$$

The overall formula is shown in (65), which considers the error e and it applies either the shrink forces or the rotational forces. The shrink forces order the particles while the rotational forces move them to form a circular line.

$$\vec{F} = C_1 \cdot \overrightarrow{F_{Equidistance}} + \begin{cases} (C_4 + f) \cdot C_3 \cdot \overrightarrow{F_{Shrink}} + (1 - f) \cdot C_2 \cdot \overrightarrow{F_{Rotation}} & \text{if } |g_n| > t_{a,min} \\ C_3 \cdot \overrightarrow{F_{Shrink}} & \text{if } |g_n| \leq t_{a,min} \end{cases} \tag{65}$$

Results

C_n are parameters used to normalize and measure each force. Throughout the experiments, the following values showed good results (Table 10):

Parameter	C_1	C_2	C_3	C_4	C_5
Value	0.8	4.0	0.1	0.1	200.0

Table 10: Parameter values.

The topology of the map is obtained by another algorithm, such as a Voronoi diagram. An A* algorithm can be used to find a suitable path. Often, the shortest path is chosen. In these examples, a path is found within the topology map, which has to be optimized from a random state of the 'particles'.

In Figure 100, $t_{a,min}$ is set to zero, and the minimal steering angle e is therefore zero. The path is a smooth curve and there is no straight line. In contrast to Figure 100, the parameter $t_{a,min}$ in Figure 101 is set to a value greater than zero. Thus, the path tends to have more straight lines and narrow curves.

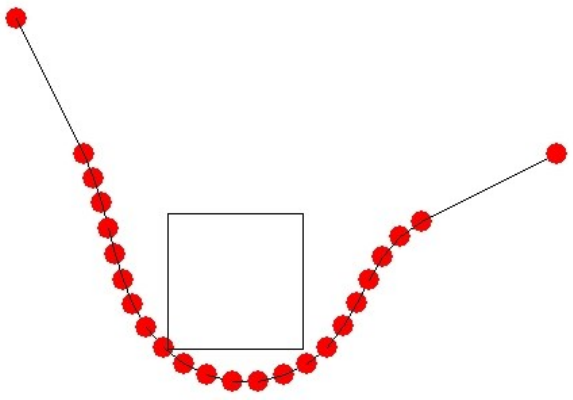


Figure 100: Path with $t_{a,min} = 0$.

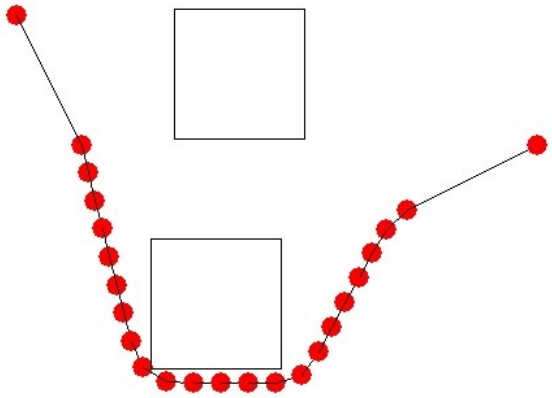


Figure 101: Path with $t_{a,min} > 0$.

8.6 Robot Program Generation

To solve tasks such as handling, welding, gluing or cutting, industrial robots have to be programmed. Robot programs consist of different commands for movement and equipment control, and are usually stored in a robot program file. The development of the robot program may be carried out manually by teach-in or with tool support. The program is written in a specific robot language, such as Melfa Basic IV for Mitsubishi robots. Special knowledge is required for each type of robot, and the development of robot programs itself is complex.

For the enhanced online robot programming system, the robot is controlled along the trajectory described by movement commands. In this study, fixed body dynamics was assumed, and trajectories are therefore independent of the speed of the movement. The trajectory is required to be continuous and smooth to conform to the physical nature of the robot's movement possibilities. As described in Section 8.5.7, a robot trajectory is assembled from path segments with assigned movement types. The standard movement primitives of industrial robots are usually linear, circular and joint movements.

Finding a path is accomplished by the path planning system. It sends a linked list of nodes forming the movement primitives. The nodes store their Cartesian and joint space positions, and they are equidistant to their neighbouring nodes. The robot program generator constructs a trajectory from a list of nodes, and considers node position tolerances that may be delivered by the path planner. It further transforms the trajectory to robot program files in a specific robot programming language or direct movement commands transferred to the robot controller. It was accomplished by the separation of the

trajectory into movement primitives. The movement primitive extraction from a trajectory is implemented as a Matlab script, and it is generated to a shared library. Joint movements are deferred until a later stage because it is assumed that these kinds of movements are somewhat more complicated, but are manageable as an extension to the actual functionality.

Line and curve matching is the foremost challenge of the trajectory generation algorithm. Furthermore, the optimal calculation of junction points between the movement primitives is important for the line and curve-matching algorithm. A junction point connects two movement primitives so that the end node of a geometric figure is the start node of the next figure in a differentiable way.

The concatenation of movement primitives, as explained in Section 8.6.3, is not simple, because all combinations of linear, circular and later also joint movement types are allowed. For example, when two circular movements are concatenated, the connection must be smooth. At the time of computation, not all necessary data may be available. The next movement segment must be analysed, and the resulting information may then be used to construct a smooth connection between the two movement primitives.

The following subsections describe the transformation of the path to a trajectory by the approximation and alignment of the movement primitives. Subsequently, the trajectory was utilized for robot program generation, which is the final artefact of the enhanced online robot programming system. The robot programming language used for the Mitsubishi RV-2AJ robot is Mitsubishi Melfa Basic IV (Mitsubishi-Electric, 2002b). A simple command example may be given by Listing 10, where the MVR command is used for circular movements (where P1 and P2 are the start and end points and M is the midpoint), and MVS is used for linear movements (where P3 is the end point).

```
MVR P1, P2, M
MVS P3
```

Listing 10: Simple movement commands.

Code generation is achieved with the modelling framework introduced in Chapter 9 by using the Java Emitter Template (JET) mechanism (Eclipse Foundation, 2011b). Templates are used to separate dynamic and fixed file contents, for example comments and copyright information.

8.6.1 Calculating Linear Movements

The extraction of linear movement primitives from the node list was implemented by defining a line through the node positions so that the number of nodes touching the line is maximized. The node positions have tolerances which were considered by the introduction of a maximum node-line distance. The maximum distance was applied to the nodes by enlarging each node to a sphere with the radius of the maximum distance. In this way, a line is defined through the node spheres so that the number of node spheres touching the line is maximized. The line origin is always set to the calculated final point of the preceding movement primitive, or, if no such movement primitive exists, to the start node position of the new movement primitive.

In the following steps, the construction of a linear movement line is shown in the case of two dimensions. It was extended to three dimensions in the implemented algorithm. Figure 102 shows three points, which, regarding the tolerance, are lying on a line. P_0 is the start node and is fixed. The circles around P_1 and P_2 display the tolerance sphere, and the red area is a corridor that have to be touched by all nodes. The corridor describes all allowed positions of the line, and it is recalculated for each new sphere. Furthermore, the figure shows the angles which were used to calculate the corridor.

As shown in Figure 103, node P_4 does not touch the corridor and is therefore not a part of the line. The algorithm stops, and the calculation of a new movement type starts from node P_3.

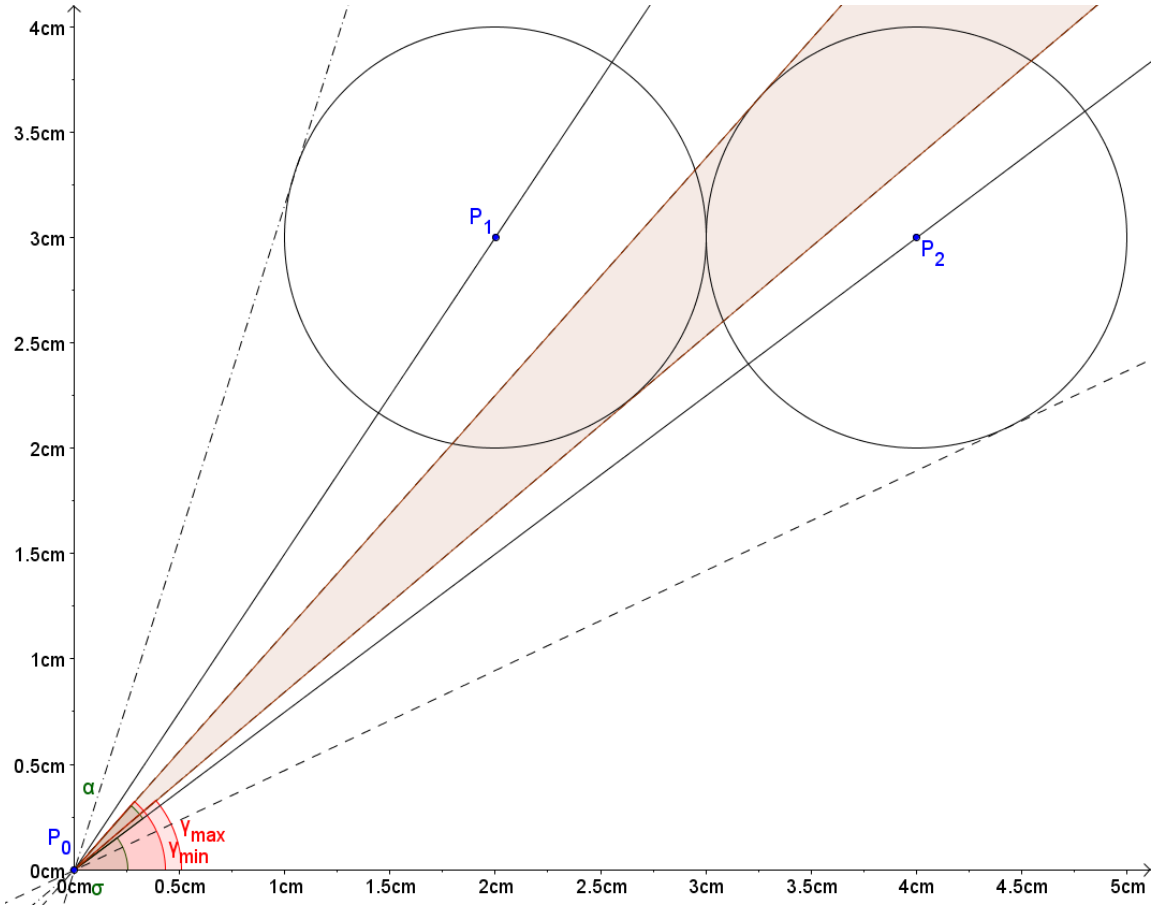


Figure 102: Linear Movement corridor (highlighted in red) calculation with three points $P_{0\text{-}2}$.

A point is calculated in polar coordinates, and may be transformed from Cartesian coordinates. Here, r_n is the radius and P_{n_x}, $P_{n_y}$and $P_{n_z}$are the coordinates of a given point. Furthermore, $\vartheta_n$and φ_n are the two angles required to describe a polar coordinate, and the $atan2$ function is defined in equation (69).

(66) $$r_n = \sqrt{{P_{n_x}}^2 + {P_{n_y}}^2 + {P_{n_z}}^2}$$

(67) $$\vartheta_n = \frac{\pi}{2} - acos\left(\frac{P_{n_z}}{r_n}\right)$$

(68) $$\varphi_n = atan2\left(P_{n_y}, P_{n_x}\right)$$

$$atan2(y,x) = \begin{cases} arctan\left(\frac{y}{x}\right) & x > 0 \\ \pi + arctan\left(\frac{y}{x}\right) & x > 0 \\ -\pi + arctan\left(\frac{y}{x}\right) & y \geq 0, x < 0 \\ \frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ undefined & y = 0, x = 0 \end{cases} \tag{69}$$

The distance of point $\overrightarrow{P_n}$ to a line may be calculated by equation (70).

$$d_n = \frac{\left|\vec{g} \times \left(\overrightarrow{P_n} - \vec{t}\right)\right|}{|\vec{g}|} \tag{70}$$

Because the distance must not be greater than a given d, the maximum allowed distance between a node and the resulting linear movement line is defined in equation (71) with $\vec{t} = (0\ 0\ 0)^T$ as

$$\mathrm{d_n} = \frac{\left|\vec{\mathrm{g}} \times \left(\overrightarrow{\mathrm{P_n}} - \vec{\mathrm{t}}\right)\right|}{|\vec{\mathrm{g}}|} \leq \mathrm{d} \tag{71}$$

Because the line goes through the first node, and the line is only given as a unit vector with $|\vec{g}| = 1$, the declaration in equation (72) was defined as

$$d_n = \left|\vec{g} \times \overrightarrow{P_n}\right| \leq d \tag{72}$$

$$|\vec{g}| \cdot \left|\overrightarrow{P_n}\right| \cdot sin(\alpha_n) \leq d \tag{73}$$

$$\left|\overrightarrow{P_n}\right| \cdot sin(\alpha_n) \leq d \tag{74}$$

$$\beta_n = \alpha_n = asin\left(\frac{d}{\sqrt{{P_{n_x}}^2 + {P_{n_y}}^2 + {P_{n_z}}^2}}\right) \tag{75}$$

The direction is calculated for both angles α_n and β_n in the x-z and the x-y planes.

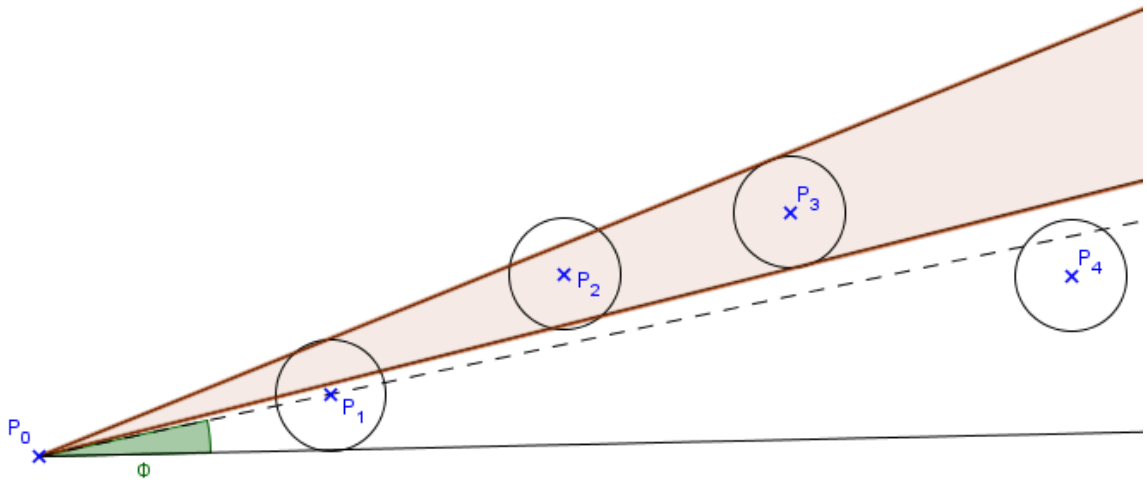


Figure 103: New node not touching the corridor.

Corridor Calculation

The allowed corridor was calculated by the intersection of all sectors, and was formed by the direction angles ϑ and φ of the tangents of each sphere through P_0. $\vartheta_{n_{max}}$ and $\vartheta_{n_{min}}$ are the two upper angles which consider the new corridor. Analogue, $\varphi_{n_{max}}$ and $\varphi_{n_{min}}$ are for the lower angles. ϑ_n and φ_n are the two angles that count for the actual corridor, and α_n and β_n are the angles of the tangents of the actual node.

(76) $$\vartheta_{n_{max}} = \vartheta_n + \alpha_n$$

(77) $$\vartheta_{n_{min}} = \vartheta_n - \alpha_n$$

(78) $$\varphi_{n_{max}} = \varphi_n + \beta_n$$

(79) $$\varphi_{n_{min}} = \varphi_n - \beta_n$$

The allowed corridor is calculated iteratively for each new node as follows, with $\tau_{\vartheta_{max}}$, $\tau_{\vartheta_{min}}$ and $\tau_{\varphi_{max}}$, $\tau_{\varphi_{min}}$ being new possible corridor bounds.

(80) $$\tau_{\vartheta_{max}} = \min\left(\vartheta_{\mathrm{n_{max}}}, \tau_{\vartheta_{max}}\right)$$

(81) $$\tau_{\vartheta_{min}} = \max\left(\vartheta_{\mathrm{n_{min}}}, \tau_{\vartheta_{min}}\right)$$

(82) $$\tau_{\varphi_{max}} = \min\left(\varphi_{\mathrm{n_{max}}}, \tau_{\varphi_{max}}\right)$$

(83) $$\tau_{\varphi_{min}} = \max\left(\varphi_{\mathrm{n_{min}}}, \tau_{\varphi_{min}}\right)$$

Each new node is first checked to be within the corridor. On subsequent checks, the corridor is recalculated using the new node. The corridor size decreases with each iteration. The iteration stops when a new node is not lying within the allowed corridor. In this case, the final node is calculated and a new movement primitive is started from the final point.

Movement Primitive Final Point Calculation

The final direction of the movement line is calculated only at the last node by equations (84) and (85).

(84) $$\vartheta_l = \tau_{\vartheta_{min}} + \frac{\left|\tau_{\vartheta_{max}} - \tau_{\vartheta_{min}}\right|}{2}$$

(85) $$\varphi_l = \tau_{\varphi_{min}} + \frac{\left|\tau_{\varphi_{max}} - \tau_{\varphi_{min}}\right|}{2}$$

The Cartesian coordinate of the endpoint is calculated using the resulting final direction, which is actually the bisecting line of the corridor between ϑ_l and φ_l. The length u of the line is the distance between the start point and the last valid point. Thus, the end-point is calculated by equation (86).

(86) $$\overrightarrow{\mathrm{finalpoint}} = \begin{pmatrix} u \cdot sin(\vartheta_l) \cdot cos(\varphi_l) \\ u \cdot sin(\vartheta_l) \cdot sin(\varphi_l) \\ u \cdot cos(\varphi_l) \end{pmatrix}$$

If the start node is not the origin, the vector to the start node must be regarded. In addition, if there is no third node within the corridor, the final point is set to the second node.

8.6.2 Calculating Circular Movements

This chapter describes the calculation for a circular movement primitive. The nodes of a circular movement are always on a plane. An algorithm was developed which determines the number of nodes located on a common plane, considering the node position tolerances. In the next subsection, all identified nodes are checked to be on a circular line. The final point for circular movement primitives is calculated.

Nodes on a Plane

The framework calculates circular movement primitives that are located on a three-dimensional plane. The nodes have to be located on a plane with a pre-defined tolerance. The tolerance for constructing the plane is given with a tolerance on the normal vectors of each plane, which is constructed with every new node, allowing little rotation when compared to the subsequent normal vectors. The rotation is calculated using the angle between the normal vector and the subsequent normal vector, as explained in the next paragraph.

Three connected non-collinear nodes are required to construct a normal vector. The initial plane was constructed by the first three nodes, including the plane normal. In the following iterations, each normal vector is compared to its successor normal vector. Figure 104 shows four nodes and the three normal planes spanned from the points. The angles between the normal vectors n_1, n_2 and n_3 and their respective unit vectors e_1, e_2 and e_3 are used to calculate the corridor.

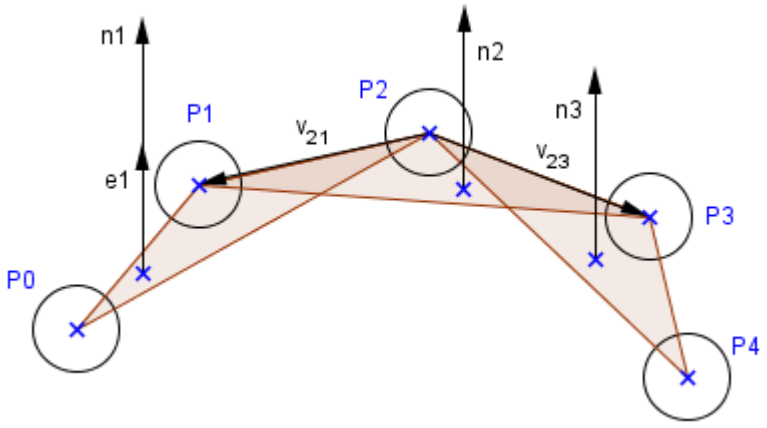


Figure 104: Planes calculated from connected nodes.

The tolerance of the nodes on the plane is added by allowing the unit normal vector e_n to be within a defined angle δ_n. For each new plane, the normal e_n is calculated by equations (87) and (88).

$$n_n = v_{n \to n-1} \times v_{n \to n+1} \tag{87}$$

$$e_n = \frac{n_n}{norm(n_n)} \tag{88}$$

The resulting plane is obtained by the last bisecting vector e_i, where i is the last node of the movement segment. In Figure 105, e_i is the resulting vector of the actual iteration (calculated from e_{i-1} and e_n) and e_{i-1} is the resulting normal vector of the last iteration. Furthermore, e_n is the actual normal vector. Because only the angle between the normal vectors e_i and e_n is relevant, the calculation is also valid in three-dimensional space. The normal of the resulting plane is e_i, and is used for the calculation in the next iteration.

Let δ_{i-1} be the corridor angle, which equals to equation (89), and let α_n be the angle between e_n and e_{i-1}. For each e_n, the allowed corridor is checked, corresponding to the inequality in equation (90).

(89) $$\delta_{i-1} = \left|\delta_{max_{e_{i-1}}}\right| = \left|\delta_{min_{e_{i-1}}}\right|$$

(90) $$\alpha_n \leq \delta_{i-1}$$

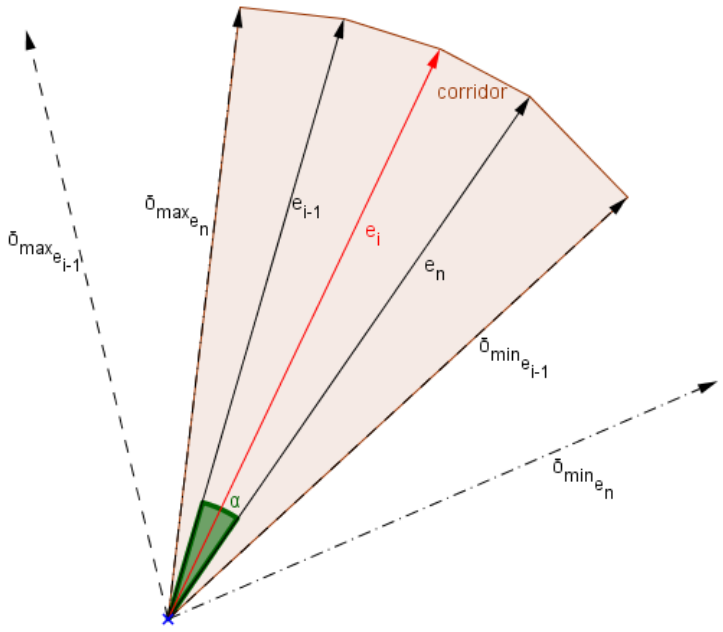


Figure 105: Calculation of the allowed corridor in two dimensions.

If the inequality in equation (90) is true, the bisecting vector between e_n and e_{n-1} is calculated in equation (91).

(91) $$e_i = \frac{e_n - e_{i-1}}{norm(e_n - e_{i-1})}$$

The angle α_n between e_n and e_{i-1} is applied to calculate the new tolerance δ_i with

$$\delta_i = \delta_{i-1} - \frac{\alpha_n}{2} \tag{92}$$

Nodes in a Circular Segment

It is assumed that all nodes building a circular movement section are on a plane, which results in calculations in a two-dimensional space. The first three nodes, including the starting node, perform a curve if the angles between the nodes are within a certain tolerance. Figure 106 demonstrates the situation.

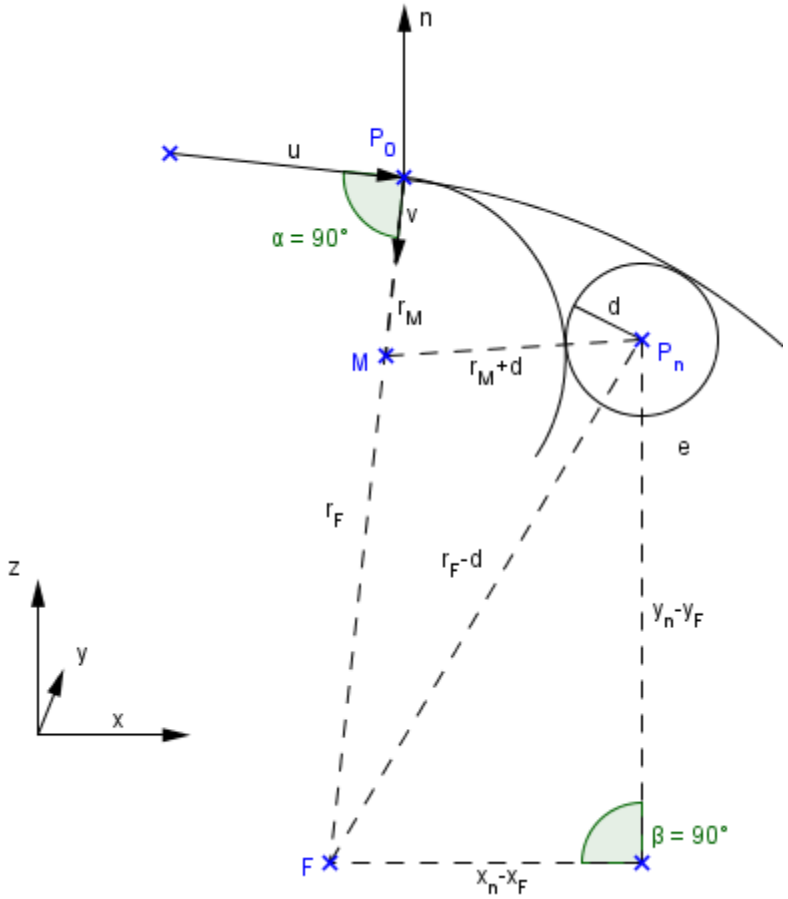


Figure 106: Nodes in a circular segment.

In Figure 106, the nodes $P_0(x_0; x_0)$, $P_n(x_n; x_n)$, the normal vector $\vec{v} = (x_v; y_v)$, and the tolerance d are given. r_M and r_F are calculated in the next paragraph.

Calculation of r_M

To calculate the minimum tolerance, which is the minimum radius r_M in a two-dimensional space, the position of M is given by equation (93).

(93) $$\overrightarrow{M} = \overrightarrow{P_0} + r_M \cdot \vec{v}$$

(94) $$x_M = x_{P_0} + r_M \cdot x_v$$

(95) $$y_M = y_{P_0} + r_M \cdot y_v$$

Referring to Figure 106, equation (96) was applied.

(96) $$(x_n - x_M)^2 + (y_n - y_M)^2 = (r_M - d)^2$$

Equation (97) results in x_M and y_M from applying the equations (94) and (95) to equation (96).

(97) $$r_M^2 \cdot \left(x_v^2 + y_v^2 - 1\right) - r_M \cdot 2 \cdot \left(x_v \cdot (x_n - x_0) + y_v \cdot \left(y_n - y_0\right) + d\right) + (x_n - x_0)^2 + \left(y_n - y_0\right)^2 - d^2 = 0$$

Considering that $|\vec{v}| = 1$ the following assumption was made:

(98) $$x_v^2 + y_v^2 = 1$$

(99) $$r_M^2 \cdot \left(x_v^2 + y_v^2 - 1\right) = 0$$

(100) $$r_M = \frac{(x_n - x_0)^2 + (y_n - y_0)^2 - d^2}{2 \cdot (x_v \cdot (x_n - x_0) + y_v \cdot (y_n - y_0) + d)}$$

Calculation of $\mathbf{r_F}$

To calculate the maximum tolerance, which is the maximum radius r_F in two-dimensional space, the position of F is given by equation (101).

(101) $$\vec{F} = \overrightarrow{P_0} + r_F \cdot \vec{v}$$

(102) $$x_F = x_0 + r_F \cdot x_v$$

(103) $$y_F = y_0 + r_F \cdot y_v$$

With respect to Figure 106, the equation (104) was applied.

(104) $$(x_n - x_F)^2 + (y_n - y_F)^2 = (r_F - d)^2$$

Equation (105) results in x_F and y_F from applying equations (102) and (103) to equation (104).

$$r_F^2 \cdot (x_v^2 + y_v^2 - 1) - r_F \cdot 2 \cdot (x_v \cdot (x_n - x_0) + y_v \cdot (y_n - y_0) - d) + (x_n - x_0)^2 + (y_n - y_0)^2 - d^2 = 0 \quad (105)$$

Considering that $|\vec{v}| = 1$, the following assumption was made:

$$x_v^2 + y_v^2 = 1 \quad (106)$$

$$r_M^2 \cdot (x_v^2 + y_v^2 - 1) = 0 \quad (107)$$

$$r_F = \frac{(x_n - x_0)^2 + (y_n - y_0)^2 - d^2}{2 \cdot (x_v \cdot (x_n - x_0) + y_v \cdot (y_n - y_0) - d)} \quad (108)$$

Local Coordinate System Calculation

All circular movement calculations were accomplished in the two-dimensional space. The plane described by e_i is three-dimensional in world space, and a local coordinate system was calculated by an arbitrary coordinate system with e_i being the z-axis. The x- and y-axis were randomly generated. Subsequently, the transformation matrix from the world to the local coordinate system was computed so that all circular movement calculations could be calculated within the local coordinate system.

Circular Movement Corridor Calculation

Corresponding to Figure 107, $r_{max} = r_F$ and $r_{min} = r_M$ are used to calculate the resulting corridor for the allowed radius r in equations (109) and (110), where r_{max-1} and r_{min-1} are the calculated radii from the last iteration.

$$r_{allowed_{max}} = \min(r_{max}; r_{max-1}) \quad (109)$$

$$r_{allowed_{min}} = \max(r_{min}; r_{min-1}) \quad (110)$$

When the calculated radius is not within the corridor, and thus does not satisfy the inequalities stated in equations (111) and (112), a new movement segment is started.

$$r_{allowed_{min}} \leq r_{max} \quad (111)$$

(112) $r_{allowed_{max}} \geq r_{min}$

Movement Primitive Final Point Calculation

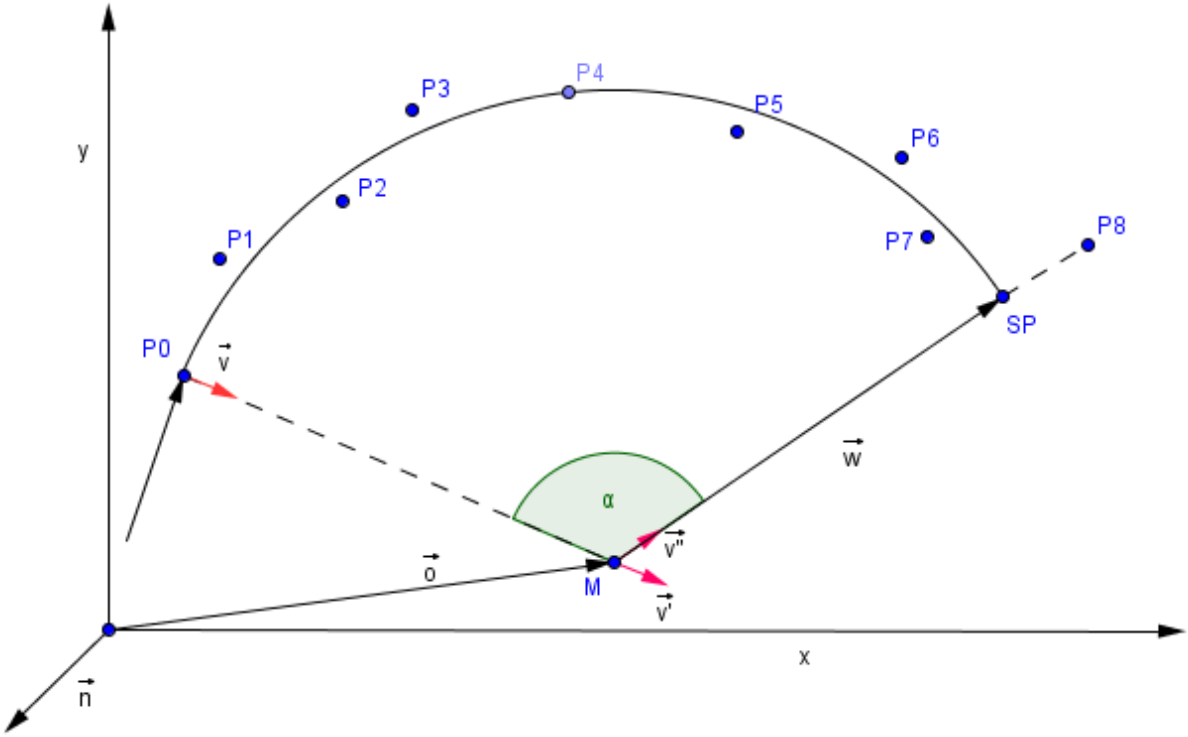


Figure 107: Final point calculation.

Figure 6 shows the final point calculation for a circular movement section. For the vector $\overrightarrow{M\,P_8}$, the vector $\vec{w}$ is calculated using equation (113) with r being given by the radius calculation previously presented.

(113) $$\vec{w} = \frac{\overrightarrow{M\,P_8}}{\text{norm}\left(\overrightarrow{M\,P_8}\right)} * \text{r}$$

For a given normalized $\vec{v}$ and α, $\overrightarrow{v''}$, which is the rotation around the $\vec{n}$- axis with an approximated angle of $180° - \alpha$, was first calculated. The approximation was established because P_8 may eventually be off the circle. The final point was therefore calculated by the equations in (114).

$$\overrightarrow{P_{final}} = \vec{o} + \vec{w}$$

(114) $$\vec{v}\ with\ \overrightarrow{|v|} = 1$$

$$\alpha := angle\ between\ P_0\ and\ P_8$$

For a given normalized $\overrightarrow{v}$, the vector $\overrightarrow{w}$ and radius r were calculated in equations (115) and (116).

(115) $$\vec{w} = r * \overrightarrow{v''}$$

(116) $$r = \|\vec{w}\| = dist(P_0; M)$$

8.6.3 Connecting Movement Primitives

At the beginning of a new movement, the movement type is unknown, and calculations for all movement primitives are therefore started until the movement type is identified. The identification method used considers the movement type that covers the most nodes. However, care is taken to allocate the nodes to the right movement primitive on transition points of two movement primitives, which may be a combination of linear and circular movement primitives.

Figure 108 illustrates the transition of two linear movement primitives. To ensure that the path is continuous and smooth, two linear movement primitives were connected using a circular movement primitive. Calculations were omitted because the Mitsubishi robot system has an option for smooth robot position transitions and ensures a continuous path.

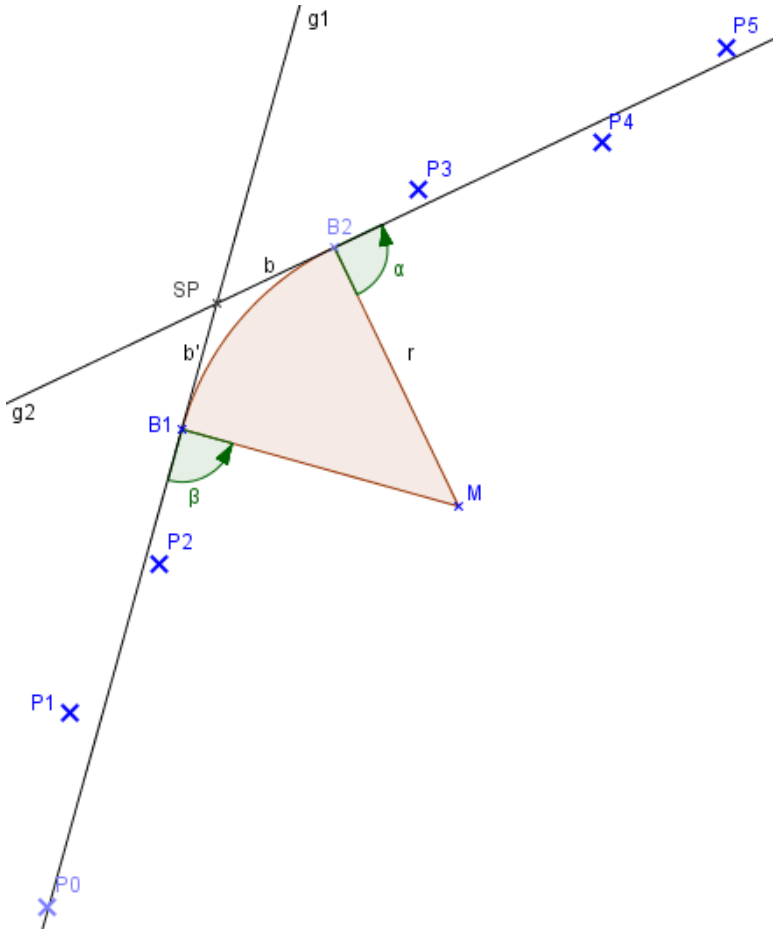


Figure 108: Connecting two linear movement primitives.

The remaining combinations are illustrated in Figure 109, Figure 110 and Figure 111.

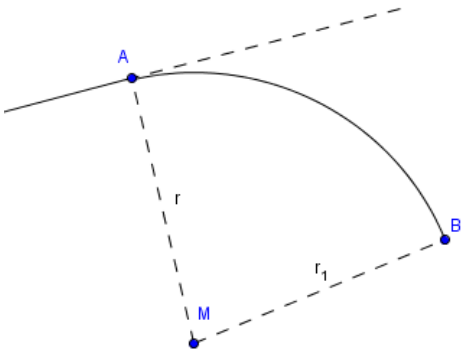


Figure 109: Connecting a linear and a circular Movement.

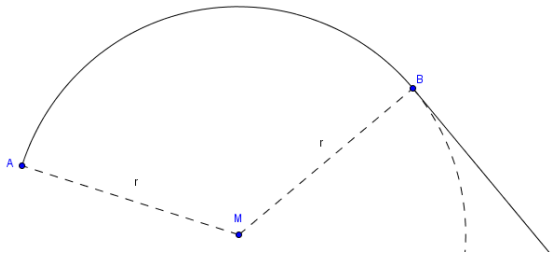


Figure 110: Connecting a circular and a linear movement.

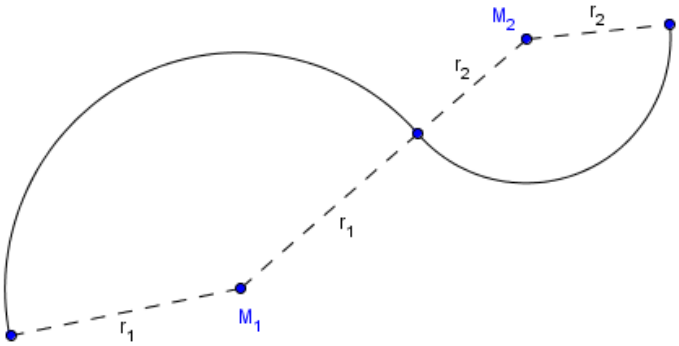


Figure 111: Connecting two circular movements.

8.7 Summary

The development of the enhanced online robot programming system driven by the requirements presented in Chapter 5 began with an evaluation of usage scenarios. The scenarios demonstrated that it should be easy to utilize the system, and it has to be removed after generating the robot program. The aim of the system was to generate a static robot program that is comparable to manually programmed robot programs.

The ease of utilization of the system was guaranteed by an expert system that supports the operator during robot programming. In practice, the ability to control the robot manually has become very important to the execution of manual exploration, in order to define the mission and to place virtual objects in the working cell. The expert system itself is efficiently employed only when the mission is at least defined, which can either be done online or offline using a simulation system or with known locations.

The presented online robot programming approach is different when compared to existing approaches. First, the human operator reports collisions, and it is therefore generally available and cost efficient. Second, the applied trajectory-planning algorithm is able to handle efficiently the provided information, and it intelligently controls the robot within the robot cell to compute the robot trajectory with the help of the interconnected mission and trajectory planner in collaboration with the generation of the robot program.

The mission planner plans the mission and controls the path planner, which provides trajectory length information to the mission planner. The mission planner in turn re-plans the mission with an applied hysteresis. Planning a trajectory online always utilizes the real production system, which is executed throughout the robot-program generation process. Thus, the hysteresis on the trajectory length was applied to prevent the production system from executing re-planning too often.

The trajectory planner connects two given locations, and reports the trajectory length to the mission planner. The in-memory world model plays an important role because it also provides the roadmap that was, in the first instance, planned to store just geometric data for the trajectory planning neural network.

The trajectory planning neural network approximated the obstacles and created a roadmap within the free space. Obstacle approximation was optimized by object simplification, surface reconstruction and progressive mesh algorithms. The road map generation is easy to calculate, the possibility of parallel computing is presented, and higher dimension calculations are possible. The roads are simplified to a topological map and forces are applied to straighten and shrink the roads.

Calculation of the node movement was computationally intensive because many nodes have had to be considered for each calculation iteration. In addition, the calculation of collisions also produced a high processor load. The calculations of node movements depended on random inputs, which are hard to generate for online systems. Random inputs lead to a slow convergence of the neural network, even in a simulation environment. The proposed Coloured Kohonen map rarely formed a Voronoi diagram, and required further improvements.

Nevertheless, the basic principle was transferred to a cell-based approach, which stored joint locations within the roadmap cells. In fact, the combination of the cell-based roadmap with joint locations together with the Elastic Net trajectory generation approach realized the proposed enhanced online robot programing system. The generation to a robot program was accomplished by analysing the created trajectory.

The new motion-planning algorithm plans with only local knowledge smooth trajectories that consist of linear and circular movement primitives and generates a static robot program. The system considers objects with predictable movements. Timely synchronized objects that support the generation of events were also supported within the generated (static) robot program.

9 Research of a Software Development Framework for Complex Systems

The motivation for the model driven code generation framework is based on the requirement to rapidly connect distributed software components which are written in different programming languages. They also required to run on different platforms, sensors and third party tools such as Matlab (TheMathworks, 2011) across a network without the need for the time-consuming development of data communication and tool connection infrastructure.

This was accomplished using model-based software development including code generation, which entails the composition of applications from pre-designed hull software components enriched with the business logic of the application. The details regarding the implementation of the components are hidden behind well-defined interfaces. Thus, much improved software quality becomes realistic. Moreover, previous experiences with component-based software development in other application domains have resulted in drastically improved software development productivity, which is sometimes more than one order of magnitude greater than conventional software development (Sutherland, 1998, Zincke, 1997).

Matlab/Simulink is often adopted as a development environment because of its fast modelling and code generation capabilities as well as its valuable library functions. Connecting such a tool to a distributed software system supports the developer during software development by enabling communication with existing components.

The run-time architecture consists of interconnected components, communicating through message passing, which is executed by a communication middleware. Each component is typically a process running on a node such as a computer or an embedded device. An evaluation of existing communication middlewares was carried out in Section 9.2.

A model-driven approach was chosen in order to increase the usability of the framework with a domain specific modelling language which was derived from the Real-Time Object-Oriented Modelling (ROOM) language (Selic, 1996a, Selic, 1996b, Selic *et al.*, 1994). This language also defines the run-time behaviour of the generated software components.

The commercial tool Rational Rose Real-Time from IBM (IBM Corp., 2011), formerly known as ObjecTime, was a toolset supporting the ROOM language. Unfortunately, this toolset is no longer available, and therefore makes it necessary to re-implement the code-

execution-model, which is described in Appendix E. The eclipse project eTrice (eTrice Group, 2011) was recently shifted from the proposal phase to the incubation phase, and aims to implement the ROOM language together with code generators and tooling for model editing.

A major goal of the proposed framework is to enable sensor-based robot control applications to be built from libraries of reusable software components. For this purpose, the framework provides standard interface specifications for implementing reusable components. A well-written and debugged library of software components facilitates the rapid development of reliable sensor-based control systems.

Existing robot control frameworks introduce re-configurable software components as well as special communication and code execution models (Griph *et al.*, 2004, Lee and Yangsheng, 1998, Wason and Wen, 2011). These approaches attempt to enhance the configuration of the components for re-use and the running system itself. However, this chapter also proposes to enhance the usability through graphical modelling and code generation.

9.1 System Modelling

ROOM defines a visual modelling language with formal semantics and a code execution model, which is a set of rules defining the system behaviour (Selic, 1996a, Selic, 1996b, Selic *et al.*, 1994). The visual modelling language is optimized for specifying, visualizing, documenting and automating the construction of complex, event-driven, and potentially distributed real-time systems. By connecting several components, an interaction flow via messages may be established between them.

In the proposed framework, a component can be developed in Java, C#, C++ and C, and can be deployed on different processing units. The processing unit may be a general-purpose processor, digital signal processor or a field-programmable gate array (FPGA), where each processing unit may have its special system architecture that influences, for example, the handling of threads.

In addition, a component may also be a complete development environment, which allows direct communication with existing components during development. The integration of tools is explained in Section 9.5.

The component behaviour is described as a hierarchical state machine which provides a number of powerful features, including group transitions, transitions to history, state variables, initial points, and synchronous message communications.

The developer writes user programs for state transitions where the component has to perform an action. Additionally, each state may have an entry and an exit function, which are executed when the component enters or exits the state, respectively. Advantages are that components may be distributed on different nodes with ease and better encapsulation is reached, because only the component interfaces, and not the type of the component, are required in order to interact with it.

ROOM also defines a message service that controls the logical message flow within a physical thread, while a middleware, which is further described in Section 9.2, is responsible for transmitting the messages. The implemented message service is optimized for speed in the local delivery of messages through the utilization of operating-system specific communication mechanisms. It should be sufficiently abstract to be used by any operating system, and should be concrete enough to fulfil requirements of speed, code size and memory consumption. The implemented message service is included together with the code execution model in a runtime library. An instantiated message service is identified by the network port number and the internet protocol (IP) address of the host.

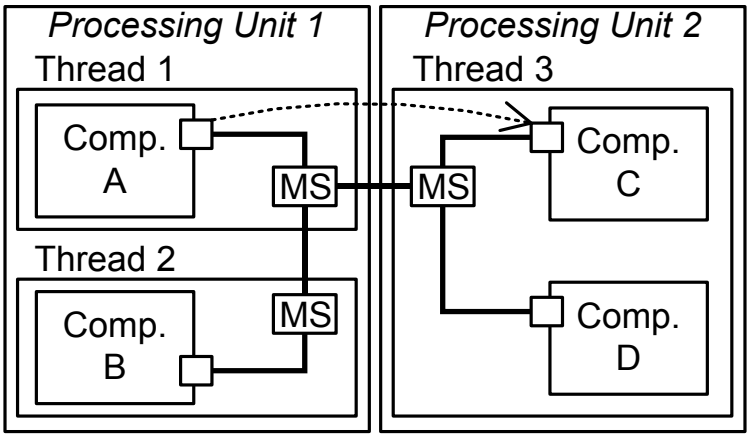


Figure 112: Communication overview: Message passing
from component A to component C (dashed arrow).

The ROOM communication system illustrated in Figure 112 consists of processing units, threads, components and message services (MS) along with its connectivity. The ports of each component may communicate with other components via connections to the

message service, which handles local and remote message passing. A message from the port of component A to the port of component C (see dashed arrow) may be passed through both message services until it gets to the target port. In this example, messages from component B may only be sent to component A.

9.2 Communication Middleware

Currently available communication mechanisms may generally be separated into three categories: transport level, message passing and remote procedure calls.

The transport level is simply a pipe to send data streams or packets without any formatting specification, such as serial ports or TCP/IP. Direct socket communication requires the development of a proprietary protocol and exception handling, which involves significant effort. Furthermore, marshalling and de-marshalling have to be implemented, and this is particularly complex because of the requested compatibility between the different programming languages. For example, if it is required that a C++ object be transformed into a Java object.

Message passing adds structure to the packets to define the content, but it still requires the user software to build and send the messages. ZeroC Ice (ZeroC Inc., 2011) and CORBA are middleware systems that build an abstract communication layer.

Remote-procedure-calls attempt to expose functions or full objects across a process or network boundary without the user software being aware of the boundary. Remote method invocation may be given as an example.

A comparison of the different communication middlewares supports the choice of the ZeroC Ice middleware. Its implementation is available on various platforms, including embedded systems, and for different programming languages such as Java, C++ and C#. While CORBA may be an alternative, it appears to be complex and does not have the ability to transmit objects, therefore allowing only primitive data types, while ZeroC Ice may handle object transmission. In addition, ZeroC provides Eclipse support, which simplifies the usage of ZeroC Slice, which is the interface definition language.

9.3 The Toolchain

A general overview of the workflow is given in Figure 113. The toolchain creates and synchronizes source code from a given graphical model which includes the modelled

behaviour of each component. The visual modelling language ROOM is represented as graphical elements in the commercial off-the-shelf editor Enterprise Architect from SparxSystems (Sparx Systems, 2011). This graphical model is utilized to create source code with the help of the eclipse modelling framework (EMF) and its code generation capabilities (Eclipse Foundation, 2011a). The runtime library provides a communication layer, the implementation of the code execution model and the message service.

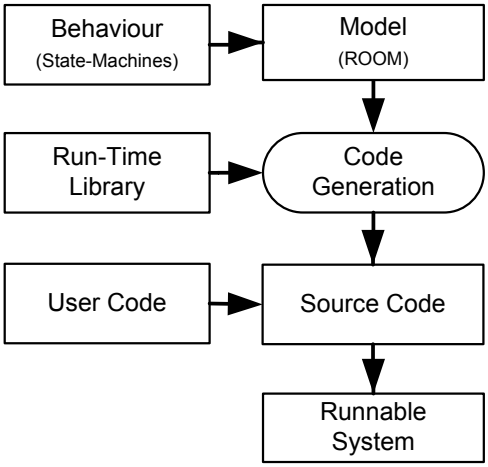


Figure 113: Code generation workflow.

The generated source code can be synchronized with the written source code of the user to simultaneously allow modelling and code implementation. Finally, the source code can be compiled to a runnable application for the target system, e.g. a personal computer with a Windows operating system or an embedded system with a PowerPC operating system.

9.4 Toolchain Implementation

A more detailed description of the toolchain is given in Figure 114. The graphical notation elements of ROOM were integrated into Enterprise Architect (Sparx Systems, 2011) with the help of an Enterprise Architect specific model driven generation (MDG) technology file. These modelling elements are utilized to create visual models of executable software systems.

A C# to Java application communication channel was implemented with a direct socket connection to the Java model repository application. It is utilized to store the visual model into the model repository, which was defined with the eclipse ecore editor.

The template-based code generator application based on JET (Eclipse Foundation, 2011c) transforms the model to Java source code.

The Code Merger tool utilizes JMerge (Eclipse Foundation, 2011a), and runs as a headless eclipse application, which starts a minimal eclipse framework in the background. It merges the generated source code with the existing one.

The toolchain supports the automatic generation of eclipse Java projects for each component and the runnable system. These projects may be imported into the eclipse workspace. All link dependencies including the link to the run-time library were automatically set, and a UniMod state machine (eVelopers Corporation, 2011) was generated using each component project to define the behaviour of the component.

The runtime library was implemented in a platform-dependent manner, and includes the ROOM code execution model and the middleware from ZeroC Ice (ZeroC Inc., 2011).

The middleware supports a target abstraction layer, which simplifies the creation of the platform specific library. This framework also enables the use of specialized tools such as Matlab/Simulink, as further described in Section 9.5.

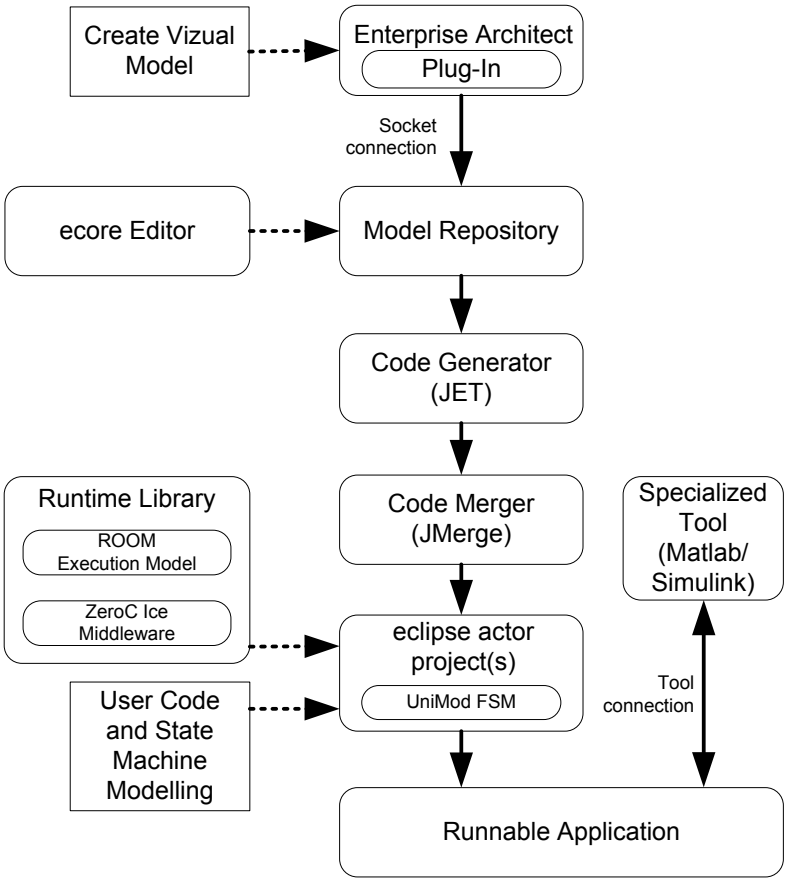


Figure 114: Toolchain implementation.

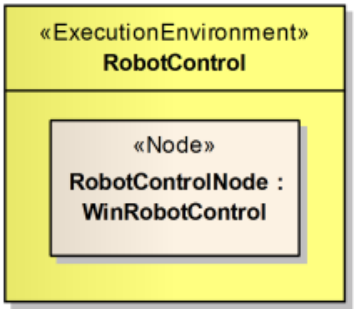


Figure 115: Execution environment.

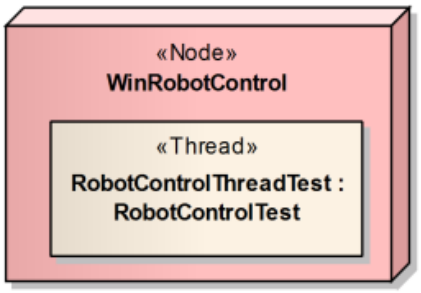


Figure 116: Node.

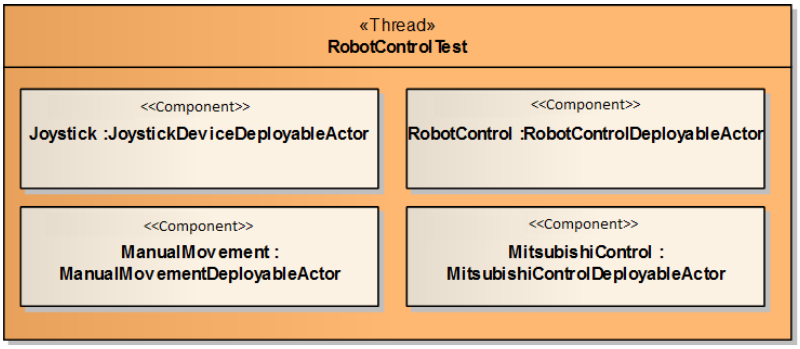


Figure 117: Actor deployment.

9.5 Connecting Specialized Tools

Specialized tools usually have enhanced functionality, which enables them to solve domain-specific development tasks. They may also have been established as common tools within these domains. The integration of such tools into the communication framework adds communication capabilities to other components, e.g. for sensor or control functionality, during development. The development may be finalized by generating a DLL or an executable, which may successively be used within the communication network. DLLs can always be utilized with the help of visual modelling elements that support such libraries and generate the necessary code to incorporate the libraries. The dyncall library (Adler and Philipp, 2011) was employed within the run-time library for this purpose.

A direct integration of specialized development tools was reached through tool specific integration technologies. For example, Matlab may be connected through the Microsoft Component Object Model (COM) or Dynamic Data Exchange (DDE) technology for message passing, which is described by Kohrt *et al.* (2006a). The middleware can also be directly utilized with an S-function to establish communication with the distributed components.

A ‘Plugin Manager’ software component was developed to utilize shared libraries with Java in a generic manner. The component allows the generic use of shared libraries, DLLs on Windows machines and libraries on Linux machines. The component encapsulates the Java/DLL intercommunication as well as the usable functions of the libraries. A function call is initiated by a synchronous message. The message contains all of the data that is necessary to call the library function, e.g. function name and parameters. The call-back functionality allows the native libraries to call Java functions. The calling sequence is illustrated in Figure 118. The Plugin Manager is further described in Appendix H.

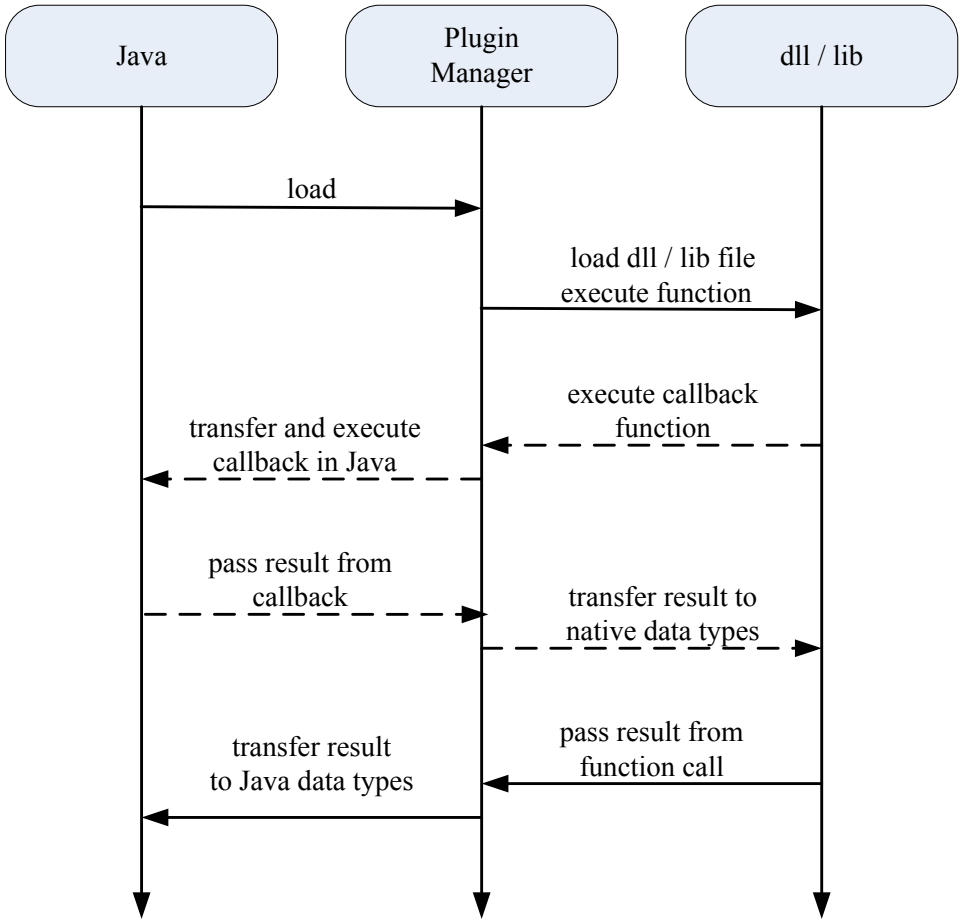


Figure 118: Plugin manager communication.

9.6 Code Generation Example

As depicted in Figure 7, a robot control application with a joystick for the articulated Mitsubishi RV-2AJ robot demonstrates modelling and code generation. Applications are defined by the instantiation of an ‘Execution Environment’, which is named ‘Robot Control’ in Figure 115. Although a single ‘Win Robot Control’ node is deployed to the execution environment for the entire application, several additional nodes may be

deployed. Physical threads are modelled to allow thread deployment. Components are finally deployed to those threads (Figure 117), while their connectivity is modelled in a thread-independent manner, as illustrated in Figure 119. The interface definition of the 'Manual Movement Deployable Component' in Figure 120 describes the provided and required interfaces, which are fixed to component ports. The 'Control Port' provides component life-cycle interfaces such as 'Control In' in Figure 121 to start, stop, initialize, release and locate the component. Additional component property management is implemented with the set and update property signals. Synchronous and asynchronous message passing is supported. Each interface defines allowed signals that have to be modelled in the UniMod finite state machine, as depicted in Figure 122. A message is received via the port interfaces through the port to the state machine of the component, which fires a transition.

The executed transition method contains the user code. The generation process generates methods such as the initialization methods shown in Listing 11, which was derived from the 'Init' transition. JMerge uses code tags such as '@generated not', or is overwritten by the code generation process.

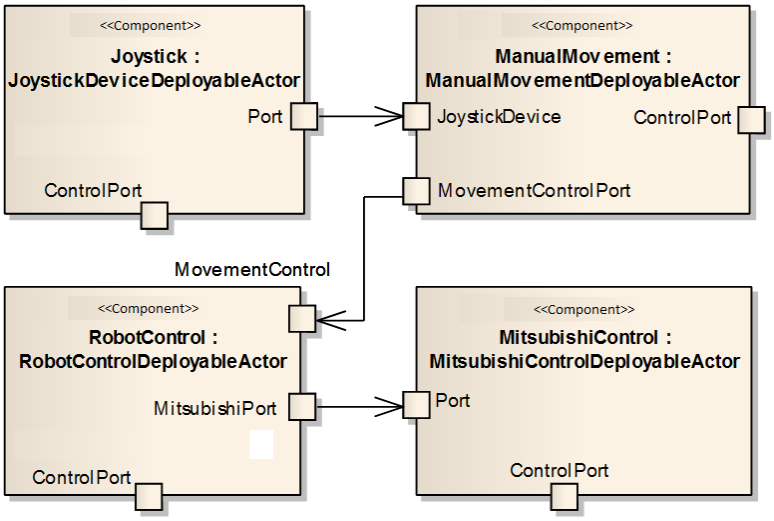


Figure 119: Component connections.

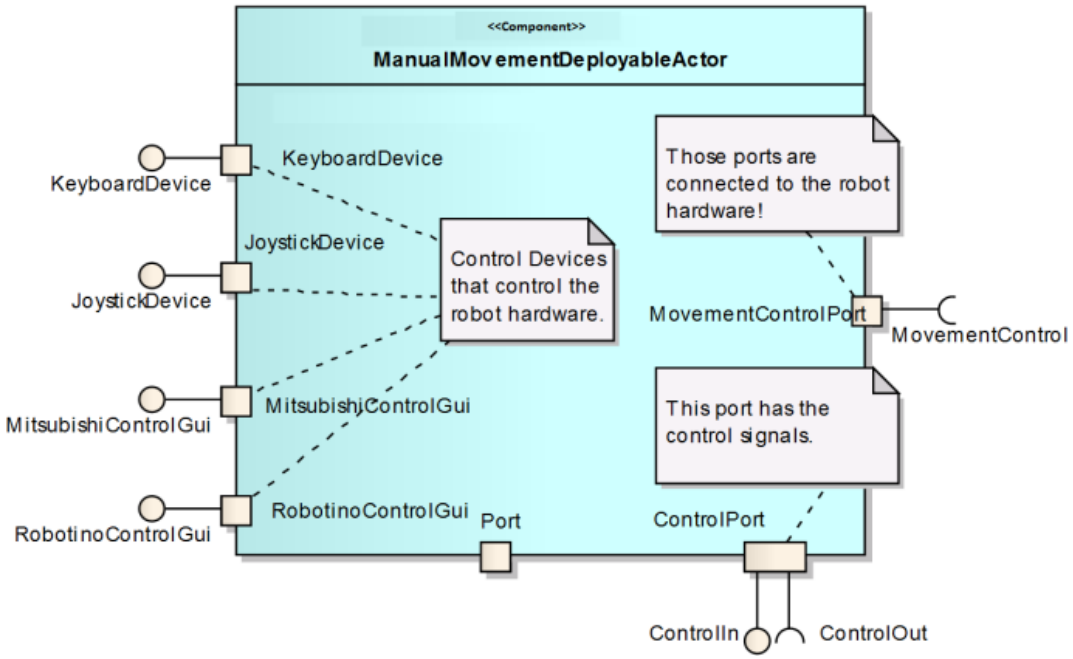


Figure 120: Component interfaces.

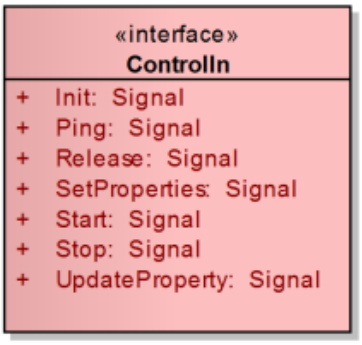


Figure 121: Interface definition.

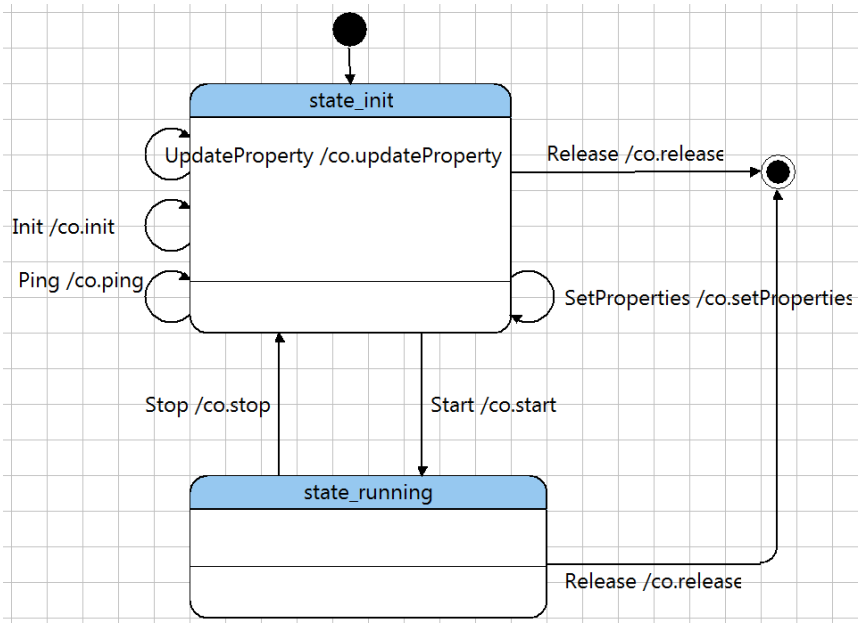


Figure 122: UniMod state machine diagram example.

```
/**
* Init the component.
* @generated
*/
protected void init()
{…}
```

Listing 11: Generated Java code.

Other tags such as '@unmodifiable' may also be used to control the merge functionality.

9.7 Summary

This chapter highlights important aspects in the development of the proposed model driven toolchain. The various model-to-model transformation stages and tools are presented from graphical ROOM models to the runnable application. The toolchain may generally be used for software development, and for specific problem domains such as robotics. The extensibility of the domain specific language allows domain-oriented engineering. The level of abstraction is a significant issue for the handling of large software systems. The abstraction level is raised by using a model driven toolchain. Standard designs and concepts may be easily integrated and used by the developers who need only the graphical front end to such extensions. Encapsulation results in the reuse of the so-called black box, which is a favourable form of it, since the economics of scale allow more focus to be made on software design, software reviews and software testing.

The integration of specialized tools and development environments enhanced the development process.

The proposed model-based code generation framework has a significant productivity benefit, although implementation of the toolchain requires that significant investments be made. However, once a toolchain is developed, it can easily be applied.

ROOM is a message-based system that is based on state machines, and it requires training for inexperienced developers. The message service is an additional layer that interprets and transfers messages to the target component port, which may lead to a delay in the message delivery. The delay must be considered, especially for time-critical systems. Therefore, it plays a key role in the performance of the system. Nevertheless, such a toolchain can be valuable for large software development projects, and allows a strict encapsulation into components with clearly defined interfaces. The intention is to continue with this approach, and to further enhance the modelling and code generation features, especially for debugging purposes and the implementation of a state machine (with a graphical editor) alternative to the slow UniMod state machine. The Simulink Stateflow state machines may also be used in the Simulink context, but it requires adaptation to be made usable in non-Simulink contexts.

The main advantages of model-driven development include better maintainability, a uniform programming model, reusable model parts, simple but efficient communication, higher abstraction, code generation, system-wide optimization possibilities and focused development in relation to the business logic.

10 System Implementation

The online path planning and programming support system is an approach that can reduce the robot programming time, including preparation and installation. It generates acceptable robot programs and considers the modern industrial basic goals of flexibility, speed and optimization, which are mentioned in Chapter 8. It finds a trade-off between shortest-path finding and trajectory forming and maintainability. Finally, it generates a downloadable robot program file.

In this section, the general execution of the programming assistant is described, and a scenario (see Figure 126 and Figure 127) was chosen to demonstrate the proposed approach. The system is executed with a real five-axis industrial scale, articulated Mitsubishi RV-2AJ robot (Kohrt *et al.*, 2008). The algorithm utilizes an octree as the world model (as described in Chapter 6) and joint positions attached to the octree cells. During implementation, the algorithm was tested in simulated two-dimensional space using a quadtree as world model and world positions attached to the quadtree cells. The proposed algorithm works in real surroundings. The illustrations shown in this section are simplified to support the understanding of the algorithm.

In the chosen real scenario, the two obstacles O_1 and O_2 are provided as CAD objects, and they have been imported into the in-memory environment model. The obstacle O_3 should be unknown to the system, and was therefore not imported. The chosen scenario consists of a mission with the start and target positions P_1 and P_{10}.

First, the general workflow of online robot programming is described in Section 10.1, followed by the data import in Section 10.2 and mission preparation in Section 10.3. Subsequently, the roadmap was generated within the world model in Section 10.4, and is utilized as a corridor for the configuration space positions of the robot. Shortest-path planning is applied for those positions in Section 10.5, which may lead to a path from the start to the target, which is transformed to a trajectory. Section 10.7 illustrates the path planning behaviour with an additional obstacle, which leads to the re-planning of the path. Finally, the robot program is generated in Section 10.8 and the robot programming durations are compared to manual programming in Section 10.9.

10.1 General Workflow

In general, the operator executes the robot-program generation system after it is set up. Collisions are detected with connected sensors such as the cameras or the collision indication buttons. The operator indicates collisions with static or dynamic obstacles. The support system automatically controls the movements of the robot until a suitable robot trajectory is found, if one exists. High accuracy is not needed, since the used trajectory generation algorithms and strategies may handle low accuracies. One strategy is the adoption of the Voronoi features, which maximize the clearance to obstacles.

The system first tries to explore the working space to build the in-memory topology. Subsequently, a robot path to the target position is computed. Target positions are either application locations or are part of an application path, which may be a part of a mission. A mission may have multiple application paths and locations, which results in the well-known travelling-salesman-problem. The planning problem is solved in order to minimize the overall path length. This also includes the path-planning algorithm.

The movement of the robot is slow enough to allow the operator to detect collisions. The environment is stored within a world model, which is capable of storing collision positions. It creates a roadmap in the Voronoi form, and supports path searching and trajectory generation by combining the world and joint spaces. The robot movement benefits from the roadmap generation by maximizing the clearance to the obstacles using collision detection. A hysteresis that reduces re-planning is applied to reduce real robot movements. In addition, this hysteresis also increases the knowledge of the environment by adding sensor data to the world model.

10.2 Pre-Existing Data Import

In the chosen scenario, the two obstacles, O_1 and O_2, are given as drawing exchange format files, and are imported either with the robot or with the pointing device by placing virtual objects or by absolute data of the DXF file to the environment model. One obstacle, O_3, is ‘unknown’ to the system (not imported).

10.3 Mission Preparation

The chosen scenario consists of a mission with positions P_1 and P_{10} and the application path P_7 to P_8, which is a straight line with the hand tool equipment of the robot closed. The pointing device was used to store the locations of the application paths together with the support system.

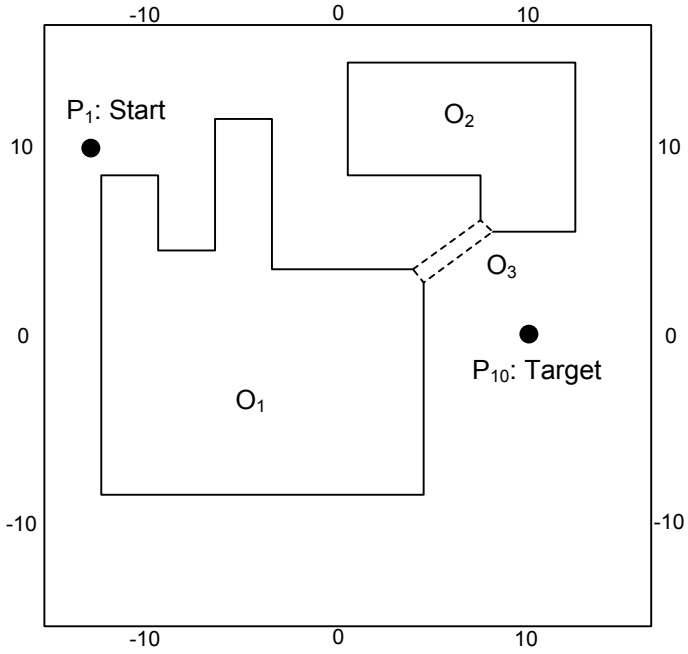


Figure 123: Experimental scenario (2D example in 3D world).

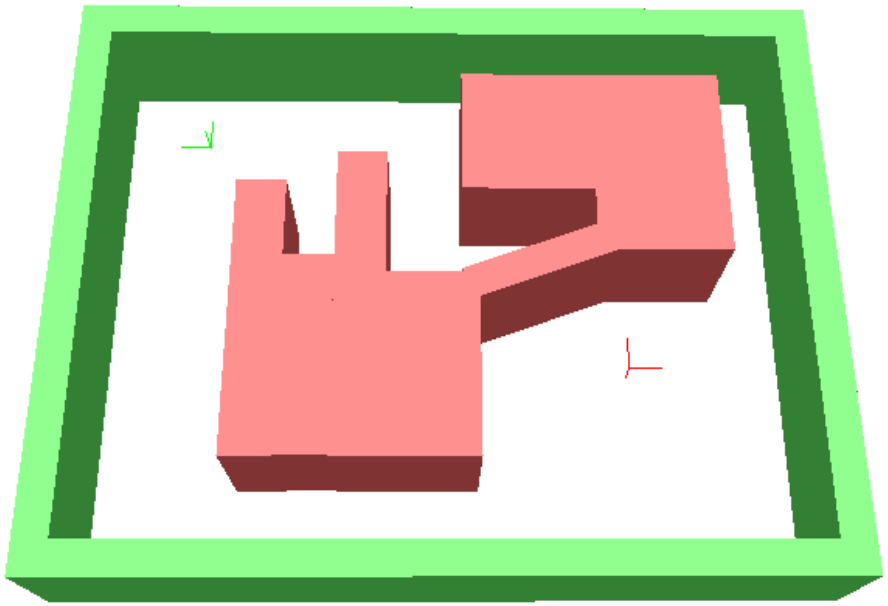


Figure 124: Screenshot of the experimental scenario.

10.4 Roadmap Generation

The scenario in Figure 123 is processed using the roadmap shown in Figure 125. Each cell of the roadmap contains the produced robot positions in the configuration space, as explained in Subsection 8.5.3.

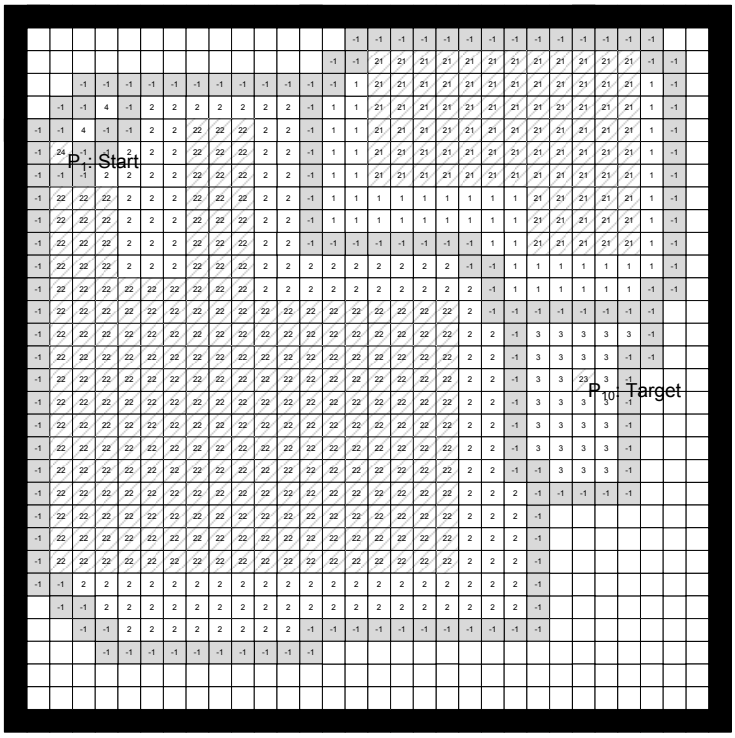


Figure 125: Roadmap of the scenario (without obstacle O_3).

10.5 Path-Planning Application

Using the roadmap generated in Section 10.4, the resulting corridor is illustrated in Figure 126, including the indicated configuration space positions. The search is executed on these positions, and it finds a path, as illustrated in the figure. Configuration space positions are also added to the start and target positions including their extended cells, as explained in Subsection 8.5.6.

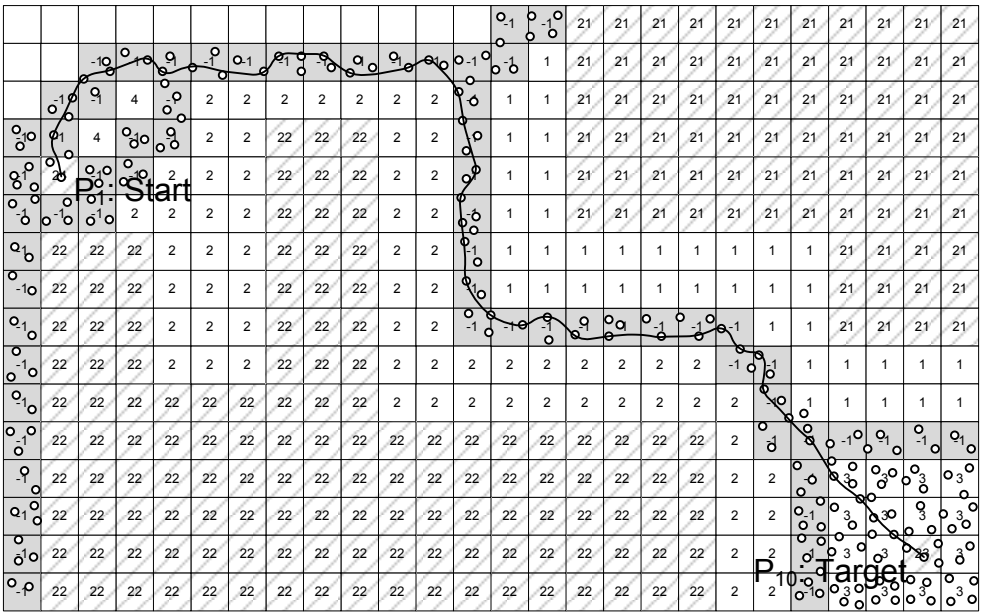


Figure 126: Roadmap corridor including configuration space positions.

10.6 Elastic Net Trajectory Generation

As shown in Figure 128, the found path is processed and adapted to a feasible trajectory.

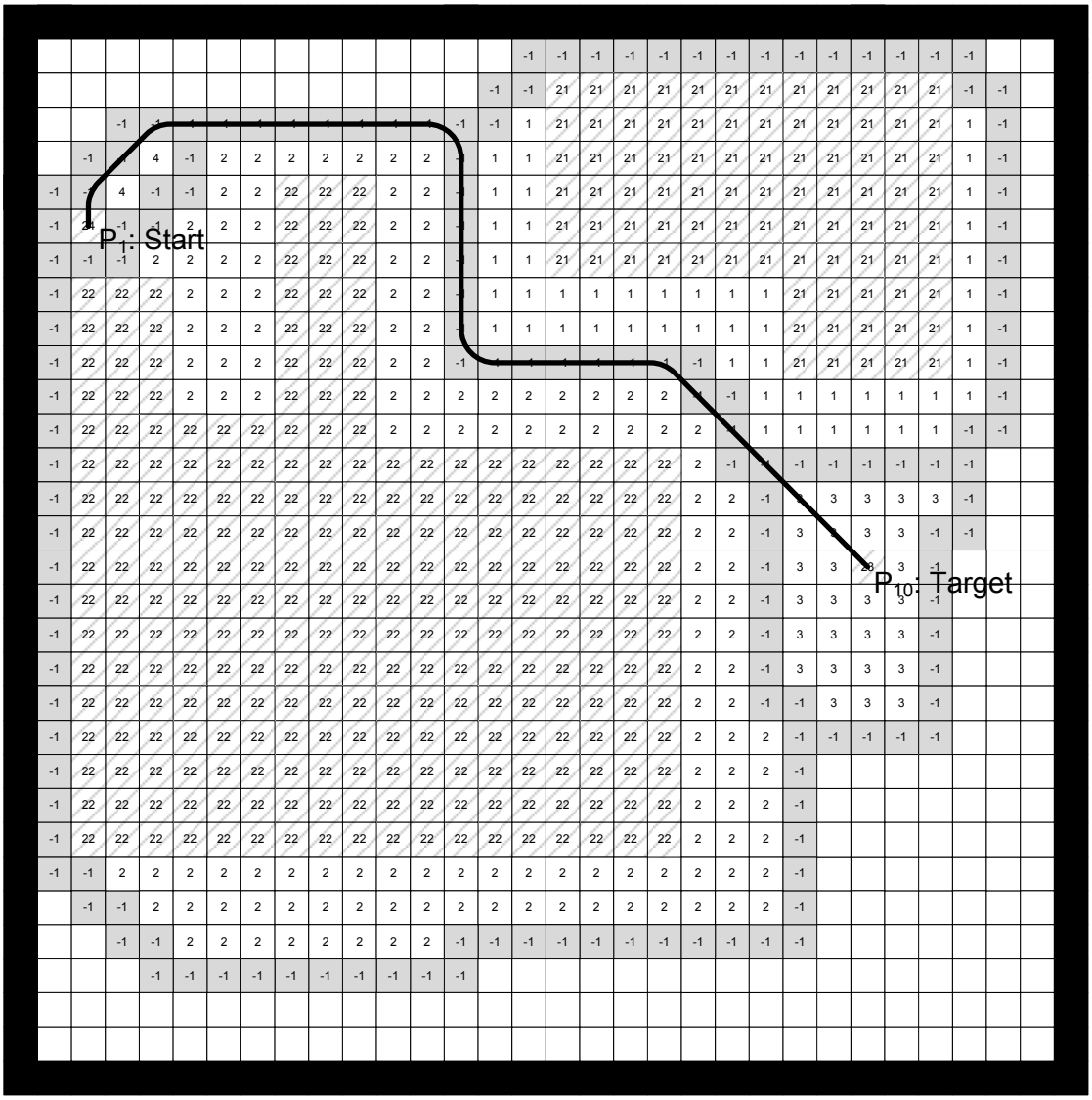


Figure 127: Trajectory through the roadmap without obstacle O3.

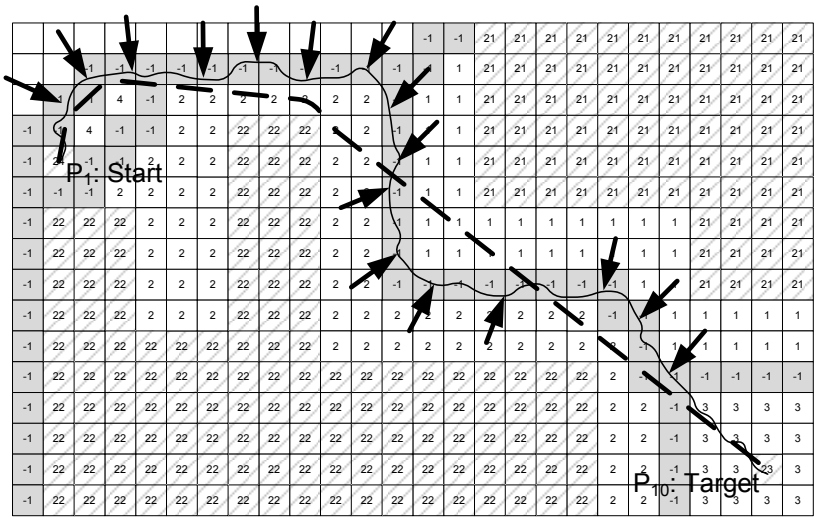


Figure 128: Elastic net trajectory generation.

The elastic net algorithm is parameterized with respect to its shrink forces. These forces (shown as arrows in Figure 128) move the particles on a straight line, and thus push the trajectory to the obstacles. The stronger the force, the more the trajectory is moved towards the obstacles, and the greater will be the number of collisions that may occur. The path-planning system first controls the real robot along a trajectory with low shrink forces applied to reduce the number of collision indications until either a collision is indicated or the target is reached. After the final trajectory is found, the shrink force may be raised to optimize the trajectory.

10.7 Re-planning of the Robot Path

As mentioned in Section 10.4, the search is executed within the roadmap corridor containing configuration space positions of the robot. During the movement execution of a solution, new information about the workspace and the obstacles may be added to the world model. This normally happens when collisions are indicated. With a dynamic update of the world model, a new search is initiated. The real robot stops its previous movements, moves back to the last common trajectory position and follows the new trajectory. Figure 129 illustrates the environment exploration; the resulting world model is updated to recognize obstacle O_3. As a result, the Voronoi roadmap plans a new trajectory around the newly added obstacle location. It turns out that these locations are also occupied, and therefore, a completely new trajectory is found, as shown in Figure 130, which is further modified as described in Section 10.4.

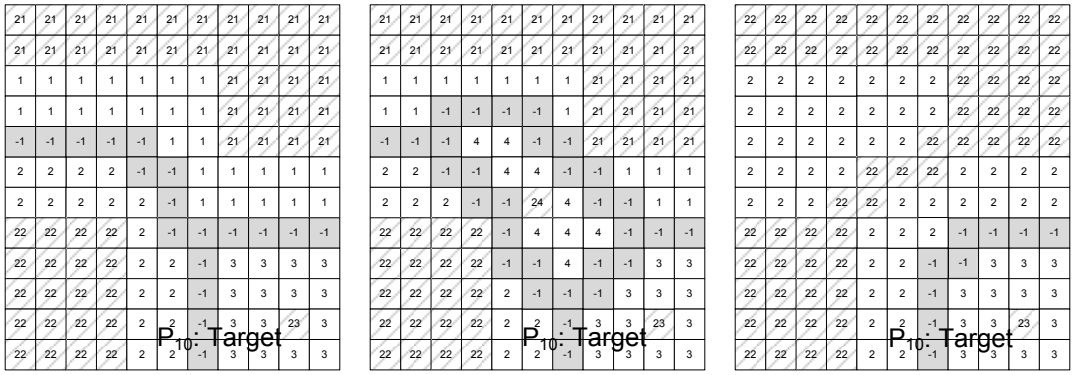


Figure 129: Adding collision indication positions (part of obstacle O3).

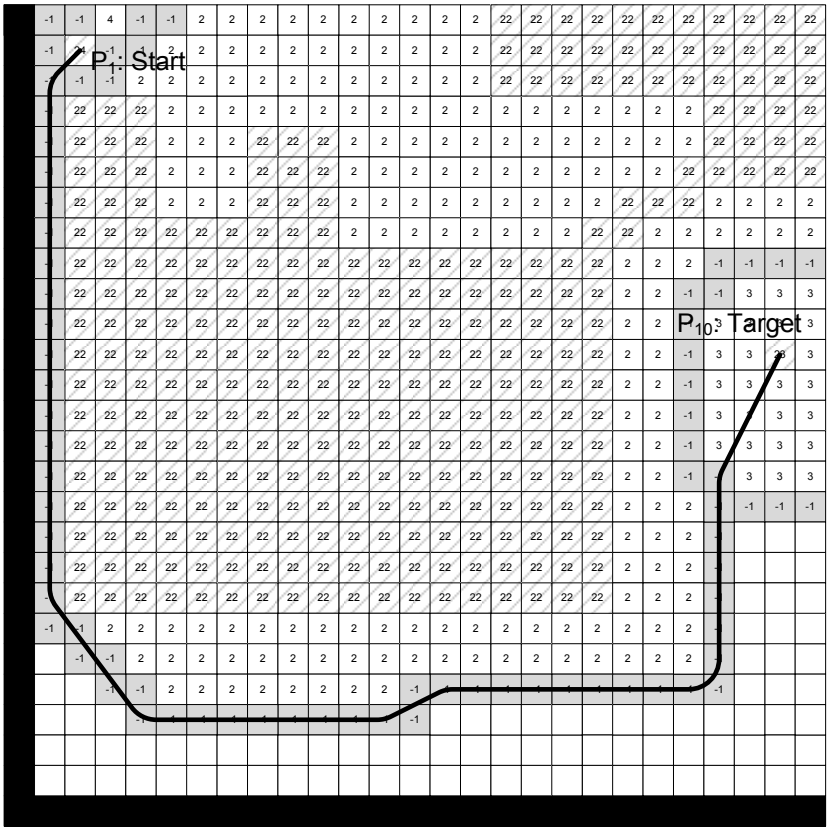


Figure 130: New re-planned path.

The path-planning algorithm searches within the robot configuration space of an articulated robot, which is located within the corridor. A configuration space is a part of the working space with a specific setting of the robot arm parameters ARM, ELBOW and FLIP (Abramowski, 1989, Siegert and Bocionek, 1996). The transition from one class into another class is not trivial. A solution would be to combine all configurations into a single configuration graph, and to detect neighbouring nodes of each configuration graph, adding an edge between them. Two positions within the configuration space graph may be connected, if there is a continuous function between them which is kinematically valid.

Robot arm parameters were not considered in this approach and illegal positions were considered as collisions.

10.8 Robot Program Generation

The scenario in Figure 126 was created with a mission containing the start and target locations as well as a single application path from P_7 to P_8. Once the mission is successfully planned, the robot program file may be generated.

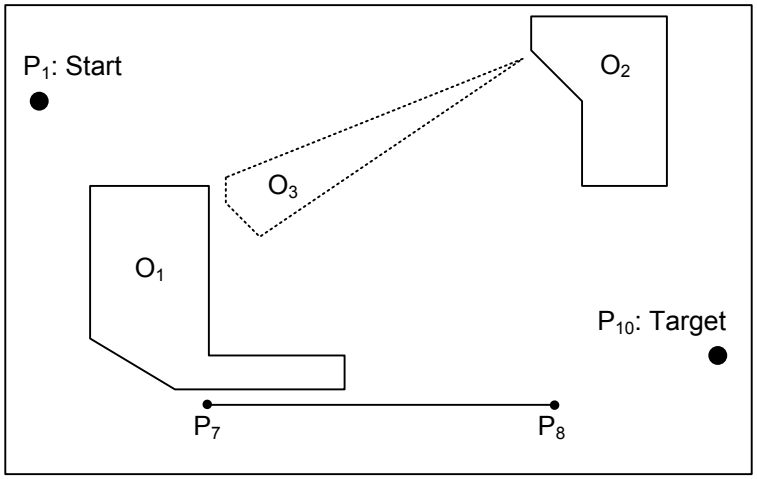


Figure 131: Experimental scenario.

The trajectory planning results are depicted in Figure 132, and are compared to the manually-programmed trajectory. The duration of the robot programming task is summarized in Section 10.9.

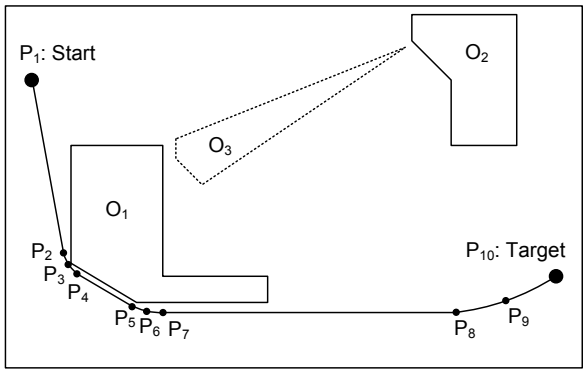


Figure 132: Automatically planned path.

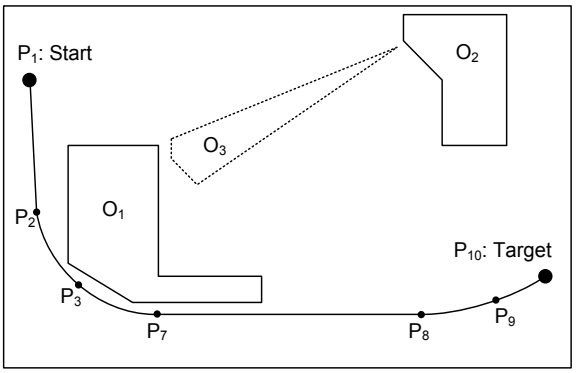


Figure 133: Manually planned path.

The program generation of the automatically planned path is template-based. Thus, only the dynamic content of the file is shown in Listing 12 and Listing 13:

```
10 MOV P2
20 MVR P2,P3,P7
30 HOPEN 1
40 MVS P8
50 MVR P8,P9,P10
```

Listing 12: Manually-programmed Melfa Basic IV file.

```
10 MOV P2
20 MVR P2,P3,P4
30 MVS P5
40 MVR P5,P6,P7
50 HOPEN 1
60 MVS P8
40 MVR P8,P9,P10
```

Listing 13: Automatically generated Melfa Basic IV robot program file.

The movement primitives circular and linear are respectively identified as *MVR* and *MVS* robot commands. The program, which is composed of 6-movement primitives, is still readable by a human. The final movements of the robot are comparable to the manually programmed ones. Manual modifications may still be carried out within the program, even for larger missions.

10.9 Robot Programming Duration

The overall time taken for the proposed system to generate a robot program file for the scenario was about 20 minutes (see Table 11, row 9), including mission preparation, data import and program file generation. The proposed system was compared with offline programming and conventional online programming methods. Both programming methods include the use of tools such as the Mitsubishi programming tool COSIROP/MELSOFT (Mitsubishi-Electric, 2008) or RobCAD. Offline programming and conventional online programming requires highly-skilled operators, while only a basic knowledge is required for supported robot programming. Online programming only considers the available physical objects, whereas offline and supported programming support models of these objects. The time taken for each step in the process is given in Table 11.

Id	Program execution time** Task	Online [seconds]	Supported [seconds]	Offline [seconds]
0	Offline programming	0	0	7200
1	System installation	10	600	10
2	DXF import	0	30	0
3	DXF placement	0	300	0
4	Set start/goal locations	60	60/0*	0-60
5	Set application locations	60	60/0*	0-60
6	Program or modify path	240	60	0-240
7	Save/upload program	20	10	20
8	Sum (online)	390	1120/1000	30-390
9	Sum (overall)	390	1120/1000	7230-7590

Table 11: Path planning execution times.

(* if locations are stored within DXF; ** for extrapolated times).

The times shown in Table 11 may be divided into fixed and task-dependent times. Usually, within an industrial setting, it is not required to place numerous models into the workspace; therefore, they may be seen as fixed.

Moreover, it should not be necessary to set the locations, although program generation is highly dependent on the size of the program (see rows 4-6 in Table 11).

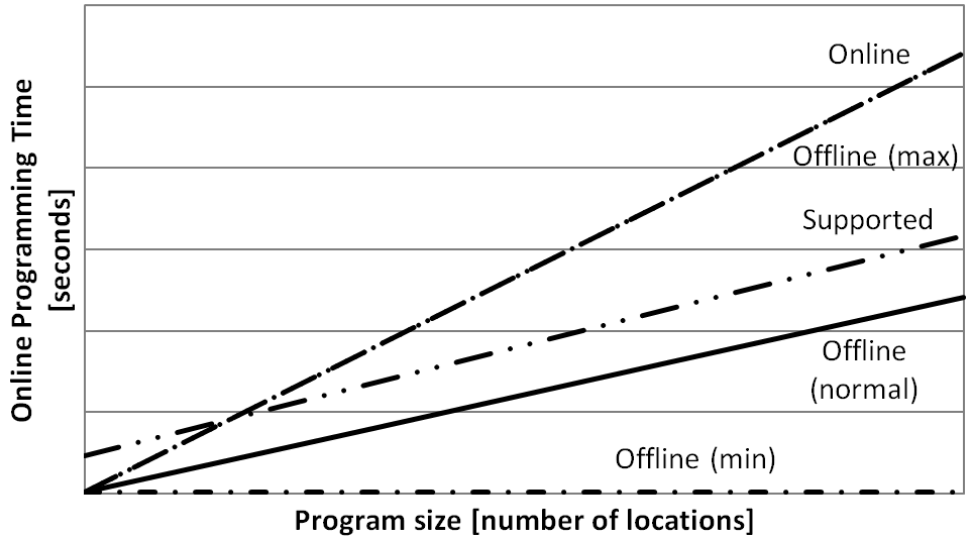


Table 12: Comparison of the online programming times.

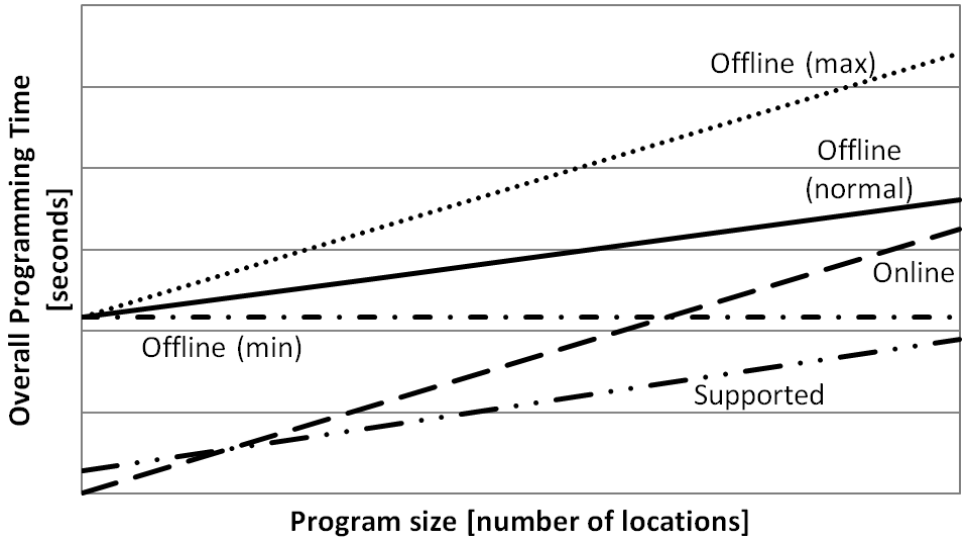


Table 13: Comparison of the overall programming times.

Table 12 illustrates the online programming time only and Table 13 represents the overall programming time for each programming method. Offline programming must be separated into minimum, maximum and normal values, which represent the online modification of the offline-generated program within the robot cell. The normal values may vary within the minimum and maximum values, depending on the quality of the offline generated robot program. Online programming can be applied very quickly, and should be used for small program sizes since the programming time significantly increases relative to the program size. Supported online programming requires an equal amount of time and a small fixed installation time when compared to normal values of the offline programming method.

Table 13 illustrates the online programming time including the preparation times, and it shows an additional preparation time for offline programming also mentioned in Table 11, row 0. The offline preparation time can be omitted entirely to save offline programming expenses, since the speed of programming for offline and supported online programming is equal. This is highly dependent on the quality of offline-generated programs, and may affect the ‘offline (normal)’ values in Tables 2 and 3. In the small example scenario presented, a total of 2 hours of offline programming, including the operator and the simulation tool, could be omitted, leading to cost savings. Therefore, supported online programming is recommended, especially for small batched manufacturing and high-volume production.

10.10 Summary

A trajectory planning approach has been presented based on the properties of the roadmap generation algorithm and the elastic net. The pre-calculated configuration space is deterministically sampled and stored within the octree cells. To reduce the search space, only configurations within the roadmap are considered during the A* search. The generated roadmap is based on the maximization of the clearance to obstacles in world space. It can be calculated simply and quickly, applying the proposed cell-based algorithm. New obstacle locations are dynamically added to the world model, which allows re-planning of the path. The elastic net optimizes the robot configurations of the found path to generate a manageable trajectory consisting of circular and linear movement primitives. It adapts itself to obstacles and to unreachable regions. Through the applied forces of the elastic net algorithm, the extent of adaptation is controlled. The presented algorithm is applicable for mobile and articulated robots working in a high-dimensional space.

One of the main benefits derived from this approach resides in the real time capability, which enables online robot programming. The Voronoi creation algorithm optimizes the Voronoi edges during real-time, which is an important aspect for practical use. Compared to offline programming, the presented approach does not require any pre-processing of information. The presented robot programming support system utilizing the trajectory-planning approach takes over the most complicated tasks, considering the basic knowledge of the operator. It renders offline systems as unnecessary, and helps to minimize robot-programming costs.

11 Discussion

Manufacture on an industrial scale may require the programming of robotic manipulator devices. In such cases, precise and accurate programming is necessary if error free production and high quality products are to be assured. Online robot programming is a time consuming and complicated task, and requires that the entire production system be put out of production for significant time periods while data is being entered. Only well trained operators are able to execute this step in a satisfying way with respect to obtaining quality outcomes and short durations. Thus, the outcome and duration are closely linked to the experience of the worker.

The complexity of programming remains one of the major hurdles preventing automation using industrial robots for small to medium sized manufacturing. Offline programming with a simulation system has been introduced for large volume manufacturing but the additional efforts in offline programming makes it inefficient for small to medium sized manufacturing. Although online programming methods have been researched in the past to make online programming more intuitive, less reliant on operator skill and more automatic, most of the research outcomes have not become commercially available.

The general research aim was to establish an enhanced online robot programming system, which helps the robot operator to create robot programs in an industrial production environment. Its use must be kept simple for the operator and it has to function with the delivered sensor data. The created robot program has to be manually changeable and maintainable. The framework is defined by the employed algorithm and the usage of the robot programming support system in real environments, together with the limited sensory input.

This work helps operators to improve their productivity. The acceptance of the robot programming system is dependent on its usability. The techniques applied in the system are of a complex nature, but are not transparent to the user. The interface only offers up front the information that is really needed and what is considered to be the most valuable information. The robot programming system is designed in such a way that it guides the operator throughout the process, and gives advice regarding the optimum manufacturing strategy and mission and trajectory planning.

The development of the system started by evaluating key requirements for the production industry to enhance online robot programming, especially when compared to

the offline programming approach. One important outcome of this work was the connection of the Mitsubishi RV-2AJ manipulator, together with its kinematics calculations and control. It also introduced a probabilistic world model which fuses sensor information required for automated path planning. The world model generates a roadmap that allows path planning in real-time, even with inaccurate and less sensor information. The found path is further transformed by an elastic net algorithm into a robot trajectory, and is subsequently generated into a robot program. A model-driven code generation framework helps to overcome the software implementation complexity.

The identified requirements for industrial robot programming include a fast robot programming approach that delivers high-quality results, and which is paired with intuitive usage. Considering the additional requirements of maintainability and reusability, a software design was proposed and implemented in this study.

Commanding the robot manipulator and receiving position information in a real-time manner is indispensable for the robot-programming framework. It has been shown that the employed Mitsubishi robot can be controlled in real-time using manually written software extensions. The integration of the programming system into the existing manufacture environment has been proven for the employed robot.

The developed probabilistic world model is optimized for Cartesian space, configuration space and CAD data. Each source type is stored in its own storage, and is queried by the roadmap generation algorithm. This was necessary because collisions of the robot manipulator are indicated by sensors, and the collision points have to be stored in the configuration space of the robot. Cartesian space and CAD data can be stored directly without any transformation. The CAD data is stored within Java3D, which is also used for visualization and collision checks. This approach considers CAD data, world and robot joint coordinates (obstacles and collision indication postures), and joins them in the octree. The transition is an important step, since inverse calculations of target positions for articulated robots often result in non-singular robot postures. Reported collisions occur in a single posture, and postures have therefore been stored within the octree cells for obstacle avoidance.

The achieved algorithm employs Voronoi roadmaps in the first instance. This allows a high probability for collision-free movements of the robot through the workspace, considering a minimum knowledge of obstacles within the environment. The Voronoi

roadmap supports path planning with only little sensory input, which is most often obtainable in real environments. While the robot is moving along the Voronoi path, collision information indicated by the operator or other sensors is used to improve the roadmap, and exploration of the environment therefore takes place. A trajectory planning approach has been presented based on the properties of the roadmap generation algorithm and the elastic net for the planning of missions with multiple goals. The pre-calculated configuration space is deterministically sampled and stored within the world space. Only configurations within the roadmap are considered during path searching to reduce the search space. The generated roadmap is based on the maximization of the clearance to obstacles in world space; thereby reducing the requirements for accuracy. It can be calculated quickly and easily by applying the proposed cell-based algorithm. New obstacle and collision locations are dynamically added to the world model, which allows the re-planning of the path.

Shortest path planning is executed on points along the Voronoi edges, and is optimized in the second stage to generate the trajectory. Although other solution candidates may be shorter after optimization, this approach presents a good approximation. This two-stage approach allows the use of low accuracies in the search stage, which speeds up the algorithm. The accuracy of the octree controls the capability of the path-searching algorithm to find small passages. The creation of discrete configuration-space elements is optimised for accuracy. An excess of discrete positions may lead to increased path planning times, whereas too few positions prevent the path planner from finding a solution.

The application of the elastic net both transforms the found path into a trajectory and optimizes that trajectory. It deforms and stretches the path to reduce the clearance to the obstacles, and the world model is thereby updated. This is an important feature to stretch or shorten the generated trajectories along Voronoi edges, which are otherwise not short and smooth.

The developed elastic net moves the robot configurations of the found path so that a trajectory consisting of circular and linear movement primitives is generated. It adapts itself to obstacles and to unreachable regions. Through the applied forces of the elastic net algorithm, the extent of the adaptation is controlled. Together with the Voronoi based roadmap, this path-planning approach provides a customised solution that handles inaccurate information.

The movement primitives are stored within the robot program file which considers the special syntax of the target robot programming language and can be uploaded directly to the robot system.

The tool is applicable to real industrial scenarios where articulated robots work in multi-dimensional spaces. One of the main benefits derived from this application is its real-time capability. By creating the opportunity to work successfully online, offline simulation systems become unnecessary; moreover, the overall time required for larger missions decreases. This support system is based on two specifics: the Voronoi roadmap and the elastic net, which both target the planning of missions with multiple goals. The new approach transforms the user interaction into a simplified task that generates acceptable trajectories which are applicable for industrial robot programming. In addition, it works successfully with only a basic knowledge of the operator, and requires the use of only the software application. The trade-off's optimality, path planning & smoothing, and maintainability are considered in the new approach. The new criteria maintainability and reusability were introduced, and the experiment has demonstrated that the system successfully addresses and satisfies the modern requirements emanating from the industrial market. The process is optimized, offline programming time may be saved, and online programming becomes easier.

12 Conclusions

The general research aim was to establish an enhanced online robot programming system, which helps the robot operator to create robot program files in an industrial production environment and which renders offline robot programming unnecessary.

The adoption of online programming by industry, objective one, was addressed in Chapter 5 and the results showed that a system for online robot programming was required which is able to generate robot programs online with a minimum production downtime. A fast robot program creation can only be realized with a simple HMI and an intelligent trajectory planning algorithm. The intelligent trajectory planning algorithm requires both, the availability of an efficient world model and a robot control framework as well as a robot model. A comparison with offline programming in Section 5.5 showed that the use of CAD data is important because real objects are not always at hand. The current offline system capabilities have to be met to replace the offline with online programming systems.

Objective two was addressed in Chapter 6 and the results showed that besides processing of the inexact sensor data to make them consistent, the types of information sources were important. Not only six dimensional position and orientation data in spatial space was required, but also robot joint space coordinates and CAD model data. In fact, this lead to three different world models merged into one. The merged world model provides the occupancy information to the trajectory planning system.

Objective three was addressed in Chapter 7 and the results showed the control of the Mitsubishi RV-2AJ robot manipulator and the Festo Robotino robots were possible. The robot control capability was important because the mission and trajectory planning algorithm moves the robots during online robot programming to explore the environment and to find a shortest trajectory. The robot kinematic model was important for the trajectory planner for forward and backward calculations during trajectory computations.

Objective four was addressed in Chapter 8 and the results showed that the required simple and efficient use of the system and the feasibility of trajectory planning within a real industrial environment were successfully solved with a cell based trajectory planning algorithm. It is based on Voronoi diagram approximation within a hierarchical data structure for the world model that also combines the robot joint and Cartesian space. The so found paths were transformed to particles in order to create trajectories, which were then transformed with templates to a robot program file. By combining the robot joint and Cartesian space, the search space was reduced to the cells only in order to allow the

Voronoi diagram approximation. The robot joint coordinates were then further used for the shortest path-finding algorithm.

Objective five was addressed in Chapter 9 and the results showed that domain specific modelling might simplify complex software systems but require a large amount of time for its implementation. In this work, the time for its implementation was larger than its benefit, thus, it will become more important for large projects and development teams or recurring projects in a specific domain.

In summary, the investigation has produced a new approach to the programming of robots in industry. The techniques developed in this study benefit an improvement in the speed of online robot programming and can render offline programming unnecessary. At the same time the quality of the automatic generated robot programs is equal compared to manually written or offline generated programs and it may still be improved in future. The costs for the equipment and infrastructure as well as the skilled workers for offline programming can be saved. The amount of time for pre-processing has been reduced drastically and helps to reduce costs. Overall, it is considered that the research has accomplished its stated aims. This study has provided a new and important contribution to the development of techniques for trajectory planning and assisted robot programming.

Finally, on a general scientific level, this work shows that technical solutions require a good usability in order to be practically applicable. The knowledge transfer between the human operator and the expert system has been implemented through a fluent workflow with a graphical user interface to guide the operator. The developed algorithms are targeted to an application in the real production system, but they can also be applied to offline robot programming systems to help the offline simulation expert to generate feasible and high quality robot trajectories. The implemented software development system for complex systems is not restricted to robotic applications and can be used for software development in general. The generic results of this research may be used in a wide variety of alternative applications in which trajectory planning is required. Not only in small to medium sized and high volume production industry but also in the diverse fields of research and development for further investigations into robot trajectory planning, home robots, surgery and health care assistant machines.

13 Future Work

The investigations introduced new findings in the field of assisted online robot programming, and proposed a new robot programming approach in the production industry. Further works can be undertaken based on the results of this study and in related areas.

This study introduced a new system for the enhancement of assisted robot programming. The operator benefits by obtaining advice regarding all robot programming tasks including trajectory planning, and the throughput is therefore increased. Further work is still required to enable the system to be applicable in different areas than manufacturing.

The enhanced online robot programming system should be integrated in the entire process as tightly as possible. One possibility would be the enhancement of the HMI to improve manual robot control and the pointing device. Manual robot control functionalities should be further developed, including the use of neural networks. For example, the joystick may be extended to move the robot, while unreachable portions of the world space may be automatically avoided.

The handling of dynamic obstacles should be further researched and the synchronization with the support system should be automated. Other cooperating robots are types of dynamic obstacles, which support information exchange for further integration into the mission-planning algorithm and the world model. This may enable the formation of a single, holistic world model of the production cell including all robots sharing their local world model and multi-robot control. This requires the exchange of world models and planning information. Because every robot has its own world model, these models have to be calibrated to obtain the absolute positions of each of the models.

Dynamic collision avoidance may lead to the permanent use of the proposed system. The flexibility of industrial robots can be optimized by allowing production robots to avoid moving obstacles while executing their pre-programmed task. Therefore, the identified obstacle types mentioned in Subsection 8.5.7 come into play. While this thesis handles only static and timely synchronized objects with predictable movement, other obstacle types such as timely unsynchronized obstacles or obstacles with unpredictable movement may also be considered.

The standard A* algorithm used may be extended in the future to the Anytime Dynamic A* (AD*) algorithm (Likhachev *et al.*, 2005). The uniform sampling scheme that was

applied in this work tends to have more joint coordinates within the corridor than are necessary. This has a direct impact on the performance of the path-finding algorithm. The proposed algorithm should be extended to use a non-uniform configuration space-sampling scheme.

The robot kinematic may be provided using a software module. However, it is not always possible to access those software modules. A learnable robot kinematic module may be employed to use any robot type, regardless of its geometry. Function approximator neural networks have also shown good results. Through supervised and unsupervised online learning, the input and output of the kinematic learning module may be optimized during runtime.

Mission- and task-specific extensions to the software have not yet been incorporated. These include application path information for welding, adhesive bonding and handling. The definition of the robot application path, e.g. spraying, gluing, painting, handling and cutting, should be further investigated to provide additional application-specific configurability. This gives the operator the ability to modify the outcome.

The complete software package was developed as a mixture of Java and C++ code, and required an additional communication layer, for example to call native functions directly from Java. For the integration of GUIs, only the middleware is sufficient, and it renders the communication layer obsolete. Therefore, it is intended to transfer the remaining software to Matlab/Simulink in order to improve the quality of the software.

References

AARNO, D., KRAGIC, D. and CHRISTENSEN, H.-I. (2004) Artificial Potential Biased Probabilistic Roadmap Method. *IEEE International Conference on Robotics and Automation.* 461–466.

ABRAMOWSKI, S. 1989. *Exakte Algorithmen zur Bewegung gelenkgekoppelter Objekte zwischen festen Hindernissen (Exact algorithms for the movement of joint-coupled objects between static obstacles).* PhD, TU Dortmund.

ADLER, D. and PHILIPP, T. 2011. *Dyncall Library* [Online]. Available: www.dyncall.org [Accessed 1.6.2011].

AKGUNDUZ, A., BANERJEE, P. and MEHROTRA, S. (2005) A Linear Programming Solution for Exact Collision Detection. *Computing and Information Science in Engineering.* 5**,** 48-55.

AL-MULHEM, M. and AL-MAGHRABI, T. (1997) An efficient algorithm to solve the E TSP. *IEEE Conference on Electrical and Computer Engineering.* 1**,** 269 - 272.

ARKIN, R. C. 1987. *Towards cosmopolitan robots: Intelligent navigation in extended man-made environments.* University of Massachusetts, Dept. of Computer and Information Science.

ARKIN, R. C. (1989) Navigational path planning for a vision-based mobile robot. *Robotica.* 7**,** 49-63.

ARKIN, R. C. (1992) Behavior-Based Robot Navigation for Extended Domains. *Journal of Adaptive Behavior.* 1**,** 201-225.

ARKIN, R. C. and CRAIG, R. (1989a) Motor Schema-Based Mobile Robot Navigation. *The International Journal of Robotics Research.* 8**,** 92-112.

AUPETIT, M., COUTURIER, P. and MASSOTTE, P. Function Approximation with Continuous Self Organizing Maps using Neighboring Influence Interpolation. Proceedings of Neural Computation, 2000, Berlin, Germany.

AURENHAMMER, F. (1991) Voronoi diagrams-a survey of a fundamental geometric data structure. *ACM Comput. Surv.* 23**,** 345-405.

BALMELLI, L., KOVAVEVI, J. and VETTERLI, M. (1999) Quadtrees for Embedded Surface Visualization: Constraints and Efficient Data Structures. *International Conference on Image Processing.* 2**,** 487-491.

BALZERT, H. 1999. Lehrbuch der Objektmodellierung (Workbook for Object Modelling), 3827402859, Spektrum Verlag.

BARRAQUAND, J. and LATOMBE, J. C. (1991) Robot motion planning: A distributed representation approach. *International Journal of Robotics Research.* 10**,** 628–649.

BHATTACHARYA, P. 2001. *Efficient Neighbor Finding Algorithms in Quadtree and Octree.* Master Thesis, India Inst. Technology, Kanpur.

BHATTACHARYA, P. and GAVRILOVA, M. L. (2008) Roadmap-Based Path Planning - Using the Voronoi Diagram for a Clearance-Based Shortest Path. *IEEE Robotics and Automation Magazine.* 15**,** 58-66.

BI, Z. M. and SHERMAN, Y. T. L. (2007) A Framework for CAD- and Sensor-Based Robotic Coating Automation. *Industrial Informatics, IEEE Transactions on.* 3**,** 84-91.

CECCARELLI, M. (ed.) BJORN SOLVANG, G. S. A. P. K. 2008. Robot Programming in Machining Operations, Robot Manipulators. In: Robot Manipulators. InTech.

BOADA, B. L., BLANCO, D. and MORENO, L. (2004) Symbolic Place Recognition in Voronoi-Based Maps by Using Hidden Markov Models. *Journal of Intelligent and Robotic Systems.* 39**,** 173-197.

BROOKS, R. A. (1983) Model-Based Three Dimensional Interpretations of Two Dimensional Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 5**,** 140 - 150.

CHENG, G. and ZELL, A. (1999) Multiple Growing Cell Structures. *Neural Network World.* 5**,** 425-452.

CHUANG, J.-H. (1998) Potential-based modeling of three dimensional workspace for obstacle avoidance. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION.* 14**,** 778 - 785.

CLOUDGARDEN (2011) Jigloo [Online]. Available: www.cloudgarden.com/jigloo [Accessed 27.6.2013].

COMUNITY (2012) ODEJava [Online]. Available: http://java.net/projects/odejava [Accessed 13.6.2013].

CONNOLLY, C. I. (1992) Applications of harmonic functions to robotics. *Proceedings of the 1992 IEEE International Symposium on Intelligent Control.* 498-502.

CORKE, P. I. (1996) A Robotics Toolbox for Matlab. *IEEE Robotics and Automation Magazine.* 3**,** 24-32.

CORKE, P. I. (2005) Machine Vision Toolbox. *IEEE Robotics and Automation Magazine.* 12**,** 16-25.

CRAIG, J. J. 2003. Introduction to Robotics: Mechanics and Control, 0201543613, Prentice Hall.

DEMIRIS, Y. and BILLARD, A. (2007) Special Issue on Robot Learning by Observation, Demonstration, and Imitation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics.* 37**,** 254-255.

DENAVIT, J. and HARTENBERG, R. S. (1955) A kinematic notation for lower-pair mechanisms based on matrices. *Journal of Applied Mechanics.* 1**,** 215-221.

DI, X., GHOSH, B. K., NING, X. and TZYH, J. T. (1998) Intelligent robotic manipulation with hybrid position/force control in an uncalibrated workspace. *Proceedings IEEE International Conference on Robotics and Automation.* 2**,** 1671–1676.

DONALD, B.-R., XAVIER, P.-G., CANNY, J.-F. and REIF, J.-H. (1993) Kinodynamic Motion Planning. *Journal of the ACM.* 40**,** 1048-1066.

DUBINS, L. E. (1957) On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics.* 79**,** 497-516.

ECLIPSE FOUNDATION. 2006. *eclipse IDE* [Online]. Available: http://www.eclipse.org [Accessed 2.5.2013].

ECLIPSE FOUNDATION. 2011a. *Eclipse Modelling Framework (EMF)* [Online]. Available: http://www.eclipse.org/modeling/emf [Accessed 24.3.2011].

ECLIPSE FOUNDATION. 2011b. *GMF* [Online]. Available: http://www.eclipse.org/modeling/gmp [Accessed 5.6.2011].

ECLIPSE FOUNDATION. 2011c. *Java Emitter Templates (JET)* [Online]. Available: http://www.eclipse.org/modeling/m2t/?project=jet [Accessed 13.6.2013].

ECLIPSE FOUNDATION. 2011d. *Xtext* [Online]. Available: http://www.eclipse.org/Xtext [Accessed 5.7.2011].

EDELSBRUNNER, H. 1987. Algorithms in Combinatorial Geometry, Berlin, Heidelberg, 978-3540137221, Springer

ERDMANN, M. and LOZANO-PEREZ, T. (1987) On Multiple Moving Objects. *Algorithmica.* 2**,** 477-521.

ETRICE GROUP. 2011. *Eclipse eTrice project page* [Online]. Available: http://www.eclipse.org/etrice [Accessed 13.6.2013].

EURON. 2012. *Web Page of the European Robotics Research Network (EURON)* [Online]. Available: http://www.euron.org [Accessed 13.6.2013].

EVELOPERS CORPORATION. 2011. *UniMod* [Online]. Available: http://unimod.sourceforge.net [Accessed 13.6.2013].

FESTO. 2011. *Robotino* [Online]. Festo. Available: www.robotino.de [Accessed 13.6.2013].

SCHIEHLEN, W. (ed.) FISETTE, P. and SAMIN, J. C. 1993. In: Advanced Multibody System Dynamics. Kluwer Academic Publishers, 373-378.

FLANAGAN, D. 2002. Java in a Nutshell, 389721332X, O'Reilly Media.

FRAICHARD, T. (1999) Trajectory Planning in a Dynamic Workspace: a 'State-Time Space' Approach. *Advanced Robotics.* 13**,** 75-94.

FRITZKE, B. (1991) Let It Grow: Self-Organizing Feature Maps with Problem Dependent Cell Structure. *Proceedings of ICANN.* 403-308.

FRITZKE, B. (1995) A Growing Neural Gas Network Learns Topologies. *Advances in Neural Information Processing Systems.* 625–632.

FRITZKE, B. and WILKE, P. (1991) FLEXMAP A Neural Network for the Traveling Salesman Problem with Linear Time and Space Complexity. *IEEE International Joint Conference on Neural Networks.* 2**,** 929 - 934.

FUJIMURA, K. (1995) Time-minimum routes in time-dependent networks. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION.* 11(3)**,** 343-351.

GANDHI, D. and CERVERA, E. (2003) Sensor covering of a robot arm for collision avoidance. *IEEE International Conference on Systems, Man and Cybernetics.* 5**,** 4951 - 4955.

GARGA, A. K. and BOSE, N. K. (1994) A neural network approach to the construction of Delaunay tessellation of points in Rd. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on.* 41**,** 611 -613.

GARGANTINI, I. (1982a) An Effective Way to Represent Quadtrees. *Commun. ACM.* 25**,** 905-910.

GARGANTINI, I. (1982b) Linear octtrees for fast processing of three-dimensional objects. *Computer Graphics and Image Processing.* 20(4)**,** 363-374.

GARLAND, M. (1999) Multiresolution Modeling Survey and Future Opportunities. *Eurographics '99 -- State of the Art Reports.* 111-131.

GE, S. S. (2004) Differential neural networks for robust nonlinear control. *International Journal of Adaptive Control and Signal Processing.* 18**,** 315–316.

GE, S. S. and CUI, Y. J. (2000) New potential functions for mobile robot path planning. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION.* 16.

GLAVINA, B. 1990. *Planung kollisionsfreier Bewegungen für Manipulatoren durch Kombination von zielgerichteter Suche und zufallsgesteuerter Zwischenzielerzeugung (Planning of collisions free movements for manipulators through combination of target directed search and random controlled intermediate target creation).* Thesis, Technische Universitat München.

GLOBUS, A. Octree Optimization. Symposium on Electronic Imaging Science and Technology, 1991.

GONZALEZ-GALVAN, E. J., LOREDO-FLORES, A., LABORICO-AVILES, E. D., PAZOS-FLORES, F. and CERVANTES-SANCHEZ, J. J. An algorithm for optimal closed-path generation over arbitrary surfaces using uncalibrated vision. IEEE International Conference on Robotics and Automation, 10-14 April 2007 2007. 2465-2470.

GOTO, T., KOSAKA, T. and NOBORIO, H. On the heuristics of A* or A algorithm in ITS and robot path-planning. Intelligent Robots and Systems, 27-31 Oct. 2003. 1159-1166.

GRAN, C. A. 1999. *Octree-based Simplifications of Polyhedral Solids.* Thesis, Universitat Politècnica de Catalunya.

GRIPH, F. S., HOGBEN, C. H. A. and BUCKLEY, M. A. (2004) A generic component framework for real-time control. *IEEE Transactions on Nuclear Science.* 51**,** 558-564.

GUTMANN, J. S., WEIGEL, T. and NEBEL, B. (2001) A fast, accurate, and robust method for self-localization in polygonial environments using laser-range-fingers. *Advanced Rabotica.* 14**,** 651-668.

HAEGELE, M., NEUGEBAUER, J. and SCHRAFT, R.-D. (2001) From Robots to Robot Assistants. *International Symposium on Robotics.*

HÄGELE, M., SCHAAF, W. and HELMS, E. (2002) Robot Assistants at Manual Workplaces: Effective Co-operation and Safety Aspects. *International Symposium on Robotics.*

HALL, D. L. and LLINAS, J. (1997) An introduction to multisensor data fusion. *PROCEEDINGS OF THE IEEE.* 85**,** 6 -23.

HEIM, A. 1999. Modellierung, Simulation und optimale Bahnplanung bei Industrierobotern (Modelling, simulation and optimal path planning for industrial robots), 3896754629, 9783896754622, Herbert Utz Verlag.

HENRICH, D., WURLL, C. and WORN, H. Online path planning with optimal C-space discretization. IEEE/RSJ International Conference on Intelligent Robots and Systems, 13-17 Oct 1998. 1479-1484.

HILTON, A., STODDART, A. J., ILLINGWORTH, J. and WINDEATT, T. (1996) Reliable surface reconstruction from multiple range images. *Lecture Notes in Computer Science.* 1064**,** 117-126.

HOFF, K., CULVER, T., KEYSER, J., LIN, M. C. and MANOCHA, D. Interactive motion planning using hardware-accelerated computation of generalized Voronoi diagrams. International Conference on Robotics and Automation, 2000. IEEE, 2931-2937.

HOFF, K. E., CULVER, T., KEYSER, J., LIN, M. and MANOCHA, D. Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware. SIGGRAPH '99, 1999. ACM Press/Addison-Wesley Publishing Co.

HOPPE, H. (1998) Efficient Implementation of Progrssive Meshes. *Computers & Graphics.* 22**,** 27-36.

HSU, D., KINDEL, R., LATOMBE, J. and ROCK, S. (2002) Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research.* 21**,** 233-255.

HU, Z., MARSHALL, C., BICKER, R. and TAYLOR, P. (2007) Automatic surface roughing with 3D machine vision and cooperative robot control. *Robotics and Autonomous Systems.* 55**,** 552-560.

HUI, Z., HEPING, C., NING, X., ZHANG, G. and JIANMIN, H. On-Line Path Generation for Robotic Deburring of Cast Aluminum Wheels. International Conference on Intelligent Robots and Systems, 2006. IEEE/RSJ, 2400-2405.

HWANG, J. Y., KIM, J. S., LIM, S. S. and PARK, K. H. (2003) A fast path planning by path graph optimization. *IEEE Transactions on Systems, Man and Cybernetics, Part A.* 33**,** 121-129.

IBM CORP. 2011. *Rational Rose Real-Time* [Online]. Available: http://www.ibm.com [Accessed 27.6.2013].

INGAKI, H., SUGIHARA, K. and SUGIE, N. Numerically robust incremental algorithm for constructing three-dimensional Voronoi diagrams. Proc. 4th Canad. Conf. Comput. Geom., 1992. 334--339.

INTERNATIONAL FEDERATION OF ROBOTICS 2005. World Robotics 2005, New York/Geneva, United Nations Publications.

INTERNATIONAL FEDERATION OF ROBOTICS 2011. World Robotics 2011, United Nations Publications.

IVRISSIMTZIS, L. P., JEONG, W. K. and SEIDEL, H. P. Using Growing Cell Structures for Surface Reconstruction. Shape Modeling International, 2003. 78 - 86.

KAGAMI, S., KUFFNER, J. J., NISHIWAKI, K., OKADA, K., INABA, M. and INOUE, H. Humanoid arm motion planning using stereo vision and RRT search. Intelligent Robots and Systems, 2003. IEEE/RSJ, 2167 - 2172.

KAIN, S., HEUSCHMANN, C. and SCHILLER, F. (2008) Von der virtuellen Inbetriebnahme zur Betriebsparallelen Simulation (From the virtual setup to production parallel simulation). *Atp-Edition.* 8**,** 48-52.

KANT, K. and ZUCKER, S. W. (1986) Toward efficient trajectory planning: The path-velocity decomposition. *Int. I. of Robotics Research.* 5(3)**,** 72-89.

KAZEMI, M. and MEHRANDEZH, M. (2004a) Robotic Navigation Using Harmonic Function-based Probabilistic Roadmaps. *Proceedings of IEEE International Conference on Robotics and Automation.* 4765-4770.

KAZEMI, M., MEHRANDEZH, M. and GUPTA, K. (2005) An Incremental Harmonic Function-based Probabilistic Roadmap Approach to Robot Path Planning.

Proceedings of IEEE International Conference on Robotics and Automation. 2148-2153.

KHATIB, O. (1986) Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Rob. Res.* 5**,** 90-98.

KIEFER, J., BERGERT, M. and ROSSDEUTSCHER, M. (2010) Mechatronic objects in production engineering. *Atp-Edition.* 12**,** 36-45.

KIM, J. O. and KHOSLA, P. K. (1992) Real-time obstacle avoidance using harmonic potential functions. *IEEE Transactions on Robotics and Automation.* 8**,** 338-349.

KIM, K.-Y., KIM, D.-W. and NNAJI, B. O. (2002) Robot arc welding task sequencing using genetic algorithms. *IIE Transactions.* 34**,** 865-880.

KNUTH, D. E. 1973. The Art of Computer Programming - Sorting and Searching, Reading, MA, Addison-Wesley.

KODITSCHEK, D. E. and RIMON, E. (1990) Robot navigation functions on manifolds with boundary. *Adv. Appl. Math.* 11**,** 412-442.

KRISTENSEN, S., HORSTMANN, S., KLANDT, J., LOHNERT, F. and STOPP, A. Human-Friendly Interaction for Learning and Cooperation. Robotics and Automation, 2001. IEEE, 2590 - 2595.

KRISTENSEN, S., NEUMANN, M., HORSTMANN, S., LOHNERT, F. and STOPP, A. (2002) Tactile Man-Robot Interaction for an Industrial Service Robot. *Lecture Notes in Computer Science.* 2238**,** 177-194.

KUCUK, S. and BINGUL, Z. (2004) The Inverse Kinematics Solutions of Industrial Robot Manipulators. *IEEE Conferance on Mechatronics.* 274-279.

CUBERO, S. (ed.) KUCUK, S. and BINGUL, Z. 2006. Robot Kinematics: Forward and Inverse Kinematics. In: Industrial-Robotics-Theory-Modelling-Control. Pro Literatur Verlag, 964.

LANGLOIS, D., ELLIOTT, J. and CROFT, E. A. (2001) Sensor uncertainty management for an encapsulated logical device architecture Part II: A control policy for sensor uncertainty. *American Control Conference.*

LATOMBE, J.-C. 1991. Robot Motion Planning, Kluwer Academic Publishers.

LAVALLE, S. and KUFFNER, J. J. Rapidly-exploring random trees: Progress and prospects. Algorithmic Foundations of Robotics, 2000.

LAVALLE, S. M. 2006. Planning Algorithms, Cambridge University Press.

LAVENDER, D., BOWYER, A., DAVENPORT, J., WALLIS, A. and WOODWARK, J. (1992) Voronoi diagrams of set-theoretic solid models. *Computer Graphics and Applications, IEEE.* 12**,** 69-77.

LEBEDEV, D. V., STEIL, J. J. and RITTER, H. Real-time path planning in dynamic environments a comparison of three neural network models. Systems, Man and Cybernetics, 2003b. 3408 - 3413.

LEE, D. T. (1982) Medial Axis Transformation of a Planar Shape. *Pattern Analysis and Machine Intelligence, IEEE Transactions on.* PAMI-4**,** 363-369.

LEE, M. and SAMET, H. (2000) Navigating through triangle meshes implemented as linear quadtrees. *ACM Trans. Graph.* 19**,** 79-121.

LENZ, A. and PIPE, A. G. (2003) A dynamically sized radial basis function neural network for joint control of a puma 500 manipulator. *IEEE International Symposium on Intelligent Control.* 170 - 175.

LIPPIELLO, V. (2005) Real-time visual tracking based on BSP-tree representations of object boundary. *Robotica.* 23**,** 365-375.

MAËL, E. (1996) A Hierarchical Network for Learning Robust Models of Kinematic Chains. *ICANN.*

MAHLER, S. 2003. *Erzeugung und Evaluierung von Oktalbaumstrukturen als Schnittstelle zu CAD-Programmen (Creation and evaluation of Octal tree structures as interface for CAD applications).* Master Thesis, Institut für Parallele und Verteilte Systeme Universität Stuttgart.

MALETZKI, G., PAWLETTA, T., DÜNOW, P. and LAMPE, B. (2008) Simulationsmodellbasiertes Radpid Prototyping von komplexen Robotersteuerungen. *Atp-Edition.* 8**,** 54-60.

MAO-LIN, N. and MENG, J. E. (2000) Decentralized control of robot manipulators with couplings and uncertainties. *American Control Conference.* 3326–3330.

MARTIN, H. (1998) Voronoi diagrams and offset curves of curvilinear polygons. *Computer-Aided Design.* 30**,** 287-300.

MASEHIAN, E. and AMIN-NASERI, M. R. (2004) A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning. *J. Robot. Syst.* 21**,** 275-300.

MASOUD, S. A. and MASOUD, A. A. (2000) Constrained Motion Control Using Vector Potential Fields. *IEEE transactions on systems, MAN, and Cybernetics-Part A: Systems and Humans.* 30.

MATHWORKS 1997. Image Processing Toolbox Version 2, Mathworks.

MERKLE, P. E. 2004. *Entwicklung eines Octree-Verfahrens zur 3D-Volumenrekonstruktion auf Voxelbasis (Development of an octree procedure for the 3D volume reconstruction on the basis of voxel).* Technische Universitätä Berlin.

MILENKOVIC, V. (1993) Robust polygon modelling. *Computer-Aided Design.* 25**,** 546-566.

MÍNGUEZ, J., MONTANO, L. and SANTOS-VICTOR, J. (2002) Reactive Navigation for Non-holonomic Robots using the Ego-Kinematic Space. *IEEE lntemational Conference on Robotics \& Automation.*

MITSUBISHI-ELECTRIC (2002a) MELFA Industrial Robots Instruction Manual Controller CR1 (Mitsubishi-Electric, Ratingen, Germany) [Online]. Available: http://www.mitsubishi-automation.com [Accessed 27.6.2013].

MITSUBISHI-ELECTRIC (2002b) MELFA Industrial Robots Instruction Manual Controller CR1/CR2/CR2A (Mitsubishi-Electric, Ratingen, Germany) [Online]. Available: http://www.mitsubishi-automation.com [Accessed 27.6.2013].

MITSUBISHI-ELECTRIC (2003) MELFA Industrial Robots Instruction Manual (Functions and Operations) CR1/CR2/CR3/CR4/CR7/CR8 Controller (Mitsubishi-Electric, Ratingen, Germany) [Online]. Available: http://www.mitsubishi-automation.com [Accessed 27.6.2013].

MITSUBISHI-ELECTRIC (2011) COSIROP and COSIMIR [Online]. Available: www.mitsubishi.com [Accessed 27.6.2013].

MYOUNG HWAN, C. and WOO WON, L. A force/moment sensor for intuitive robot teaching application. Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on, 2001 2001. 4011-4016 vol.4.

NANDI, G. C. and MITRA, D. (2005) Fusion Strategies for Minimizing Sensing-Level Uncertainty in Manipulator Control. *Journal of Intelligent and Robotic Systems.* 43**,** 1-32.

NICHOLSON, A. 2005. *Rapid adaptive programming using image data.* PhD, University of Wollongong.

OKABE, A., BOOTS, B., SUGIHARA, K., CHIU, S. N. and KENDALL, D. G. 2008. References. In: Spatial Tessellations. John Wiley & Sons, Inc., 585-655.

PAN, Z., POLDEN, J., LARKIN, N., DUIN, S. V. and NORRISH, J. (2010) Recent Progress on Programming Methods for Industrial Robots. *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK).* 1-8.

PAN, Z. and ZHANG, H. Robotic programming for manufacturing industry. International Conference on Mechanical Engineering and Mechanics, 5-7 Nov. 2007, Wuxi, China.

PAPULA, L. 1998. Mathematische Formelsammlung (Mathematical formulary), Braunschweig, 3-528-44442-8, Vieweg.

PAUL, R. P. 1981. Robot Manipulators: Mathematics, Programming and Control, 026216082X, MIT Press.

PAYEUR, P. Improving robot path planning efficiency with probabilistic virtual environment models. Virtual Environments, Human-Computer Interfaces and Measurement Systems, 2004. IEEE 13 - 18.

PIPE, A. G. (2001) An architecture for learning 'potential field' cognitive maps with an application to mobile robot navigation. *Journal of Adaptive Behaviour.* 8(2)**,** 173-204.

PIRES, J. N., GODINHO, T. and FERREIRA, P. (2004) CAD interface for automatic robot welding programming. *Industrial Robot: An International Journal.* 31**,** 71 - 76.

POLLEFEYS, M. (2000) Tutorial on 3D Modeling from Images (ECCV 2000) [Online]. Available: http://www.cs.unc.edu/~marc/tutorial.pdf.

PRASSLER, E., BANK, D. and KLUGE, B. (2002) Key Technologies in Robot Assistants: Motion Coordination between a Human and a Mobile Robot.

PREPARATA, F. P. and SHAMOS, M. I. 1985. Computational geometry: an introduction, 0387961313, Springer.

QUEK, F., JAIN, R. and WEYMOUTH, T. E. (1993) An abstraction-based approach to 3-D pose determination from range images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on.* 15**,** 722-736.

RANGANATHAN, A. and KOENIG, S. PDRRTs: Integrated Graph Based and Cell Based planning. Intelligent Robots and Systems, 2004. IEEE/RSJ, 2799 - 2806.

RAO, T. M. and ARKIN, R. C. (1990a) 3D Navigational Path Planning. *Robotica.* 8**,** 195-205.

RAO, T. M. and ARKIN, R. C. 3D path planning for flying crawling Robots. SPIE 1195, 1990b.

RAUBER, A., MERKL, D. and DITTENBACH, M. (2002) The Growing Hierarchical Self Organizing Map Exploratory Analysis of High-Dimensional Data. *IEEE Transactions on Neural Networks.* 13**,** 1331 - 1341.

REIF, J. H. and WANG, H. (2000) Nonuniform Discretization for Kinodynamic Motion Planning and its Applications. *SIAM Journal on Computing.* 30**,** 161-190.

RITTER, H., MARTINETZ, T. and SCHULTEN, K. 1992. Neural Computation and Self-Organizing Maps, Addison-Wesley.

RITTER, H., MARTINETZ, T. and SCHULTEN, K. 1994. Neuronale Netze (Neural Networks), Oldenbourg.

ROSELL, J. and INIGUEZ, P. (2005) Path planning using Harmonic Functions and Probabilistic Cell Decomposition. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation.* 1803- 1808.

RUSSELL, S.-J. and NORVIG, P. 2002. Artificial Intelligence: A Modern Approach (2nd Edition), Prentice Hall.

SAMET, H. 1990. Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS, Reading, MA, Addison-Wesley Longman.

SAMET, H. 1994. The Design and Analysis of Spatial Data Structures, 0201502550, Addison Wesley.

SÁNCHEZ, G. and LATOMBE, J.-C. (2003) A Single Query Bi Directional Probabilistic Road map Planner with Lazy Collision Checking. *Springer Tracts in Advanced Robotics.* 6**,** 403-417.

SÁNCHEZ, G. and LATOMBE, J. C. (2002) On delaying collision checking in PRM planning: application to multi-robot coordination. *Int. Journal of Robotics Research.* 21**,** 5-26.

SCHIEHLEN, W. O. 1990. Multibody Systems Handbook, 978-3-642-50997-1, Springer.

SCHRACK, G. (1992) Finding neighbors of equal size in linear quadtrees and octrees in constant time. *CVGIP: Image Underst.* 55**,** 221-230.

SCHWARZER, F., SAHA, M. and LATOMBE, J. C. (2004) Exact Collision Checking of Robot Paths. *Algorithmic Foundations of Robotics.* 5**,** 25-42.

SELIC, B., GULLEKSON, G. and WARD, P. T. 1994. Real-time object-oriented modeling, 0471599174, Wiley Professional Computing.

SHAMOS, M. I. and HOEY, D. Closest-point problems. Foundations of Computer Science, 1975., 16th Annual Symposium on, 13-15 Oct. 1975 1975. 151-162.

SIEGERT, H.-J. and BOCIONEK, S. 1996. Robotik, Programmierung intelligenter Roboter (Robotics, programming of intelligent robots), Springer, Berlin.

SMITH, R. 2012. *Open Dynamics Engine* [Online]. Available: http://www.ode.org/ [Accessed 26.07.2012].

SMITH, R., SELF, M. and CHEESEMAN, P. 1990. Estimating Uncertain Spatial Relationships in Robotics. In: Autonomous Robot Vehicles. Springer, Berlin, 167–193.

SPARX SYSTEMS. 2011. *Enterprise Architect* [Online]. Available: http://www.sparxsystems.com.

SPONG, M. W., HUTCHINSON, S. and VIDYASAGAR, M. 2004. Robot Dynamics and Control.

STOPP, A., BALDAUF, T., HANTSCHE, R., HORSTMANN, S., KRISTENSEN, S., LOHNERT, F., PRIEM, C. and RIISCHER, B. The Manufacturing Assistant: Safe, Interactive Teaching of Operation Sequences. Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on, 2002. 386 - 391.

SUGIHARA, K. and IRI, M. (1994) A robust Topology-Oriented Incremental algorithm for Voronoi diagrams. *International Journal of Computational Geometry and Applications.* 179-228.

SUGITA, S., ITAYA, T. and TAKEUCHI, Y. (2004) Development of robot teaching support devices to automate deburring and finishing works in casting. *The International Journal of Advanced Manufacturing Technology.* 23**,** 183-189.

SUN-MICROSYSTEMS. 2012. *Java 3D* [Online]. Available: http://java3d.java.net/ [Accessed 25.07.2012].

SUTHERLAND, J. (1998) Why I love the OMG: emergence of a business object component architecture. *StandardView.* 6**,** 4-13.

TAKARICS, B., SZEMES, P. T., NEMETH, G. and KORONDI, P. Welding trajectory reconstruction based on the Intelligent Space concept. Human System Interactions, 2008 Conference on, 25-27 May 2008 2008. 791-796.

THEMATHWORKS. 2011. *Matlab/Simulink* [Online]. Available: http://www.mathworks.de.

THIEMERMANN, S. 2005. *Direkte Mensch-Roboter-Kooperation in der Kleinteilemontage mit einem SCARA-Roboter (Direct human robot cooperation for small pieces assembing with a SCARA robot).* Thesis, Universität Stuttgart.

VALAVANIS, K. P., HEBERT, T., KOLLURU, R. and TSOURVELOUDIS, N. (2000) Mobile robot navigation in 2 D dynamic environments using an electrostatic potential field. *IEEE Transactions on Systems, MAN, and Cybernetics—Part A: Systems and Humans.* 30.

VLEUGELS, J. M., KOK, J. N. and OVERMARS, M. H. (1993) Motion Planning Using a Colored Kohonen Network. *RUU-CS.*

VUKOBRATOVIC, M. and KIRCANSKI, N. 1982. Real-Time Dynamics of Manipulation Robots, Springer Verlag.

WARREN, C. W. Global path planning using artificial potential fields. Robotics and Automation, 1989. 316 - 321.

WAYDO, S. Vehicle motion planning using stream functions. Robotics and Automation, 2003. IEEE, 2484 - 2491.

WENRUI, D. and KAMPKER, M. PIN-a PC-based robot simulation and offline programming system using macro programming techniques. Industrial Electronics Society, 1999. IECON '99 Proceedings. The 25th Annual Conference of the IEEE, 1999 1999. 442-446 vol.1.

WESTKÄMPER, E., SCHRAFT, R. D., NEUGEBAUER, J.-G. and RITTER, A. (1999) Holonic Task Generation for Mobile Robots. *30th International Symposium on Robotics.* 553-560.

WHEELER, M. 1996. *Automatic modeling and localization for object recognition.* Carnegie Mellon University.

WÖSCH, T., NEUBAUER, W., V. WICHERT, G. and KEMÉNY, Z. Robot Motion Control for Assistance Tasks. Robot and Human Interactive Communication, 2002. IEEE, 524 - 529.

XIANG, L. and DAOXIONG, G. A comparative study of A-star algorithms for search and rescue in perfect maze. Electric Information and Control Engineering (ICEICE), 2011 International Conference on, 15-17 April 2011 2011. 24-27.

YANG, D.-H. and HONG, S.-K. (2007) A roadmap construction algorithm for mobile robot path planning using skeleton maps. *Advanced Robotics.* 21**,** 51-63.

CETTO, J. A., FILIPE, J. & FERRIER, J.-L. (eds.). YANG, J., DYMOND, P. and JENKIN, M. 2011. Exploiting Hierarchical Probabilistic Motion Planning for Robot Reachable Workspace Estimation. In: Informatics in Control Automation and Robotics. Springer Berlin Heidelberg, 229-241.

YANG, L. and LAVALLE, S. M. (2003) The sampling-based neighborhood graph: An approach to computing and executing feedback motion strategies. *IEEE Trans. on Robotics and Automation.* 20**,** 419– 432.

YANG, L. and LAVALLE, S. M. (2004) The sampling-based neighborhood graph: an approach to computing and executing feedback motion strategies. *Robotics and Automation, IEEE Transactions on.* 20**,** 419-432.

ZEROC INC. 2011. *ZeroC Ice* [Online]. Available: http://www.zeroc.com [Accessed 27.6.2013].

ZLAJPAH, L. (1999) On-line obstacle avoidance control for redundant robots using tactile sensors. *IASTED Int. Conf. Control and Applications.* 533-538.

A. List of Publications

The following is a list of publications produced by the author during the course of the investigations outlined in this thesis.

1. KOHRT, C., PIPE, A., SCHIEDERMEIER, G., STAMP, R. and KIELY, J. 2012. A Flexible Model Driven Robotics Development Framework. *The 43rd International Symposium on Robotics (ISR).* Taipei, Taiwan.
   This publication is based on investigations accomplished as part of objective five.

   Abstract - A flexible robotics development framework has been established to allow rapid development of high-performance real-time applications from distributed software components. The framework interconnects software components and hardware devices as well as specialized third party software applications to allow integration into the communication system with ease. A model driven approach has been chosen in order to raise the usability of the framework using a visual modeling language. A communication middleware has been evaluated for the interconnection of the components. This paper introduces the required tools, proposes a model driven development framework for robotic applications and provides experiences in the development and use of such frameworks.

2. KOHRT, C., PIPE, A. G., KIELY, J., STAMP, R. and SCHIEDERMEIER, G. 2012. A cell based voronoi roadmap for motion planning of articulated robots using movement primitives. *International Conference on Robotics and Biomimetics* (ROBIO). Guangzhou, China: IEEE.
   This publication is based on investigations accomplished as part of the objectives two and four.

   Abstract - The manufacturing industry today is still focused on the maximization of production. A possible development able to support the global achievement of this goal is the implementation of a new support system for trajectory planning, specific for industrial robots. This paper describes the trajectory-planning algorithm, able to generate trajectories manageable by human operators, consisting of linear and circular movement primitives. First, the world model and a topology preserving

roadmap are stored in a probabilistic occupancy octree by applying a cell extension based algorithm. Successively, the roadmap is constructed within the free reachable joint space maximizing the clearance to the obstacles. A search algorithm is applied on robot configuration positions within the roadmap to identify a path avoiding static obstacles. Finally, the resulting path is converted through an elastic net algorithm into a robot trajectory, which consists of canonical ordered linear and circular movement primitives. The algorithm is demonstrated in a real industrial manipulator context.

3. KOHRT, C., PIPE, A., SCHIEDERMEIER, G., STAMP, R. and KIELY, J. 2011. An Online Robot Trajectory Planning and Programming Support System for Industrial Use. *Journal of Robotics and Computer-Integrated Manufacturing.*
   This publication is based on investigations accomplished as part of objective one.

   Abstract - The manufacturing industry today is still looking for enhancement of their production. Programming of articulated production robots is a major area for improvement. Today, offline simulation modified by manual programming is widely used to reduce production downtimes but requires financial investments in terms of additional personnel and equipment costs. The requirements have been evaluated considering modern manufacturing aspects and a new online robot trajectory planning and programming support system is presented for industrial use. The proposed methodology is executed solely online, rendering offline simulation obsolete and thereby reduces costs. To enable this system, a new cell-based Voronoi generation algorithm, together with a trajectory planner, is introduced. The robot trajectories so achieved are comparable to manually programmed robot programs. The results for a Mitsubishi RV-2AJ five axis industrial robot are presented.

4. KOHRT, C., PIPE, A., SCHIEDERMEIER, G., STAMP, R. and KIELY, J. 2008. A robot manipulator communications and control framework. *Proc. IEEE Int. Conf. on Mechatronics and Automation ICMA.*
   This publication is based on investigations accomplished as part of objective three.

Abstract - The use of industrial scale experimental machinery robot systems such as the Mitsubishi RV-2AJ manipulator in research to experimentally prove new theories is a great opportunity. The robot manipulator communications and control framework written in Java simplifies the use of Mitsubishi robot manipulators and provides communication between a personal computer and the robot. Connecting a personal computer leads to different communication modes each with specific properties, explained in detail. Integration of the framework for scientific use is shown in conjunction with a graphical user-interface and within Simulink as a Simulink block. An example application for assisted robot program generation is described.

5. KOHRT, C., SCHIEDERMEIER, G., PIPE, A. G., KIELY, J. and STAMP, R. 2006. Nonholonomic Motion Planning by Means of Particles. *International Mechatronics and Automation Conference.* Luoyang, China: IEEE.
   This publication is based on investigations accomplished as part of objective four.

   Abstract - In this article a new approach to planning of a nonholonomic motion is presented. A flexible, intelligent planner based on a static map and the topology of the robot's environment has been developed. The approach uses 'particles' to construct automatically a path between two given locations. The generated path is a smooth trajectory, where the length of the path is kept at a minimum and obstacles are avoided. This concept applies to robots meeting the restrictions of a Dubin's car (nonholonomic robot that can only move forward). After the basic concepts of the approach has been described, simulations will be presented.

6. KOHRT, C., ROJKO, R., REICHER, T. and SCHIEDERMEIER, G. 2006. With Model Based Design To Productive Solutions Professional GUIs For Simulink By Utilizing The Java SWT Library. *WEKA FACHZEITSCHR.-VERLAG, KFZ-Elektronik.*
   This publication is based on investigations accomplished as part of the objectives four and five.

   Abstract - The Model-Based Design (MBD) approach is a widely used method to solve sci-entific engineering challenges [1]. Matlab/Simulink as a representative of

MBD is a tool capable of exploiting the advantageous aspects of a graphical user inter-face (GUI). The latter is created with a tool named GUIDE, which is shipped with the Matlab/Simulink software. Unfortunately, user interfaces created with GUIDE have some drawbacks. Thus, new approaches are needed to overcome these draw-backs to improve the design of the GUI. It is surprising, that the Java SWT library (Standard Widget Toolkit) is not used for such user interfaces. Although not sup-ported by Mathworks, this article compares the features of an SWT based GUI to the GUIDE, explains the practical implementation of SWT GUIs by examples and gives an outlook to the wide field of applications taking benefit.

B. Materials & Equipment

The proposed support system is applied on a 5-axis industrial-scale, articulated Mitsubishi RV-2AJ robot with an additional Ethernet card installed. It is a nonlinear system with five rotary joints. The robot is equipped with the Mitsubishi CR1 controller and a teach pendant. The main areas of the robot are assembly, manufacture, pick & place and handling tasks. Communication between this system and a personal computer is possible (Kohrt *et al.*, 2008); the commercial viability has already been demonstrated (Mitsubishi-Electric, 2008). The equipment is shown in Figure 134.

Figure 134: Devices overview.

The robot manipulator communications and control framework is executed on the personal computer, which has an Ethernet and serial port connection to the robot controller. The teach pendant and the robot are connected to the controller. The vision system and the pointing device are plugged in to the personal computer. The framework is verified with a visual servo-control application including collision detection and Matlab/Simulink integration.

The Industrial Robot Manipulator Mitsubishi RV-2AJ

A Mitsubishi RV-2AJ robot as shown in Figure 135 is used with an additional Ethernet card installed throughout this work. It is a typical industrial robot widely used. The robot is installed at the lab of the Computer Sciences Department at the University of Applied

Sciences Landshut, Germany. The robot is equipped with the Mitsubishi CR1 controller and a teach pendant.

Figure 135: The Robot manipulator Mitsubishi RV-2AJ.

These robots are advanced, but mature and industrially proven machines; their commercial viability has already been demonstrated in the manufacture of car sub-assemblies, semiconductor memories and other industrial/consumer goods within companies such as Jaguar and Audi. The main areas of application are:

- Assembling / manufacturing,
- handling in laboratories,
- semiconductor manufacturing and monitoring,
- blank manufacturing and monitoring,
- pick and place and
- robot training.

The robot type RV-2AJ is an articulated robot (R) that operates vertically (V) with maximum payload of 2 kg. It is the Mitsubishi robot series S with 5 joints. Data of the robot arm RV-2AJ:

- Repeatability 0.02 mm
- Max. payload 2 kg
- Max. velocity 2,100 mm/s
- Reach 410 mm

- robot weight 17 kg

The robot has the following functions:

- Compliance Control function
- Multitasking operating system
- Load-based acceleration optimization
- Individual axis torque monitoring
- Sensor less crash detection
- Control functions for additional axes
- IP65 protection rating (axes 4-6)

The Controller

The controller Mitsubishi CR1 Mitsubishi CR1 is a New Architecture Robot Controller (NARC).

Figure 136: CR1 Controller.

Standard functions of the robot controller are:

- Easy-to-learn control instruction set,
- axis, linear and three dimensional circular interpolation,
- subroutines,
- execute up to 32 programs simultaneously,
- integrated math functions,
- integrated palletizing functions,
- interrupt handling,
- compliance Control function and
- tracking (conveyor belt synchronization).

The Mobile Robot Robotino

The mobile robot Robotino in Figure 137, produced by the company Festo (Festo, 2011), is employed as an experimental framework to research on path planning algorithm development. The Robotino robot is featured with different sensors like a camera and twelve infrared proximity sensors, which have been utilized for sensor fusion development. It provides a Java robot control framework that can directly be employed. The provided robot control framework supports wireless local area network connections to command the robot and to obtain sensor information.

Figure 137: A Robotino robot from the company Festo.

C. Robot Control

This appendix summarizes the protocol format of the Mitsubishi CR1 Controller for transmitting and receiving.

Controller Parameters

Table 14 has been used to set up the controller for each communication mode.

Parameter name	Details	Number of elements	Default value	Controller commnuni-cation mode	Data link mode	Real-time external control mode
NETIP	IP address of robot controller	Character string 1	192.168.0.1	X	X	X
NETMSK	Sub-net-mask	Character string 1	255.255.255.255	X	X	X
NETPORT	Port No. Range 0 to 32767 For function expansion (reserved) Correspond to OPT 11-19 of COMDEV ---------- (OPT11) (OPT12) (OPT13) (OPT14) (OPT15) (OPT16) (OPT17) (OPT18) (OPT19)	Numerical value 10	 10000, 10001, 10002, 10003, 10004, 10005, 10006, 10007, 10008, 10009	X	X	X
CPRCE11 CPRCE12 CPRCE13 CPRCE14 CPRCE15 CPRCE16 CPRCE17 CPRCE18 CPRCE19	Protocol 0: No-procedure 1: Procedure 2: Data link (1: Procedure has currently no function.) Correspond to OPT 11-19 of COMDEV (OPT11) (OPT12) (OPT13) (OPT14) (OPT15) (OPT16) (OPT17) (OPT18) (OPT19)	Numerical value 9	 0, 0, 0, 0, 0, 0, 0, 0, 0	-	X	-
COMDEV	Definition of device corresponding to COM1: to 8: Definition of device corresponding to COM1: Definition of device corresponding to COM2: Definition of device corresponding to COM3: Definition of device corresponding to COM4: Definition of device corresponding to COM5: Definition of device corresponding to COM6: Definition of device corresponding to COM7: Definition of device corresponding to COM8: When the data link is applied, setting is necessary. OPT11 to OPT19 are allocated. Here, RS-232C of the controller is previously allocated to COM1: .	Character string 8	 RS232C, , , , , , , ,	-	X	-

Table 14: Controller communication mode set up.

NETMODE	Server designation (1: Server, 0: Client) (OPT11) (OPT12) (OPT13) (OPT14) (OPT15) (OPT16) (OPT17) (OPT18) (OPT19)	Numerical value 9	 1, 1, 1, 1, 1, 1, 1, 1, 1	-	X	-
NETHSTIP	The IP address of the data communication destination server. * It is valid if specified as the client by NETMODE only. (OPT11) (OPT12) (OPT13) (OPT14) (OPT15) (OPT16) (OPT17) (OPT18) (OPT19)	Character string 9 .	 192.168.0.2, 192.168.0.3, 192.168.0.4, 192.168.0.5, 192.168.0.6, 192.168.0.7, 192.168.0.8, 192.168.0.9, 192.168.0.10	-	X	-
MXTTOUT	Timeout time for executing real-time external control command (Multiple of 7.1msec, Set -1 to disable timeout)	Value 1 (0-32767)	-1	-	-	X

Table 15: Controller communication mode set up (continued).

The default parameters for the Ethernet card are:

- COM1
- 9600 baud
- 8 data bits
- even parity
- stop bits
- DTR on
- RTS/CTS on
- XON/XOFF off

Controller Protocol Format

Transmit data

[< Robot No.>];[< Slot No>];<Command> <Argument>

< Robot No.>

The robot number to be operated is specified to 1, 2 or 3. It is possible to omit it. The standard value is 1.

< Slot No>

The slot number to be operated can be specified to 1 - 33. Parameter 'TASKMAX' is a number of task slots used by the multitask program.

When the program is edited from the PC, the edit slot is used. The slot number of the edit slot is parameter TASKMAX+1. In this case, because an initial value of TASKMAX is 8, the number of the edit slot is 9. It is possible to omit it. The standard value is 1.

<Command> <Argument>

These arguments are command specific.

Receive data

Commands	Contents
QoK****	Normal status
Qok****	Error status
QeR****	Illegal data.(with error number (4 digit))
Qer****	Error status and illegal data. (with error number (4 digit))

Table 16: Receive command pattern.

QoK<Answer>

This argument differs in each command. Refer to the explanation of each command.

Qok<Error status>

This argument replies the error number when the command may not be executed. Refer to the troubleshooting manual of the robot for the description of the error number.

QeR<Illegal data with 4-digit error number>

This argument replies the error number when the command may not be executed. Refer to the troubleshooting manual of the robot for the description of the error number.

Qer<Error status and illegal data with 4-digit error number>

This argument replies the error number when the command may not be executed. Refer to the troubleshooting manual of the robot for the description of the error number.

D. Denavit-Hartenberg-Parameter

The DH-parameters are the standard method used to define the direct kinematics of a manipulator (Paul, 1981). A robot model is described with four DH-parameters for each rotational or translational joint. The joint axis for a rotational or translational degree-of-freedom is always defined by the z-axis of the coordinate system. The transformation T_i^{i-1} defined with the DH-parameter is a combination of the following four successive transformations:

- Rotation around axis z^{i-1} by the angle θ_i
- Translation along axis z^{i-1} by the distance d_i
- Translation along axis x^i by the distance a_i
- Rotation around axis x^i by the angle α_i

The parameter θ_i for a rotational joint, and d_i is non-constant for a translational joint. The final transformation matrix that depends on the four parameters is as follows:

$$T_i^{i-1}(\theta_i, d_i, a_i, \alpha_i) = \\ = {}^{i-1}A_i = \begin{pmatrix} \cos(\theta_i) & -\cos(\alpha_i)\cdot\sin(\theta_i) & \sin(\alpha_i)\cdot\sin(\theta_i) & a_i\cdot\cos(\theta_i) \\ \sin(\theta_i) & \cos(\alpha_i)\cdot\cos(\theta_i) & -\sin(\alpha_i)\cdot\cos(\theta_i) & a_i\cdot\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{117}$$

The DH-parameters are defined by construction rules for the joint coordinate system and their relations.

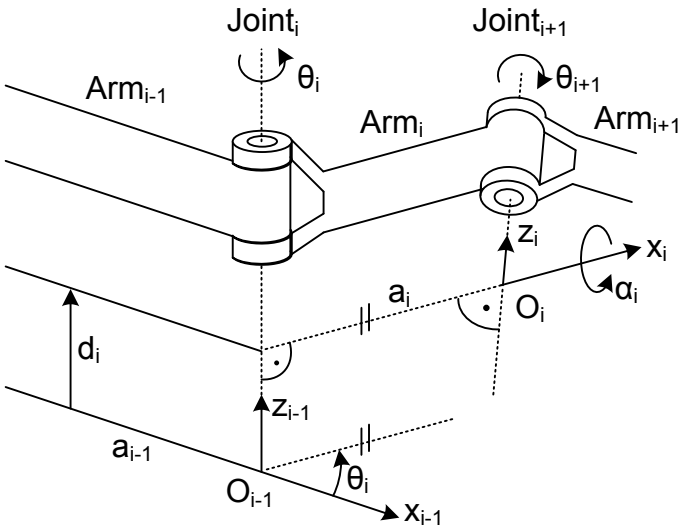


Figure 138: Constructed coordinate systems.

Basic rules for constructing the coordinate systems and the DH-parameters:

- Arm i is the connection between the i-th and the $(i+1)$-th articulation.
- The coordinate system O_i is dedicated to the i-th arm.
- The coordinate system O_0 is the fixed basic-coordinate system.
- The z_i- axis is applied along the movement-axis of the $(i+1)$-th articulation.
- The x_i- axis is the normal to the z_{i-1} axis and is pointed away from it.
- The y_i- axis is defined such that a legal framework is produced.

Special Cases:

- z_i- axis and z_{i-1} axis cross each other.
- There are two possibilities for setting x_i to be as perpendicular on the z_i- or z_{i-1}- axis. Either of them may be chosen.
- z_i- axis and z_{i-1} axis are parallel.
  The origin O_i may be arbitrary.
- z_i- axis and z_{i-1} axis cross each other.
  Both the x_i axis and the origin O_i may be arbitrary.

The coordinate system's attitude is described as follows:

- d_i Distance along the z_{i-1} axis between the origin O_{i-1} and the intercept of the z_{i-1} axis and the x_i axis.
- θ_i Articulation angle around the z_{i-1} axis from the x_{i-1} axis to the projection of the x_i axis towards the x_{i-1}, y_{i-1} plane.
- a_i Shortens the connection between z_{i-1} axis and z_i axis.
- α_i Angle of rotation around the x_i axis which levels the z_{i-1} axis with the z_i axis.

E. Execution Model

An execution model consists of a set of rules that define the system behaviour. The execution model described in the ROOM standard (Selic, 1996a, Selic, 1996b, Selic *et al.*, 1994) was employed. ROOM is a visual modelling language with formal semantics and it was developed by ObjecTime. It is optimized for the specification, visualization, documentation and automation of the construction of complex, event-driven and potentially distributed real-time systems. The actor is the basic building block used to describe the structural design of a distributed system.

Internal Structure of an Actor

Simple functionality can be realized by an actor, which has no inner structure. Actors that are more complex have an internal structure, which is a network of collaborating sub-actors joined by connectors. Therefore, the actor may delegate complex functionalities to sub-actors. Both the sub-actors and their connections are hidden from external observers. Sub-actors are actors in their own right, and can themselves be further decomposed into sub-actors. This type of decomposition can be carried on to any depth necessary, enabling the modelling of arbitrarily complex structures using only this basic set of modelling constructs.

Ports of an Actor

An actor communicates with its environment only via ports, as described in Figure 139. Ports are used to define dedicated points of interaction between the actor and its environment. Ports may either provide or require a service, which is specified by an interface. By connecting the ports of several actors, an interaction flow via messages can be established between them. The service provided by a port may either be realized by the actor itself (EndPort) or delegated to the port of a contained actor (RelayPort). The port of the contained actor has to provide the same service as the port of the (outer) actor. Communication via ports may be either synchronous or asynchronous. However, synchronous communication limits the ability of the port to deploy the actor. Communication or delegation between actors is allowed only via ports.

By employing ports, the actors can be more easily distributed on different nodes, and the role of an actor is clearly defined and better encapsulation (to interact with it, only the port is required, and not the type of the actor) has been accomplished..

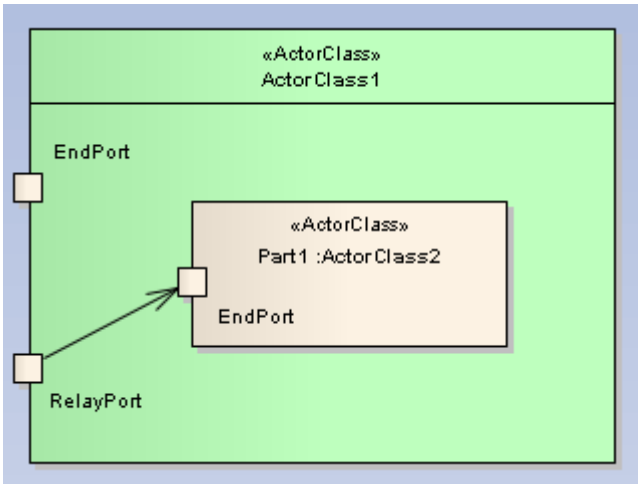


Figure 139: Actor with ports.

Behaviour of an Actor

An actor defines behaviour that is specified by operations and optionally by a UniMod state machine. A state machine represents one part of the implementation of the actor, which is hidden from external observers. Operations can be either hidden or public, based on their usage. Direct synchronous calls can be executed on that actor operation when the method is public. Because the operation is called within the thread of the calling actor, the call has to be thread-safe. This has to be verified by the user. If the method is called synchronously over the ports, no additional synchronization is required because it is handled by the execution model. A complex actor may combine the state machine with an internal network of collaborating sub-actors that are joined by connectors.

Message Service

The message service manages communication between processing units for real-time applications either on single or distributed processors. It is a middleware between software components that communicate synchronously or asynchronously with other message services or actors.

For distributed real-time applications, efficient system communication is needed. The distributed application may consist of components deployed on different processing units. The processing unit may be a general-purpose processor (GPP), digital signal processor (DSP) or a FPGA. Each processing unit may have its own special system architecture that influences different processes, for example the handling of threads. Threads can be either pre-emptive or co-operative, depending on the processing unit architecture. The deployed application may use one or more threads, and it may be programmed in languages such as

C, C++ or Java. The introduced Message Service has a runtime library for each processing unit/programming language combination that is used by the software components to communicate with other components in the same thread, between threads on the same processing unit or between processing units either synchronously or asynchronously.

Message Service Communication Types

Because different protocols can be used with Ethernets, depending on the requirements of the system, there should be flexibility with respect to the choice of the appropriate protocol. In addition, each processing unit may have different communication methods for inter-thread and intra-thread communication. This significantly affects the architecture of the Message Service, which has to be sufficiently abstract that it can be utilized by any processing unit and any programming language, and sufficiently concrete to fulfil the requirements of speed, code size and memory consumption that exist when using the processing unit and language specific methods. The following aspects have to be considered in the middleware:

- Node
- Thread
- Programming language
- Java Runtime Environment
- Operating system (Windows, Linux, PowerPC and Integrity)
- Synchronous/asynchronous method calls

A comparison of the middlewares that were evaluated to implement the logical communication framework is given in Section 9.2.

Logical Communication Framework

Figure 140 shows the logical communication framework in its complete stage of expansion. Each processing unit has at least one thread with a message service. A thread uses pre-emptive scheduling with no memory protection. Therefore, there is exactly one message service in a thread and a thread may have one or more actors. The message service provides the middleware for communicating within a thread, between the threads on one node and between threads on different nodes.

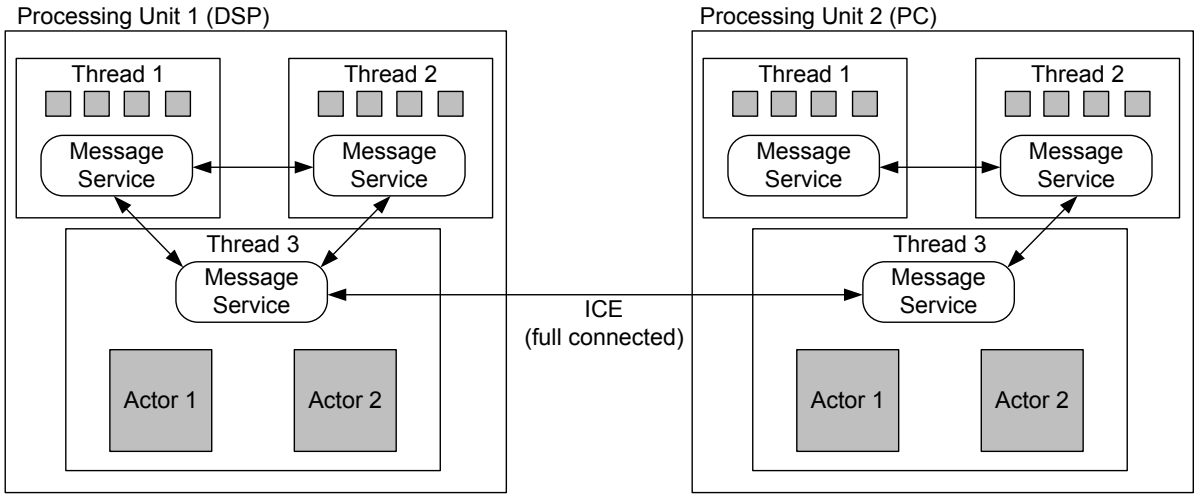


Figure 140: Communication overview.

Running Loop

Within a thread, the endless loop is controlled by a while loop which has a blocking call to read from one or more external queues. 'Blocking' means that it waits for a message on the external queue. If there are no messages, the thread sleeps and does not consume processing time.

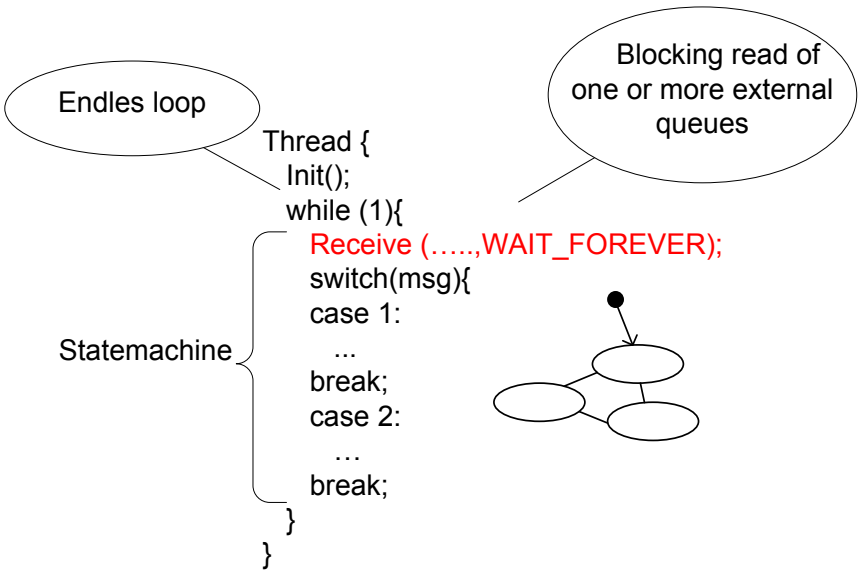


Figure 141: Running loop of a thread.

Scheduling

The scheduling is illustrated in Figure 142. First, internal events are processed, followed by external events. This execution model provides a 'run to completion' feature in order to first complete the internal state machine before processing external events.

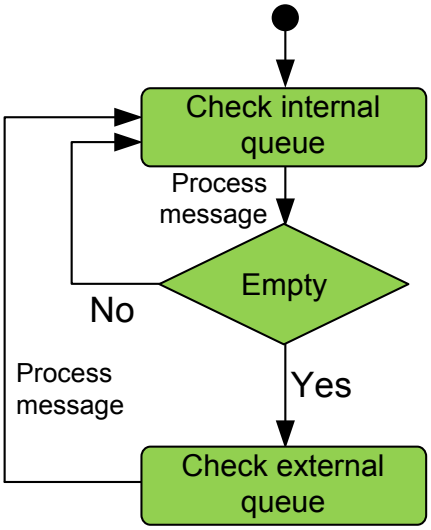


Figure 142: Event scheduling.

Thread Priorities

The priorities of threads may be settable on a processing unit. Within the execution model, thread priorities are used to allow more control over the runtime behaviour of the system. Threads with higher priorities are always executed first, and then the processing time is given to other threads. This complies with the pre-emptive threading model.

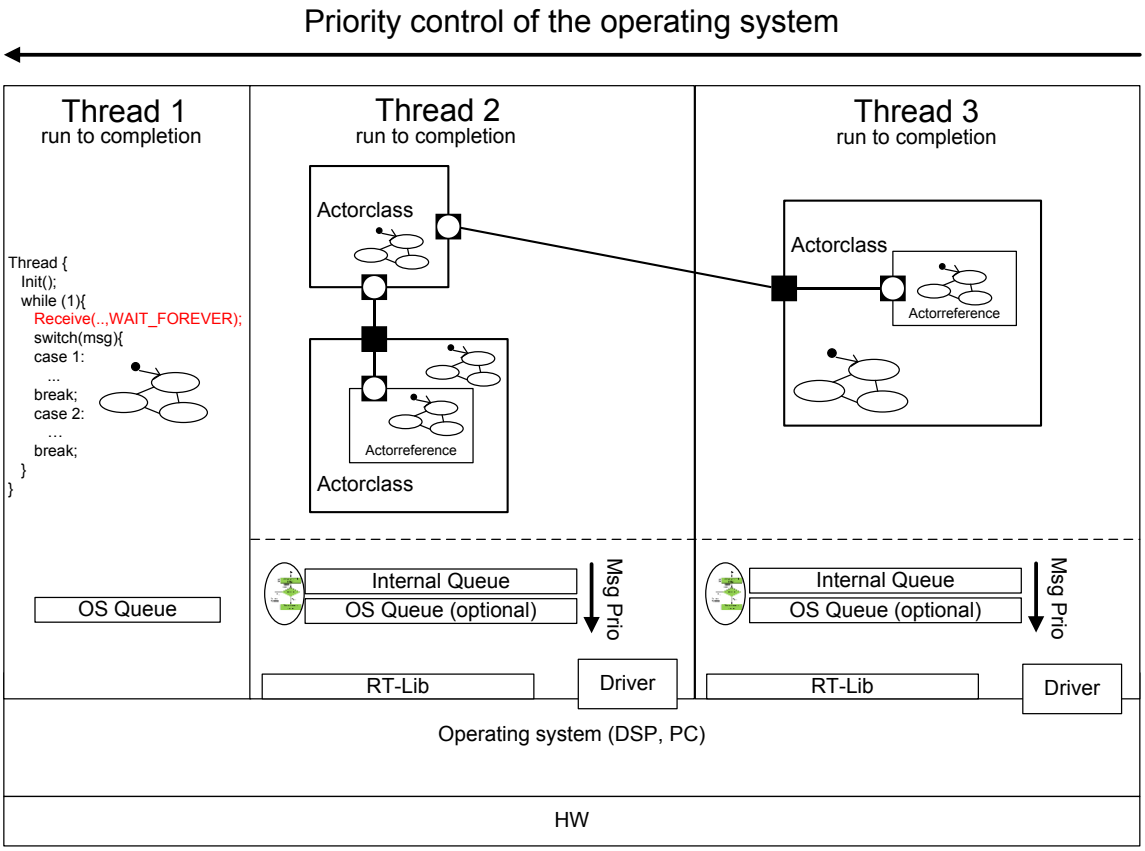


Figure 143: Thread priorities.

Hardware events

Hardware events, for example from the joystick, have to be transformed into a message format which conforms to the execution model. This is realized in interrupt routines. In Windows, the Java programming language has a hardware abstraction layer, which usually uses a listener concept. The listener implementation is then used to transform the event to a message. The message is processed either in an interrupt routine or in an event listener, which sends the message to a predefined port of an actor (see Figure 144).

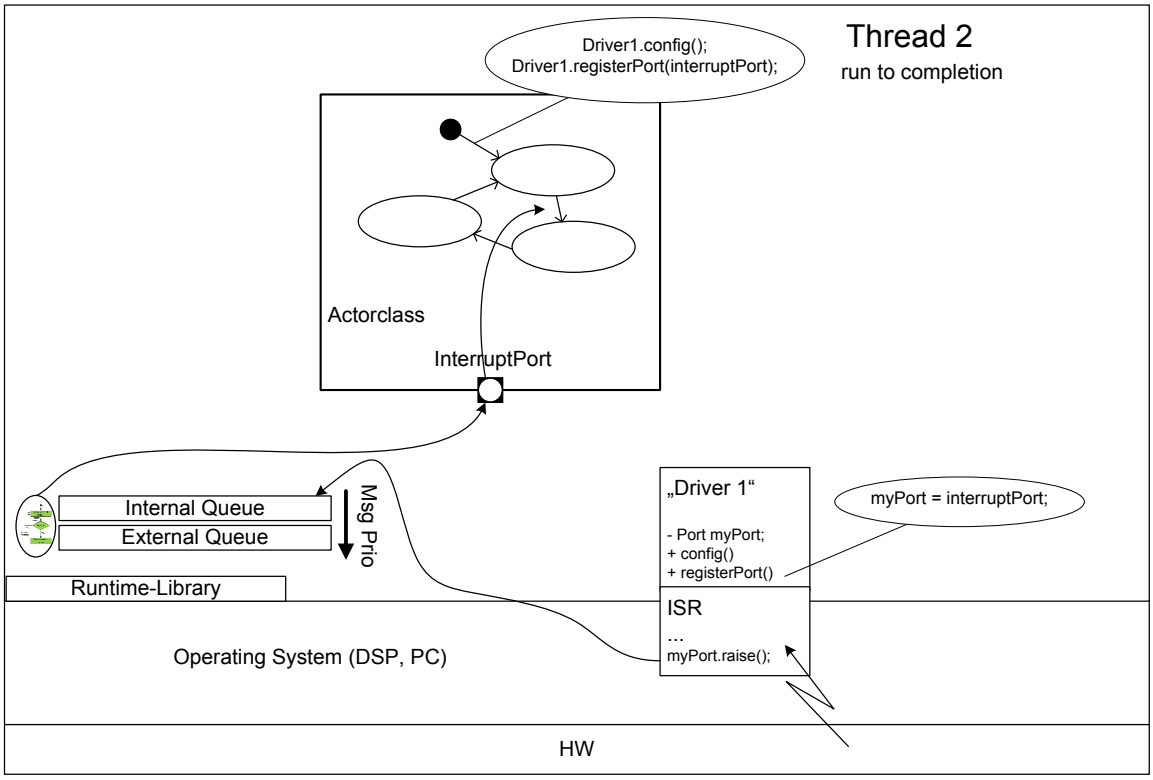


Figure 144: Interrupt handling.

Message concept

Messages are passed instead of calling methods directly. These messages have the required information to be delivered to the receiver. This makes the system more generic, and message handling is executed only within the sender, receiver and the message service. The message service routes the messages to the right receiver. A message service runs in its own thread and it is identified by the IP address and the port of the host.

Thread-Internal-Communication

A thread contains one or more actors that may communicate either synchronously or asynchronously. For communication, messages are sent to the message service, which routes the message to the receiver port. Only programming language specifics are utilized.

Inter-Thread-Communication

Between two different threads on one node, the message services have to be able to exchange messages, and this is accomplished by adding messages to the external message queue of the receiver message service. This approach therefore considers inter thread timing.

If the two actors are written in different programming languages, marshalling is required to convert messages, such as the conversion from the C++ to the Java format and vice versa. The evaluated internet communication engine (ICE) middleware is capable of accomplishing this.

Inter-Node-Communication

The most complex work was carried out for inter node communication, where the nodes have to be capable of connecting to other nodes. CORBA is a famous communication middleware that was developed for such cases. However, since CORBA is quite complex, the ICE middleware was chosen. When compared to CORBA, it was found to be simpler to use and faster, although it is not standardized.

Message Sending Examples

The communication between actors is controlled by the message service. The external communication between actors passes through the ports of the deployable actors, but messages are also passed within an actor and may execute a self-trigger to its own state machine. The message service defines communication mechanisms, which differentiate between internal, external, synchronous and asynchronous communication. The message data types are defined in the ICE middleware project. Table 17 shows the parameters of the messages. Below, some examples are used to explain internal and external message sending.

Parameter	Description
Signal	The signal name is the minimal information that has to be sent within a message and is provided by the ports.
Message Data	Additional data may be sent with the message. This data may be structured individually.

Table 17: Message parameters.

Internal messages are used for the communication within the actor. These messages may be sent from anywhere within this actor. The construction of a simple internal message is shown in Listing 14. A new internal message with message data is created using the signal ‘StartIn’. It is sufficient to know the message service that is available for each actor. The message is sent directly to the Finite-State-Machine without using any ports to the internal queue.

```
msgService.sendMessage(_JoystickDeviceControlPort._StartIn, msgData);
```

Listing 14: Simple internal message.

External messages are used for communication with other deployable actors. An example of a broadcast external message is shown in Listing 15. This message is sent to all the deployable actors that are connected to the port. In this example, the signal ‘StartIn’ is sent. Moreover, the sending port is selected through port definitions. This asynchronous message does not contain message data.

```
_JoystickDevicePort.sendBroadcast(_JoystickDeviceControlPort._StartIn);
```

Listing 15: Broadcasting external message.

Listing 16 shows an external message sent both asynchronously and synchronously through a port. The receiver is defined within the Enterprise Architect UML model and does not need to be specified. This allows the re-use of actors.

```
_JoystickDevicePort.sendMessage(_JoystickDeviceControlPort._DeviceInitFailedOu
t);
_JoystickDevicePort.invokeMessage(_JoystickDevicePort._SetStatus);
```

Listing 16: Pointed external message.

F. Kohonen Map

The neural network algorithm performs a search for each data input vector to find the best matching unit w_{bmu}, which is the neuron with the minimum distance to the input vector. w_{bmu} and its neighbouring neurons w_S are adapted by learning rules and update their weights. The network was designed as a two-layered network consisting of an input layer of neurons that are directly and entirely connected to an output layer. The output layer was organized as a two-dimensional grid, as depicted in Figure 145. w_S is the weight vector associated to the neuron placed at position s on the grid.

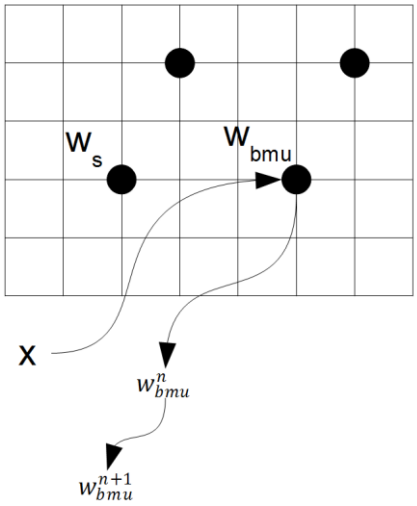


Figure 145: The Kohonen Map.

The network is trained by unsupervised learning on an input vector $x = \{\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_\mathrm{n}\}$. For each vector x presented to the input layer, a competition between the neurons takes place. Each neuron calculates the distance $d(x, w_S)$.

(118) $$d(x, w_S) = \|x - w_S\|^2.$$

The neuron w_{bmu} with the closest weight vector to x is the best matching unit of the competition.

(119) $$w_{bmu} = \min\big(d(\mathrm{x}, w_S)\big)$$

w_{bmu} learns the input vector by moving closer to it.

(120) $$w_{bmu}^{n+1} = w_{bmu}^{n} + c(t)h(s, w_{bmu}^{n})(x - w_{bmu}^{n})$$

Figure 145 illustrates the weight change process of neuron w_{bmu} in the original input space. In equation (120), $c(t)$ is the learning rate, a real parameter that decreases linearly with the learning process with equation (121).

$$c(t) = c(0)(1 - tT^{-1}) \quad (121)$$

$\mathrm{h}(\mathrm{s}, \mathrm{w}_{\mathrm{bmu}}^{\mathrm{n}})$ defines the Gaussian or Mexican hut kernel weight of $\|\mathrm{w}_{\mathrm{bmu}}^{\mathrm{n}} - \mathrm{s}\|$. The learning step is also extended to the neighbours of the winner neuron $\mathrm{w}_{\mathrm{bmu}}$. The neighbours of $\mathrm{w}_{\mathrm{bmu}}$ are the output elements whose distance to the $\mathrm{w}_{\mathrm{bmu}}$, as measured on the grid, is not greater than the decreasing neighbourhood parameter over time.

G. Node Movement Calculation

The two-dimensional case is calculated in equation (122) with the illustration in Figure 146, where $\overrightarrow{rv_g}$ is the movement vector, $\vec{v}$ is the movement result vector and $\overrightarrow{rv}$ is the obstacle node connection vector.

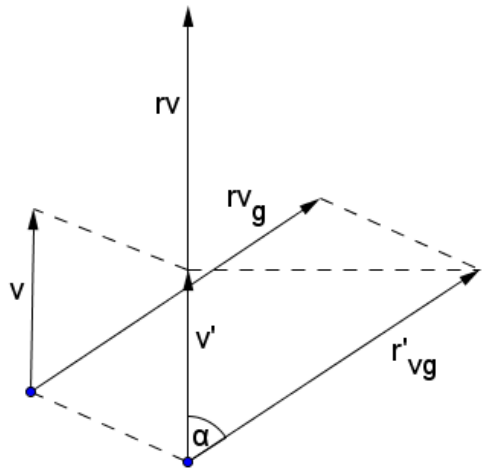


Figure 146: Vectors of movement.

$$\vec{v} = \left(\left|\overrightarrow{rv_g}\right| \cdot \left|\overrightarrow{rv}\right| \cdot \cos\alpha\right) \cdot \overrightarrow{rv}^0 \quad with\ 0 \leq \alpha \leq \pi\ (only\ in\ 2\ dimensional\ case) \tag{122}$$

For the three-dimensional case, the collision is between a vector and a polygon. A vector that collides with a polygon must be recalculated so that its direction is parallel to the polygon surface.

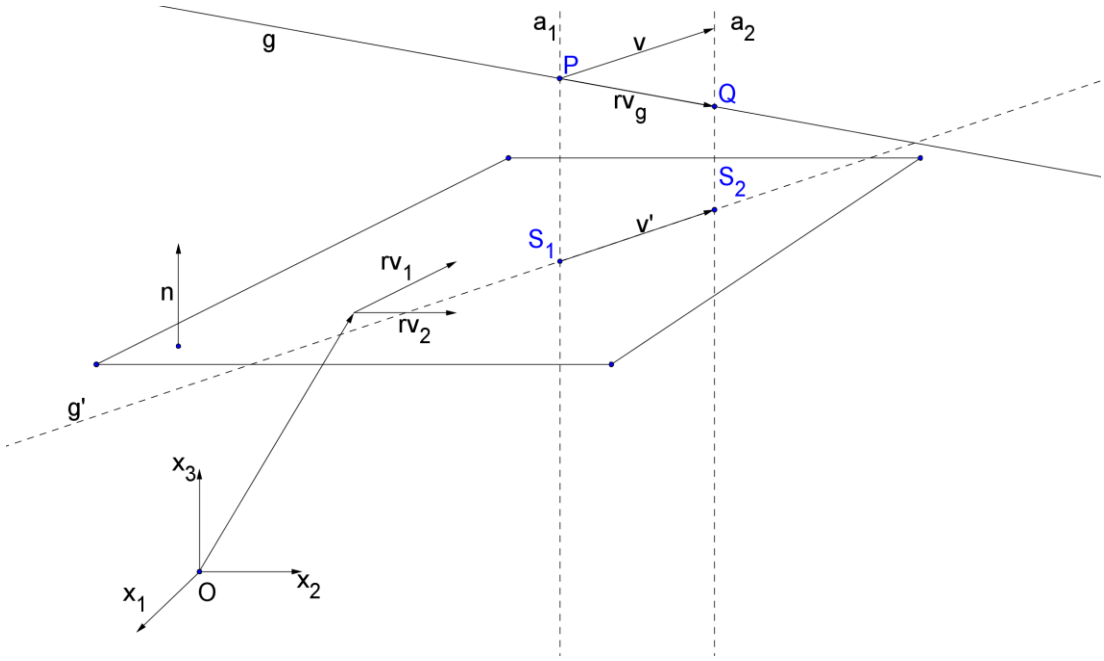


Figure 147: Recalculation of the node movement vector.

Below, the projection of a vector onto a polygon is calculated. The formulas for the parametric form of a layer and a straight line are given in equation (123) and (124), respectively.

(123) $x = \overrightarrow{ap} + \lambda \cdot \overrightarrow{rv_1} + \mu \cdot \overrightarrow{rv_2}$

(124) $x = \vec{p} + \omega \cdot \overrightarrow{rv_g}$

where $\vec{n}$ is the normal vector of the layer. Two helping straight lines are defined in (125) and (126). The intersection of the helping straight lines with the layer are named S_1 and S_2.

(125) $x = \vec{p} + \sigma \cdot \vec{n}$

(126) $x = \vec{q} + \eta \cdot \vec{n}$

Point Q is calculated in (127).

(127) $\vec{q} = \vec{p} + 1 \cdot \overrightarrow{rv_g}$

The normal form of the layer E is given in (128) and (129).

(128) $\vec{n} = \overrightarrow{rv_1} \times \overrightarrow{rv_2}$

(129) $\vec{n} \cdot (x - \overrightarrow{ap}) = 0$

The intersection S_1 of the auxiliary straight line 1 given in (125), and the layer given in (129) is calculated using the formulas (130) and (131):

(130) $\vec{n} \cdot (\vec{p} + \sigma \cdot \vec{n} - \overrightarrow{ap}) = 0 \Rightarrow \vec{n} \cdot (\vec{p} - \overrightarrow{ap}) + \sigma \cdot \vec{n}^2 = 0 \Rightarrow \sigma = \frac{-\vec{n} \cdot (\vec{p} - \overrightarrow{ap})}{\vec{n}^2}$

(131) $\overrightarrow{S_1} = \vec{P} + 2 \cdot \sigma \cdot \vec{n}$

The intersection S_2 of the auxiliary straight line 2 given by (126) and the layer given in (129) is calculated in the formulas (132) and (133).

(132) $\vec{n} \cdot (\vec{q} + \eta \cdot \vec{n} - \overrightarrow{ap}) = 0 \Rightarrow \vec{n} \cdot (\vec{q} - \overrightarrow{ap}) + \eta \cdot \vec{n}^2 = 0 \Rightarrow \eta = \frac{-\vec{n} \cdot (\vec{q} - \overrightarrow{ap})}{\vec{n}^2}$

(133) $\overrightarrow{S_2} = \vec{q} + 2 \cdot \eta \cdot \vec{n}$

Finally, the resulting vector $\vec{v}$ is calculated by (134):

(134) $\vec{v} = \vec{p} + \overrightarrow{OS_2} - \overrightarrow{OS_1}$

Finally, the node moves in the direction of $\vec{v}$.

H. Plugin Manager

The Java side 'Plugin Manager' component includes four functions, and is a Java component that is able to call C/C++ functions of DLLs, which allowed source code reuse.

The *init* function is responsible for setting the library path and loading the *PPA_Plugin_Manager.dll*, which is the gateway between Java and native libraries. Additional methods are provided to execute a library function call, and the choice of function to be used depends on the expected return type. '*Invoke(...)*' is used when a single value is expected (e.g. *int*). If an array or a two-dimensional array is expected '*invokeArray(...)*' and '*invoke2DArray(...)*' are used, respectively. Each of these functions has almost the same parameters. Listing 17 shows the 'invoke(…)' method.

```
public static Object invoke(PPA_TYPES returnType, String dllName, String
methodName, LinkedList<Object> params, Object jobj)
```

Listing 17: Invoke method from Plugin Manager.

The parameter returnType shows which type of data is expected as return value. The possibilities are *VOID*, *INT*, *FLOAT*, *DOUBLE*, *STRING*, *INT_1D*, *FLOAT_1D*, *DOUBLE_1D*, *INT_2D*, *DOUBLE_2D*, *FLOAT_2D* and *BOOLEAN*. Each of the functions expects three parameters. The first and the second ones are the name of the library file to load and the function name to be called (*String dllName, String function*), respectively. The third parameter is a linked list from the Java collection framework (*LinkedList<Object> params*), and contains the parameters that are passed to the library function. The generic type is '*Object*', because the list may contain different variable types. To allow the native method to do a call back, an instance of the calling class is passed as the final argument (*Object jobj*).

The Plugin Manager (*PPA_Plugin_Manager.dll*) buffers loaded functions to increase performance. The two functions *addDll(String dllName)* and *releaseDll(String dllName)* are responsible for loading and unloading libraries, respectively. Each of them takes a *String* as the parameter that contains the path to the library.

A function call is executed as follows. First, a linked list with arguments is created, and second, the library file is loaded. After these two steps, one or more functions from the library can be executed. Finally, the library file is again unloaded to de-allocate the used recourses.

The ‘dyncall’ (Adler and Philipp, 2011) library provides a clear and portable C application interface to dynamically issue calls to foreign code using small call kernels written in assembler. It was utilized within the plugin manager.

JNI Usage

The Java Native Interface (JNI) is a programming framework that allows Java code running in a Java Virtual Machine (JVM) to call and to be called by native applications. The latter are programs specific to a hardware operating system platform as well as libraries written in other languages, such as C++.

The JNI framework lets a native method utilize Java objects in the same way in which Java code uses these objects. A native method may create Java objects and then inspect and use these objects to perform its tasks.

Because *JNI* should communicate with the *GenericRuntimeLib*, it is used to create C++ header files with *javah*. Within a *C++* development environment such as *Visual Studio*, they then define the interfaces required to implement the main program. During the build, the post build event copies the *dll* and *pdb* (debug information for debugging) to the Java project root directory, where they may be used with JNI.

Marshalling of data types

Different data types that are exchanged by function calls have to be considered. All native data types are mapped with Java data types, and may be directly converted by JNI. For compound types such as objects, arrays and strings, the program must explicitly convert the data before passing them to methods, and vice versa. Table 18 shows the mapping of native types between Java and native code.

Native Type	Java Language Type	Description
unsigned char	jboolean	unsigned 8 bits
signed char	jbyte	signed 8 bits
unsigned short	jchar	unsigned 16 bits
Short	jshort	signed 16 bits
Long	jint	signed 32 bits
long long int64	jlong	signed 64 bits
Float	jfloat	32 bits
double	jdouble	64 bits

Table 18: Mapping of Java data types to native types.

JNIEnv*

A JNI interface pointer (*JNIEnv**) is passed as an argument to each native function. This allows interaction with the JNI environment within the native method. For example, it may be used to determine the class name of a passed object or to create new Java objects from native code. The JNI interface pointer remains valid only in the current thread. Other threads must first call *AttachCurrentThread()* to attach themselves to the JVM and obtain a valid JNI interface pointer. Once attached, a native thread works like a regular Java thread running within a native method, and remains attached to the JVM until it calls *DetachCurrentThread()*. Listing 18 and Listing 19 show how threads are attached to, and detached from the JVM.

```
JNIEnv *env;
(*g_vm)->AttachCurrentThread (g_vm, (void **) &env, NULL);
```

Listing 18: Attach native thread to JVM.

```
(*g_vm)->DetachCurrentThread (g_vm);
```

Listing 19: Detach native thread from JVM.

Implementation

This subsection describes the implementation of the call chain shown in Figure 118. The Java program uses the Plugin Manager, which is a library file, and was developed in C++ using Visual Studio. It implements the header files that were generated by javah, and has the ability to load further library files containing functions for execution. After execution, the result is passed back to the java program. The plugin manager contains the interface between Java and the native code shown in Listing 20.

```
private static native Object pluginManagerInvoke(int returnType,
String dllName, String methodName, LinkedList<Object> parameters,
int resultArrayFirstDim, int resultArraySecondDim);
```

Listing 20: Native method definition in java class.

The arguments that are passed to the function are listed in Table 19.

Name	Type	Description
returnType	Int	The excepted return type
dllName	String	Name of the dll-ile to load
methodName	String	Name of the method to call
parameters	LinkedList<Object>	The parameters that should be pushed to the method
resultArrayFirstDim	Int	Size of the return array (if expected)
resultArraySecDim	Int	Size of the second dimension of the return array (if expected)

Table 19: Data types.

The plugin manager may perform a successful execution only if it knows of the data types of the arguments and the result. While this is required to allow the allocation of sufficient memory for the native function, the result has to be converted to a correct Java object before it is passed back. Because the allocation of dynamic arrays is not possible in C or C++, the array length also has to be passed. To determine which data type to use, these are mapped to a predefined integer value, and can thus be correctly instantiated. Table 20 shows the mapping from the integer value to the data type.

Int type	Interpretation
0	int
1	float
2	double
3	string
4	int[]
5	float[]
6	double[]
7	Int[][]
8	float[][]
9	double[][]

Table 20: Return types.

Arguments are passed within a *LinkedList<Object>*, which should contain the type of the argument, as shown in the table above, and then the argument. For example, a string and an integer array are passed in the linked list, as shown in Listing 21.

```
LinkedList<Object> testList = new LinkedList<Object>();
int[] intArray = {1,2,12};
testList.add(3);
testList.add(new String("Hallo"));
testList.add(4);
testList.add(intArray);
```

Listing 21: Creating a linked list.

Before executing the native function, the plugin manager first determines the size of the passed argument list, which has to be a multiple of two because there is always a pair containing the data type definition and data given. After this, each object from the passed *LinkedList<Object>* is changed to the corresponding native type, depending on the type given in the LinkedList. The native types are saved in a structure ('*struct st_param*'), and they are then pushed to the native method using 'dyncall'.

After the execution, the return value is changed back to the expected type, and it is passed back to the Java Program.

The header file used by the *PPA_Plugin_Manager.cpp* is auto-generated by JNI, and so the following functions have to be implemented:

- The function *pluginManagerAddDll* loads a library file.
- The corresponding function *pluginManagerReleaseDll* unloads a given library file.
- The function *pluginManagerInvoke* was used as discussed above.

The Plugin Manager also throws exceptions, which are passed back to the java program, so that the user is informed about errors that occurred. Exceptions are thrown when the library file or the function may not be found. Furthermore, the size of the argument list has to be a multiple of two (always a pair of return type and argument), and it is detected when there is an incorrect argument number, either at the return type or in the argument list. An exception may also be thrown when the maximum number of loadable library files is reached. The plugin manager is now capable of loading up to 10 library files.

I. Sample Source Code

I.1 Message Service

```
package de.kohrt.ppa.common.messageservice;
import java.net.InetAddress;
...

public class MessageService extends _ITransmitIceDisp implements IMessageService,
Runnable {
    private static final long serialVersionUID = -8864537422336502554L;
    private static Logger log = Logger.getLogger(MessageService.class);
    private static Vector<MessageService> messageServices = new
Vector<MessageService>();
    private HashMap<String, ITransmitIcePrx> iceProxies = new HashMap<String,
ITransmitIcePrx>();
    public Ice.Communicator serverIceCommunicator = Ice.Util.initialize();
    private String internetAdress = "localhost";
    private String internetPort = "10000";

    private void destructConnections() {
           for (ITransmitIcePrx proxy : iceProxies.values()) {
                  proxy.ice_getCommunicator().destroy();
           }
    }

    @Override
    public String getInternetAdress() {
           return internetAdress;
    }

    private void setInternetAdress(String internetAdress) {
           this.internetAdress = internetAdress;
    }

    @Override
    public String getInternetPort() {
           return internetPort;
    }

    private void setInternetPort(String internetPort) {
           this.internetPort = internetPort;
    }

    private EventListenerList messageServiceListeners = new EventListenerList();

    private ActorPriorityQueue apq = new ActorPriorityQueue(100);

    public Boolean disposed = false;

    public boolean isDisposed() {
           return disposed;
    }

    private HashMap<String, IMessageServiceListener> addressbook = new HashMap<String,
IMessageServiceListener>();

    private Vector<DeployableActor> deployableActors;

    private boolean running = false;

    private ObjectAdapter iceAdapter;

    private MessageService iceObject;

    public Object lock = new Object();

    @Override
    public int hashCode() {
           final int prime = 31;
           int result = 1;
           result = prime * result + ((this.internetAdress == null) ? 0 :
this.internetAdress.hashCode());
```

```
            result = prime * result + ((this.internetPort == null) ? 0 :
this.internetPort.hashCode());

            return result;
    }

    @Override
    public boolean equals(Object obj) {
            if (this == obj)
                    return true;
            if (obj == null)
                    return false;
            if (getClass() != obj.getClass())
                    return false;
            MessageService other = (MessageService) obj;
            if (this.internetAdress == null) {
                    if (other.internetAdress != null)
                            return false;
            } else if (!this.internetAdress.equals(other.internetAdress))
                    return false;
            if (this.internetPort == null) {
                    if (other.internetPort != null)
                            return false;
            } else if (!this.internetPort.equals(other.internetPort))
                    return false;
            return true;
    }

    private MessageService(String ip, int port, Vector<DeployableActor>
deployableActors) throws Exception {
            init(ip, port, deployableActors);
    }

    private void init(String ip, int port, Vector<DeployableActor> deployableActors)
throws Exception {
            this.deployableActors = deployableActors;
            setInternetPort(new Integer(port).toString());
            setInternetAdress(ip);

            try {

                    iceAdapter =
serverIceCommunicator.createObjectAdapterWithEndpoints("Adapter" + ip + port, "tcp -h "
+ ip + " -p " + port);
                    iceObject = this;
                    iceAdapter.add(iceObject, serverIceCommunicator.stringToIdentity(ip +
port));
                    iceAdapter.activate();
            } catch (Ice.LocalException e) {
                    e.printStackTrace();
            } catch (Exception e) {
                    System.err.println(e.getMessage());
            }

    }

    public void connect() throws Exception {

            for (int i = 0; i < this.deployableActors.size(); i++) {
                    deployableActors.get(i).connect(iceProxies);
            }
    }

    private void close() throws MessageServiceIceException {
            this.running = false;

            this.apq.stop();

            if (serverIceCommunicator != null) {
                    serverIceCommunicator.shutdown();
                    serverIceCommunicator.destroy();
                    // iceAdapter.destroy();
                    iceObject = null;
            }

            this.destructConnections();
```

```
            if (MessageService.messageServices.contains(this))
                    MessageService.messageServices.remove(this);
    }

    private static long id = 0;

    private MessageService(Vector<DeployableActor> deployableActors) throws Exception {
            init(InetAddress.getLocalHost().getHostAddress(), new Integer(internetPort),
deployableActors);
    }

    private static void startMsgServiceThread(MessageService msgService) throws
Exception {

            if (MessageService.messageServices.contains(msgService)) {
                    String mes = "It is not allowed to create multiple message services
(more than one) on a single node.";
                    log.error(mes);
                    throw new Exception(mes);
            } else
                    MessageService.messageServices.add(msgService);

            Thread tid = new Thread(msgService);
            tid.setName("MsgService: " + msgService.internetAdress + ":" +
msgService.internetPort);
            tid.start();

            while (!msgService.running)
                    Thread.sleep(100);
    }

    public synchronized static MessageService createMessageService(String ip, int port,
Vector<DeployableActor> deployableActors) throws Exception {
            // create msgservice in new thread
            MessageService msgService = new MessageService(ip, port, deployableActors);

            if (deployableActors != null) {
                    // Init deployable actors
                    for (DeployableActor deployableActor : deployableActors) {
                            deployableActor.init(msgService);
                    }
            }

            startMsgServiceThread(msgService);

            return msgService;
    }

    @Override
    public void addMessageServiceListener(IMessageServiceListener listener, String name)
{
            addressbook.put(name, listener);
            messageServiceListeners.add(IMessageServiceListener.class, listener);
    }

    @Override
    public void run() {
            try {
                    this.running = true;

                    while (true) {
                            if (disposed == true) {
                                    if (apq.size() == 0)
                                            break;
                            }
                            pollMessage();
                    }

                    close();

            } catch (Exception e) {
                    e.printStackTrace();
            }
    }

    @Override
```

```
    public void removeMessageServiceListener(IMessageServiceListener listener) throws
Exception {
            messageServiceListeners.remove(IMessageServiceListener.class, listener);
            addressbook.remove(listener.getMessageServiceId());

            if (addressbook.size() == 0) // shutdown connections and message service
            {
                    apq.stop();
                    disposed = true;
                    synchronized(lock)
                    {
                            lock.notify();
                    }
            }
    }

    private boolean isSentToOtherNode(MsgIce msg) {
            MsgIce m = msg;

            if(msg.receiverPort==null) return false;

            String ip = msg.receiverPort.netIp;
            String ownIp = getInternetAdress();
            return !(msg.receiverPort==null || ip.equalsIgnoreCase(ownIp) ||
ip.equalsIgnoreCase(""));
    }

    private boolean isSentToSameThread(MsgIce msg) {
            String ownIp = getInternetAdress();
            String ip = msg.receiverPort.netIp;
            String port = msg.receiverPort.netPort;
            String ownPort = getInternetPort();
            return (ip.equalsIgnoreCase(ownIp) || ip.equalsIgnoreCase("")) &&
(port.equalsIgnoreCase(ownPort) || port.equalsIgnoreCase(""));
    }

    private boolean isSentToOtherThreadOnSameNode(MsgIce msg) {
            String port = msg.receiverPort.netPort;
            String ip = msg.receiverPort.netIp;
            String ownIp = getInternetAdress();
            String ownPort = getInternetPort();
            return ((ip.equalsIgnoreCase(ownIp) || ip.equalsIgnoreCase("")) &&
!(port.equalsIgnoreCase(ownPort) || port.equalsIgnoreCase("")));
    }

    private void pollMessage() {
            MsgIce msg = apq.pollMessage();

            if (msg == null)
                    return;

            try {

                    if (msg.receiverPort == null || msg.receiverPort.portName == null ||
msg.receiverPort.portName == "") {
                            log.error("Receiver port must be provided!");
                    } else
                    {

                            IMessageServiceListener listener =
addressbook.get(msg.receiverPort.portName);

                            if (listener != null) {
                                    /*
                                     * Send message through listener notification.
                                     */
                                    listener.asyncMessageArrived(msg);
                            } else {

                                    log.warn("Port '" + msg.receiverPort.portName + "' is
not ready. Waiting 100ms... and retry.");
                                    Thread.sleep(100);

                                    listener = addressbook.get(msg.receiverPort.portName);

                                    if (listener != null) {
                                            listener.asyncMessageArrived(msg);
```

```
                                } else {
                                        /*
                                         * ERROR
                                         */
                                        log.error("Receiver '" +
msg.receiverPort.portName + "' not found!");
                                }
                        }

                }
        } catch (MessageServiceOverrun e) {
                e.printStackTrace();
        } catch (Exception e) {
                e.printStackTrace();
        }
    }

    @Override
    public MsgData processMsg(MsgIce msg, Ice.Current current) throws Error {
        try {
                return invokeMessage(msg);
        } catch (Exception e) {
                throw new Error(e.getStackTrace().toString());
        }
    }

    private MsgData invokeMessage(MsgIce msg) throws Exception {
        log.debug("Invoke message " + msg.signalName + "    MessageService: " +
this.internetAdress + ", " + this.internetPort);

        if(isSentToOtherNode(msg) || isSentToOtherThreadOnSameNode(msg)) {
                return doInterThreadCall(msg);
        } else if (isSentToSameThread(msg)) {
                return doInnerThreadCall(msg);
        } else {
                throw new Exception("Unknown message service error!");
        }
    }

    @Override
    public MsgData invokeMessage(SignalInOut signalName, PortId senderPort,
Vector<PortId> receiverActorPorts, MsgData msgData) throws Exception {

        if (receiverActorPorts.size() > 1)
                throw new Exception("Sync methods do not support multiple target
ports! (" + senderPort.portName + ")");

        if (receiverActorPorts.size() == 0)
                throw new Exception("No target port connected! (" +
senderPort.portName + ")");

        MsgIce msg = new Msg(signalName, senderPort, receiverActorPorts.get(0),
ECommunicationMode.SYNC, msgData);
        return invokeMessage(msg);
    }

    @Override
    public void sendMessage(SignalOut signalName, PortId senderPort, Vector<PortId>
receiverActorPorts, MsgData msgData) throws Exception {
        if(receiverActorPorts.size()==0)
                log.warn("The port " + senderPort.portName + " ("+signalName.name+")
has no receiver. Is a receiver connected?");

        for (PortId receiverActorPort : receiverActorPorts) {
                MsgIce msg = new Msg(signalName, senderPort, receiverActorPort,
ECommunicationMode.ASYNC, msgData);
                invokeMessage(msg);
        }
    }

    private MsgData doInterThreadCall(MsgIce msg) throws Exception {
        ITransmitIcePrx proxy = null;
        String proxyIdent = "";

        try {
                proxyIdent = msg.receiverPort.netIp + msg.receiverPort.netPort;
                proxy = iceProxies.get(proxyIdent);
```

```
            } catch (Exception e) {
                    System.out.println(e);
            }

            if (proxy == null)
                    throw new Exception("Could not find the proxy " + proxyIdent + "!");

            if (msg.comMode.equals(ECommunicationMode.SYNC)) {
                    return proxy.processMsg(msg);
            } else if (msg.comMode == ECommunicationMode.ASYNC) {
                    Ice.AsyncResult r = proxy.begin_processMsg(msg);
                    try {
                            return proxy.end_processMsg(r);
                    } catch (Error e) {
                            e.printStackTrace();
                            return null;
                    }

            } else if (msg.comMode.equals(ECommunicationMode.FSMSYNC)) {
                    return proxy.processMsg(msg);
            } else {
                    return null;
            }
    }

    private MsgData doInnerThreadCall(MsgIce msg) throws Exception {

            if (msg.comMode == ECommunicationMode.FSMSYNC) {
                    // Synchron
                    if (msg.receiverPort.portName == null || msg.receiverPort.portName ==
""") {
                            throw new Exception("Broadcast sync Message is not
allowed!");
                    }

                    IMessageServiceListener listener =
addressbook.get(msg.receiverPort.portName);
                    if (listener != null) {
                            /*
                             * Send message through listener notification.
                             */
                            return listener.fsmSyncMessageArrived(msg);
                    } else {
                            /*
                             * ERROR
                             */
                            throw new Exception("Receiver '" + msg.receiverPort.portName
+ "' not found!");
                    }

            } else if (msg.comMode == ECommunicationMode.ASYNC) {

                    apq.pushMessage(msg);
                    return null;

            } else if (msg.comMode == ECommunicationMode.SYNC) {

                    if (msg.receiverPort.portName == null || msg.receiverPort.portName ==
""") {
                            throw new Exception("Broadcast sync Message is not
allowed!");
                    }

                    /*
                     * Receiver is known
                     */
                    IMessageServiceListener listener =
addressbook.get(msg.receiverPort.portName);

                    if (listener != null) {
                            return listener.invokeMethod(msg);
                    } else {
                            /*
                             * ERROR
                             */
                            throw new Exception("Receiver '" + msg.receiverPort.portName
+ "' not found!");
```

```
                }

            } else {
                log.warn("Communication mode " + msg.comMode + " is unknown!");
                return null;
            }
    }

    @Override
    public void releasePort(ActorPort port) throws Exception {

            removeMessageServiceListener(port);
    }

    @Override
    public Communicator getCommunicator() {
            return serverIceCommunicator;
    }

    @Override
    public void sendMessage(SignalOut signalName, PortId senderPort, Vector<PortId>
receiverActorPorts) throws Exception {
            for (PortId receiverActorPort : receiverActorPorts) {
                MsgIce msg = new Msg(signalName, senderPort, receiverActorPort,
ECommunicationMode.ASYNC, null);
                invokeMessage(msg);
            }
    }

    @Override
    public void sendMessage(SignalIn signalName) throws Exception {
            MsgIce msg = new Msg(signalName, signalName.port, null);
            invokeMessage(msg);
    }

    @Override
    public void sendMessage(SignalIn signalName, MsgData msgData) throws Exception {
            MsgIce msg = new Msg(signalName, signalName.port, msgData);
            invokeMessage(msg);
    }

    @Override
    public MsgData invokeFsmMessage(SignalInOut signalName, PortId senderPort,
Vector<PortId> receiverActorPorts, MsgData msgData) throws Exception {

            if (receiverActorPorts.size() > 1)
                throw new Exception("Sync methods do not support multiple target
ports! (" + senderPort.portName + ")");

            if (receiverActorPorts.size() == 0)
                throw new Exception("No target port connected! (" +
senderPort.portName + ")");

            MsgIce msg = new Msg(signalName, senderPort, receiverActorPorts.get(0),
ECommunicationMode.FSMSYNC, msgData);
            return invokeMessage(msg);
    }

    @Override
    public void sendReplyMessage(SignalOut signalName, PortId replyPort, PortId
senderPort) throws Exception {
            sendReplyMessage(signalName, replyPort, senderPort, null);
    }

    @Override
    public void sendReplyMessage(SignalOut signalName, PortId replyPort, PortId
senderPort, MsgData msgData) throws Exception {
            MsgIce msg = new Msg(signalName, senderPort, replyPort,
ECommunicationMode.ASYNC, msgData);
            invokeMessage(msg);
    }

    public void waitForDispose() throws Exception {
            waitForDispose(0);
    }

    public void waitForDispose(int i) throws Exception {
```

```
            if (disposed != true)
            {
                    synchronized (lock) {
                            try {
                                    lock.wait(i*1000);

                                    if(disposed==false)
                                            throw new Exception("MessageService not
disposed in '" + i + "' seconds!");

                            } catch (InterruptedException e) {
                                    e.printStackTrace();
                                    System.exit(1);
                            }
                    }
            }
    }
}
```

I.2 Robot Kinematics

Forward Calculation

```
public CartesianWorldPosition forwardKinematic(JointPosition position) {
    double j1 = (position.joints.get(0)) * 180. / M.PI;
    double j2 = (position.joints.get(1)) * 180. / M.PI;
    double j3 = position.joints.get(2) * 180. / M.PI;
    double j5 = (position.joints.get(4)) * 180. / M.PI;
    double j6 = (position.joints.get(5)) * 180. / M.PI;

    if (false == checkRobotJointRangesGrad(j1, j2, j3, j5, j6))
            return null;

    j1 = position.joints.get(0) + Math.PI;
    j2 = position.joints.get(1) + Math.PI / 2.;
    j3 = position.joints.get(2) + 0;
    j5 = position.joints.get(4) + Math.PI / 2.;
    j6 = position.joints.get(5) + Math.PI / 2.;

    Matrix4d m = MitsubishiRV2AJ.getAi(300., j1, 0., M.PI / 2.);
    m.mul(MitsubishiRV2AJ.getAi(0., j2, 250., 0));
    m.mul(MitsubishiRV2AJ.getAi(0., j3, 160., 0.));
    m.mul(MitsubishiRV2AJ.getAi(0., j5, 0., M.PI / 2.));
    m.mul(MitsubishiRV2AJ.getAi(72., j6, 0., 0.));
    m.mul(MitsubishiRV2AJ.getAi(0., 0., 0., 0.));

    CartesianWorldPosition pos = new CartesianWorldPosition(m.m03, m.m13, m.m23, m.m00,
            m.m10, m.m20, m.m01, m.m11, m.m21, m.m02, m.m12, m.m22);
    return pos;
}
```

Inverse Calculation

```
public Vector<JointPosition> inverseKinematic(CartesianWorldPosition p) {
    int z = 10;

    if (p == null) {
            log.error("Cartesian world position is NULL!");
            return new Vector<JointPosition>();
    }

    Point position = calculateMainAxesPositionFromTCPPosition(p);

    Vector<JointPosition> v = new Vector<JointPosition>();

    int[] c1 = { 1, -1 };
    int[] c2 = { 1, -1 };

    double[] tetas = new double[] { 0, 0, 0, 0, 0 };

    for (int i = 0; i < c1.length; i++) {
            for (int j = 0; j < c2.length; j++) {

                    boolean error = calculateMainAxes(tetas, position, c1[i], c2[j]);

                    if (!error)
                            continue;

                    calculateAuxilaryAxes(tetas, p);

                    // Prepare output variable
                    JointPosition ro = new JointPosition(new ArrayList<Double>());
                    ro.joints.add(0, convertAngle(tetas[0]));
                    ro.joints.add(1, convertAngle(tetas[1]));
                    ro.joints.add(2, convertAngle(tetas[2]));
                    ro.joints.add(3, 0d);
                    ro.joints.add(4, convertAngle(tetas[3]));
                    ro.joints.add(5, convertAngle(tetas[4]));
                    ro.joints.add(6, 0d);
                    ro.joints.add(7, 0d);

                    tetas = new double[] { 0, 0, 0, 0, 0 };

                    if (!contains(v, ro)) {
                            printAllRobotJoints(ro);

                    if (checkRobotJointRanges(ro.joints.get(0), ro.joints.get(1),
ro.joints.get(2), ro.joints.get(4), ro.joints.get(5))) {
                            v.add(ro);
                    } else
                            log.info("Position not allowed!");

                    }
            }
    }

    return v;

}
```

Common Transformation Equation

```
public static Matrix4d getAi(double d, double teta, double a, double alpha) {

    Matrix4d m = new Matrix4d();
    m.setElement(0, 0, M.cos(teta));
    m.setElement(0, 1, -M.cos(alpha) * M.sin(teta));
    m.setElement(0, 2, M.sin(alpha) * M.sin(teta));
    m.setElement(0, 3, a * M.cos(teta));

    m.setElement(1, 0, M.sin(teta));
    m.setElement(1, 1, M.cos(alpha) * M.cos(teta));
    m.setElement(1, 2, -M.sin(alpha) * M.cos(teta));
    m.setElement(1, 3, a * M.sin(teta));

    m.setElement(2, 0, 0);
    m.setElement(2, 1, M.sin(alpha));
    m.setElement(2, 2, M.cos(alpha));
    m.setElement(2, 3, d);

    m.setElement(3, 0, 0);
    m.setElement(3, 1, 0);
    m.setElement(3, 2, 0);
    m.setElement(3, 3, 1);

    return m;
}
```

I.3 Program Export

```
public void exportProgram(StateMachineContext context) throws Exception {
    log.info("exportProgram");

    MsgIce arrivedMsg = getMessage(context);

    nodes = ((MsgDataExportProgram) arrivedMsg.msgData).nodesVector;
    pOutputType.value = ((MsgDataExportProgram) arrivedMsg.msgData).mode.name();
    pRobotType.value = ((MsgDataExportProgram) arrivedMsg.msgData).robotType.name();

    if (pOutputType.value.equals(EOutputType.FILE.name())) {
            if (pRobotType.value.equals(ERobotType.MITSUBISHI.name())) {
                    generateFile(new T_mitsubishi_program_bas(), "MitsubishiProgram.bas",
                            pFilePath.value, trajectory);
                    generateFile(new T_mitsubishi_positions_bas(),
                            "MitsubishiPositions.bas", pFilePath.value, trajectory);
                    } else if (pRobotType.value.equals(ERobotType.PSEUDO.name())) {
                            generateFile(new T_pseudo_program_bas(), "PseudoProgram.bas",
                                    pFilePath.value, trajectory);
                            generateFile(new T_pseudo_positions_bas(),
                                    "MitsubishiPositions.bas", pFilePath.value,
                                            trajectory);
                    } else if (pRobotType.value.equals(ERobotType.SIMULATOR.name())) {
                            generateFile(new T_simulator_program_bas(),
                                    "SimulatorProgram.bas", pFilePath.value, trajectory);
                            generateFile(new T_simulator_positions_bas(),
                                    "SimulatorPositions.bas", pFilePath.value,
                                            trajectory);
                    } else {
                            fireEvent(_RaiseError, new MsgDataRaiseError("Robot type '" +
                                    pRobotType.value + "' not defined!"));
                    }

            } else if (pOutputType.value.equals(EOutputType.DIRECTCONTROL.name())) {
                    // TODO
            } else {
                    fireEvent(_RaiseError, new MsgDataRaiseError("Output type not
                    defined!"));
            }

            fireEvent(_FinishExport);
    }
```

I.4 Linear Octree and Trajectory Planning

```
@Override
public void init(StateMachineContext context) throws Exception {
     log.info("Init the octree.");
     // Create linear octree
     linearOctree = new LinearOctree(Type.OCTAL_JOINT, 2., 2 * 62.5);
}

public Trajectory LinearOctreePort_PlanTrajectory() throws Exception {
     Trajectory t = null;

     try {

     log.info("Start planning the trajectory.");

     log.info("Get the start and goal robot joint/Cartesian positions.");
     Pose startPose = ConvertPosition.parseXYZPosition(startRobotPosition.cartPosition);
     JointPosition startJointPosition =
            ConvertPosition.parseJOINTPosition(startRobotPosition.jointPosition);

     Pose goalPose = ConvertPosition.parseXYZPosition(goalRobotPosition.cartPosition);
     JointPosition goalJointPosition =
            ConvertPosition.parseJOINTPosition(goalRobotPosition.jointPosition);


     log.info("Store positions to the octree.”);
     OctreePoint sOctreePoint = new OctreePoint(1, new Point(startPose.x, startPose.y,
            startPose.z));
     OctreePoint gOctreePoint = new OctreePoint(1, new Point(goalPose.x, goalPose.y,
            goalPose.z));

     OctalPoint start = linearOctree.convertOctreePointToOctalPoint(sOctreePoint);
     OctalPoint goal = linearOctree.convertOctreePointToOctalPoint(gOctreePoint);

     linearOctree.points.put(start.getOctalCode(), start);
     linearOctree.points.put(goal.getOctalCode(), goal);

     AbstractDataStructure.createTopology(linearOctree.getVoxelsFromDeepestLevel());

     start.setCollisionProbability(0);
     goal.setCollisionProbability(0);

     sOctreePoint.colisionProbability=0;
     gOctreePoint.colisionProbability=0;

     JointNode startJointCell = new JointNode(start, startJointPosition);
     JointNode goalJointCell = new JointNode(goal, goalJointPosition);

     start.jointNodes.add(startJointCell);
     goal.jointNodes.add(goalJointCell);


     log.info("Do search with the A* algorithm within the octree.”);
     List<AbstractAStarNode> l = linearOctree.search(startJointCell, goalJointCell);

     Object[] o = l.toArray();

     int i = 0;
     for (Object obj : o) {
            i++;
            NPoint op1 = ((JointNode) obj).op;
            OctalPoint op = (OctalPoint)op1;
            Point cp = op.getNormalizedPosition();
     }

     // Get nodes from path (Convert octree cells to nodes)
     ArrayList<Node> nodes = LinearOctree.convertOctreeCellsToNodes((JointNode)
            l.get(o.length - 1));

     // Create path from nodes via roads
     Road road1 = new Road(nodes);
```

```
ArrayList<Road> roads = new ArrayList<Road>();
roads.add(road1);

Path path = new Path(roads, 0);

// Execute EN to get the trajectory
ElasticNet net = new ElasticJointNet();
t = net.formTrajectory(path, 0.0000000001, linearOctree, startJointCell,
        goalJointCell );

} catch (Exception e) {
        e.printStackTrace();
        System.exit(1);
}

return t;
}
```

J. Attachments

KOHRT, C., PIPE, A., SCHIEDERMEIER, G., STAMP, R. and KIELY, J. 2012. *A Flexible Model Driven Robotics Development Framework.* The 43rd Intl. Symp. on Robotics (ISR2012). Taipei, Taiwan.

A Flexible Model Driven Robotics Development Framework

Christian Kohrt
Bristol Institute of Technology, University of the West of England, Bristol, UK
e-mail: christian@kohrt.org

Richard Stamp, Anthony Pipe, Janice Kiely
Bristol Institute of Technology, University of the West of England, Bristol, UK
e-mail: richard.stamp, anthony.pipe, janice.kiely@uwe.ac.uk

Gudrun Schiedermeier
Faculty of Informatics, University of Applied Sciences Landshut, Landshut, Germany
e-mail: gschied@fh-landshut.de

***Abstract*— A flexible robotics development framework has been established to allow rapid development of high-performance real-time applications from distributed software components. The framework interconnects software components and hardware devices as well as specialized third party software applications to allow integration into the communication system with ease. A model driven approach has been chosen in order to raise the usability of the framework using a visual modeling language. A communication middleware has been evaluated for the interconnection of the components. This paper introduces the required tools, proposes a model driven development framework for robotic applications and provides experiences in the development and use of such frameworks.**

Keywords: control, framework, robot, model.

I. Introduction

The motivation for the robotics framework is based on the requirement to rapidly connect distributed software components written in different programming languages and running on different platforms, sensors and third party tools such as Matlab [1] across a network without time consuming development of data communication and tool connection infrastructure. The presented framework is especially designed for large development teams in heterogeneous software environments. An example of such an environment is given in Fig. 1.

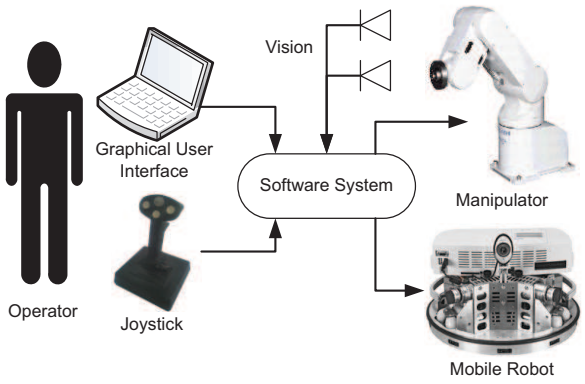


Fig. 1. The experimental system.

An operator utilizes a graphical user interface that is developed with the Java SWT framework [2] on a Windows operating system. Vision sensors are connected and processed by a Matlab/Simulink generated C++ code. The connection to the robots has been established using C# for the mobile robot [19] and a Java framework for the manipulator [20]. A Linux operating system may be used on mobile robots. Thus, the integration of heterogeneous systems becomes important.

This has been accomplished by a model based software development including code generation, which entails the composition of applications from pre-designed hull software components enriched with the business logic of the application. The details of the implementation of the components are hidden behind well-defined interfaces. Thus, much improved software quality becomes realistic. Moreover, previous experiences with component based software development in other application domains have resulted in drastically improved software development productivity - sometimes more than one order of magnitude above conventional software development [3, 4].

Matlab/Simulink is often adopted as a development environment because of its fast modeling and code generation capabilities as well as its valuable library functions. Connecting such a tool to a distributed software system supports the developer during software development by enabling communication with existing components.

The run-time architecture consists of interconnected components, communicating through message passing, which is executed by a communication middleware. Each component is typically a process running on a node such as a computer or an embedded device. An evaluation of existing communication middlewares has been carried out in chapter III.

A model driven approach has been chosen in order to raise the usability of the framework through the use of a domain specific modeling language, derived from the Real-Time Object-Oriented Modeling (ROOM) language [5-7]. This language also defines the run-time behavior of the generated software components.

The commercial tool Rational Rose Real-Time from IBM [8], formerly known as ObjecTime, was a toolset supporting the ROOM language. Unfortunately, this toolset is not available anymore and, consequently, it makes re-implementation of the code execution model and the modeling tool necessary. The eclipse project eTrice [9] has recently shifted from the proposal phase to the incubation phase and aims at an implementation of the

ROOM language together with code generators and tooling for model editing.

A major goal of the proposed framework is to enable sensor-based robot control applications to be built from libraries of reusable software components. For this purpose, the framework provides standard interface specifications for implementing reusable components. A well-written and debugged library of software components facilitates rapid development of reliable sensor-based control systems.

Existing robot control frameworks introduce re-configurable software components as well as special communication and code execution models [10-12]. While these approaches try to enhance configuration of the components for re-use and the running system itself, this paper proposes additionally to enhance the usability by graphical modeling and code generation.

II. System Modeling

ROOM defines a visual modeling language with formal semantics and a code execution model, which is a set of rules defining the system behavior [5-7]. The visual modeling language is optimized for specifying, visualizing, documenting and automating the construction of complex, event-driven, and potentially distributed real-time systems. By connecting several components, an interaction flow via messages may be established between them.

In the proposed framework, a component can be developed in Java, C#, C++ and C, deployed on different processing units. A processing unit may be a general-purpose processor, digital signal processor or a field-programmable gate array, where each processing unit may have its special system architecture that influences for example the handling of threads.

In addition, a component may also be a complete development environment, which allows direct communication to existing components during development time. The integration of tools is explained in chapter VI.

The component behavior is described as a hierarchical state machine, which provides a number of powerful features, including group transitions, transitions to history, state variables, initial points, and synchronous message communications.

The developer writes user programs for state transitions, where the component has to perform an action. Additionally, each state may have an entry and an exit function, which are executed when the component enters or exits the state respectively. This presents various advantages: components may be distributed on different nodes with ease and better encapsulation is reached, because only the component interfaces, not the type of the component, are required in order to interact with it.

ROOM also defines a message service that controls the logical message flow within a physical thread, while a middleware, further described in chapter III, is responsible to transmit the messages. The implemented message service is optimized for speed in the local delivery of messages through the utilization of operating-system specific communication mechanisms. It must be abstract enough to be used by any operating system, but furthermore concrete enough to fulfill requirements in speed, code size and memory consumption. The implemented message service is included together with the code execution model in a runtime library. An instantiated message service is identified by the network port number and the IP of the host.

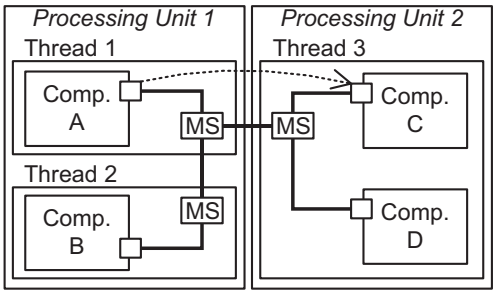


Fig. 2. Communication overview and message passing.

The ROOM communication system illustrated in Fig. 2 consists of processing units, threads, components and message services (MS) along with its connectivity. The ports of each component may communicate with other components via connections to the message service, which handles local and remote message passing. A message from the port of component A to the port of component C (see dashed arrow) may be passed through both message services until it gets to the target port. In this example, messages from component B may only be sent to component A.

III. Communication Middleware

Currently available communication mechanisms may generally be separated into three categories: transport level, message passing and remote procedure calls.

Transport level is simply a pipe to send data streams or packets without any formatting specification, such as serial ports or TCP/IP. Direct socket communication requires the development of an own protocol and exception handling which entails large effort. Furthermore, marshaling and de-marshaling have to be implemented: this is particularly complex because of the requested compatibility between the different programming languages. For example, it is required that a C++ object may be transformed into a Java object.

Message passing adds structure to the packets to define the content but still requires the user software to build and send the messages. ZeroC Ice [13] and CORBA are middleware systems that build an abstract communication layer.

Remote-procedure-calls attempt to expose functions or full objects across a process or network boundary without the user software being aware of the boundary. Remote method invocation may be given as an example.

A comparison among the different communication middlewares supports the choice of the ZeroC Ice middleware. Its implementation is available on various

platforms, including embedded systems, and for different programming languages such as Java, C++ and C# as well. CORBA might be an alternative but it seems to be complex and it does not have the ability of transmitting objects and therefore allows only primitive data types, while ZeroC Ice may handle object transmission. In addition, ZeroC provides Eclipse support, which simplifies the usage of ZeroC Slice, the interface definition language.

IV. The Toolchain

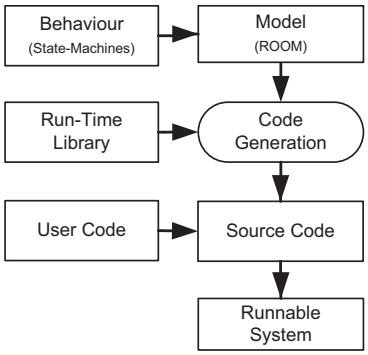


Fig. 3. Code generation workflow.

A general overview of the workflow is given in Fig. 3. The toolchain creates and synchronizes source code from a given graphical model, which includes the modeled behavior of each component. The visual modeling language ROOM is represented as graphical elements in the commercial off-the-shelf editor Enterprise Architect from SparxSystems [14]. This graphical model is utilized to create source code with the help of the eclipse modeling framework (EMF) [15] and its code generation capabilities. The runtime library provides a communication layer, the implementation of the code execution model and the message service.

The generated source code can be synchronized with the written source code of the user to allow modeling and code implementation at the same time. Finally, the source code can be compiled to a runnable application for the target system, e.g. a personal computer with a Windows operating system or an embedded system with a PowerPC operating system.

V. Toolchain Implementation

A more detailed description of the toolchain is given in Fig. 4. The graphical notation elements of ROOM have been integrated into Enterprise Architect [14] with the help of an Enterprise Architect specific MDG Technology file. These modeling elements are utilized to create visual models of executable software systems.

A C# to Java application communication channel has been implemented with a direct socket connection to the Java model repository application. It is utilized to store the visual model into the model repository, which has been defined with the eclipse ecore editor.

The template based code generator application based on Java Emitter Templates (JET) [16] transforms the model to Java source code.

The Code Merger tool utilizes JMerge [15] and runs as headless eclipse application, which starts a minimal eclipse framework in the background. It merges the generated source code with the existing one.

The toolchain supports automatic generation of eclipse Java projects for each component and the runnable system. These projects may be imported into the eclipse workspace. All link dependencies including the link to the run-time library have been automatically set and a UniMod state machine [17] is generated with each component project to define the behavior of the component.

The runtime library has been implemented in a platform dependent manner and includes the ROOM code execution model and the middleware from ZeroC Ice [13].

The middleware supports a target abstraction layer, which simplifies the creation of the platform specific library. This framework also enables the use of specialized tools such as Matlab/Simulink as further described in chapter VI.

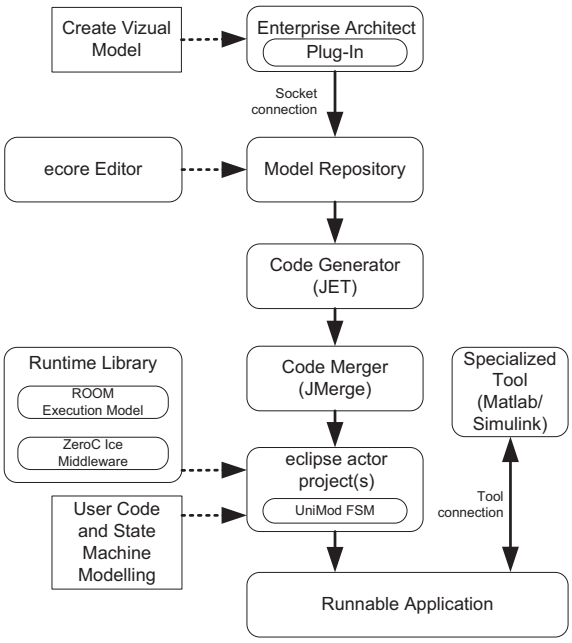


Fig. 4. Toolchain implementation.

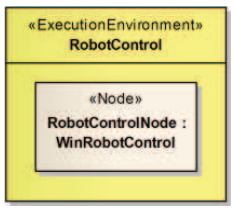


Fig. 5. Execution environment.

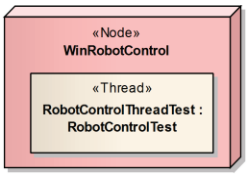


Fig. 6. Node.

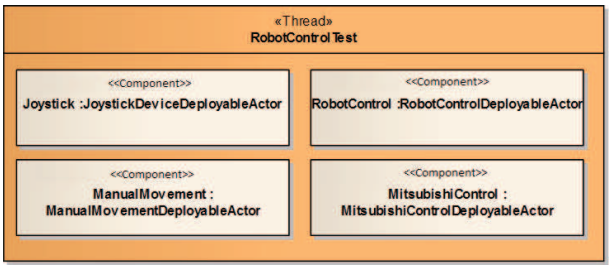


Fig. 7. Component deployment.

VI. Connecting Specialized Tools

Specialized tools have usually enhanced functionality to solve domain specific development tasks and they may have been established as common tools within these domains. Integration of such tools into the communication framework adds communication capabilities to other components, e.g. for sensor or control functionality, during development. The development may be finalized by generating a dynamic link library or an executable, which may successively be used within the communication network. Dynamic link libraries may always be utilized with the help of visual modeling elements that support such libraries and generate the necessary code to incorporate the libraries. The dyncall library [18] has been employed within the run-time library for this purpose.

A direct integration of specialized development tools has been reached through tool specific integration technologies. For example, Matlab may be connected through the Microsoft COM (Component Object Model) or DDE (Dynamic Data Exchange) technology for message passing, which is described in [2]. The middleware can also be directly utilized with an S-function to establish communication to the distributed components.

VII. Code Generation Example

As depicted in Fig. 1, a robot control application with a joystick for the articulated Mitsubishi RV-2AJ robot demonstrates modeling and code generation. Applications are defined by instantiation of an "Execution Environment", which is named "Robot Control" in Fig. 5. Although a single "Win Robot Control" node is deployed to the execution environment for the whole application, several additional nodes may be deployed. Physical threads are modeled to allow thread deployment. Components are finally deployed to those threads (Fig. 7), while their connectivity is modeled in a thread independent manner, as illustrated in Fig. 8. The interface definition of the "Manual Movement Deployable Component" in Fig. 9 describes provided and required interfaces, fixed to component ports. The "Control Port" provides component life-cycle interfaces such as "Control In" in Fig. 10 to start, stop, initialize, release and locate the component. Additional component properties management is implemented with the set and update property signals. Synchronous and asynchronous message passing is supported. Each interface defines allowed signals that have to be modeled in the UniMod finite state machine, as depicted in Fig. 11. A message is received via port interfaces through the port to the state machine of the component, which fires a transition.

The executed transition method contains the user code. The generation process generates for example the initialization methods shown in Listing 1, derived from the "Init" transition. JMerge uses code tags like „@generated" to indicate that this method is generated and overwritten until the tag is manually changed into „@generated not" or just deleted.

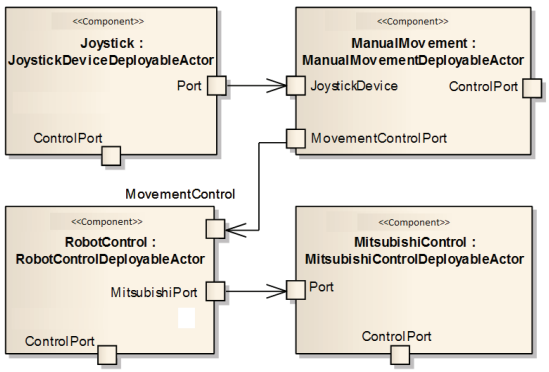


Fig. 8. Component connections.

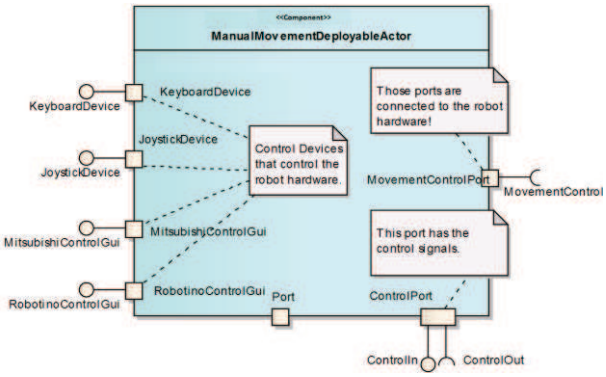


Fig. 9. Component interfaces.

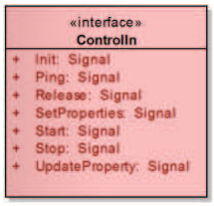


Fig. 10. Interface definition.

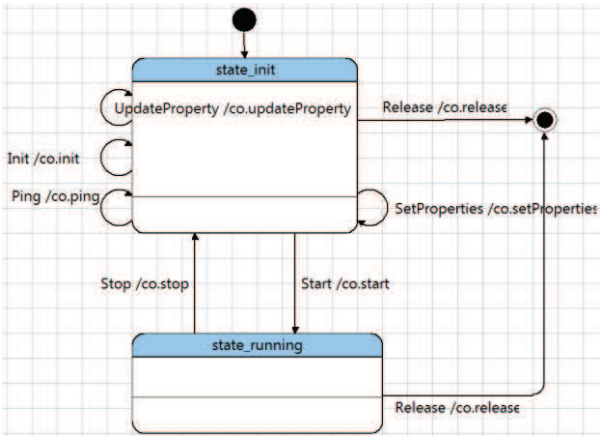


Fig. 11. UniMod state machine diagram example.

```
/**
* Init the component.
* @generated
*/
protected void init()
{…}
```
Listing 1. Generated Java code.

Other tags such as "@unmodifiable" may also be used to control the merge functionality.

VIII. Conclusion

This paper highlights important aspects in the development of the proposed model driven toolchain. The various model-to-model transformation stages and tools are presented from graphical ROOM models to the runnable application. The toolchain may be used for software development in general and for specific problem domains such as robotics. Extensibility of the domain specific language allows domain-oriented engineering. The level of abstraction is a significant aspect for the handling of large software systems. Using a model driven toolchain the abstraction level is raised. Standard designs and concepts may be integrated and used with ease by the developers who only need the graphical front end to such extensions. Encapsulation result in the so-called black box reuse, a favorable form of it, since the economics of scale allows spending more effort on software design, software reviews and software testing. Integration of specialized tools and development environments enhanced the development process.

The proposed model based code generation framework adds a significant productivity benefit, although implementation of the toolchain requires high investments. However, once a toolchain is developed, it may be applied with ease.

ROOM is a message based system based on state machines and it requires training for inexperienced developers. The message service is an additional layer that interprets and transfers messages to the target component port, which may lead to a delay in the message delivery. The delay must be considered, especially for time critical systems. Therefore, it plays a key role regarding performance of the system. Nevertheless, such a toolchain can be valuable for large software development projects and allows a strict encapsulation into components with clearly defined interfaces. It is intended to continue with this methodology and further enhance the modeling and code generation features, especially for debugging purposes and implementation of a state machine (with a graphical editor) alternative to the slow UniMod state machine. The Simulink Stateflow state machines might be used in Simulink context, but it requires adaptation to be usable in non-Simulink contexts.

The main advantages of model driven development are for example better maintainability, a uniform programming model, reusable model parts, simple but efficient communication, higher abstraction, code generation, system wide optimization possibilities and focused development in relation to the business logic.

References

[1] TheMathworks, Matlab/Simulink. 2011; Available from: http://www.mathworks.de.
[2] Kohrt C, Rojko R, Reicher T, Schiedermeier G, "With Model Based Design To Productive Solutions Professional GUIs For Simulink By Utilizing The Java SWT Library," in: KFZ-Elektronik, WEKO Verlag, May 2006, pp. 265-272.
[3] Zincke G, "How to achieve 7.52 function-points per person-day with object technology," in: Addendum to the 1997 ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, Atlanta, Georgia, United States, 274572: ACM, 1997, pp. 21-26.
[4] Sutherland J, "Why I love the OMG: emergence of a business object component architecture," StandardView, vol. 6(1), pp. 4-13, 1998.
[5] Selic B, "Real-Time Object-Oriented Modeling (ROOM)," in: IEEE Real Time Technology and Applications Symposium: IEEE Computer Society, 1996.
[6] Selic B, "Modeling real-time distributed software systems," in: Proc 4th Int Parallel and Distributed Real-Time Systems Workshop1996, pp. 11-18.
[7] Selic B, Gullekson G, Ward PT. Real-time object-oriented modeling: Wiley Professional Computing, 1994, pp.
[8] IBM Corp., Rational Rose Real-Time. 2011; Available from: http://www.ibm.com.
[9] eTrice Group, Eclipse eTrice project page. 2011; Available from: http://www.eclipse.org/etrice/.
[10] Wason JD, Wen JT, "Robot Raconteur: A communication architecture and library for robotic and automation systems," in: IEEE Conference on Automation Science and Engineering (CASE), 24-27 Aug. 2011, pp. 761-766.
[11] Griph FS, Hogben CHA, Buckley MA, "A generic component framework for real-time control," in: IEEE Transactions on Nuclear Science, vol. 51(3), pp. 558-564, 2004.
[12] Lee C, Yangsheng X, "Message-based evaluation for high-level robot control," in: Proceedings IEEE International Conference on Robotics and Automation, 16-20 May 1998, pp. 844-849 vol.1.
[13] ZeroC Inc., ZeroC Ice. 2011; Available from: http://www.zeroc.com.
[14] Sparx Systems, Enterprise Architect. 2011; Available from: http://www.sparxsystems.com.
[15] Eclipse Foundation, Eclipse Modelling Framework (EMF). 2011; Available from: http://www.eclipse.org.
[16] Eclipse Foundation, Java Emitter Templates (JET). 2011; Available from: http://www.eclipse.org.
[17] eVelopers Corporation, UniMod. 2011; Available from: http://unimod.sourceforge.net.
[18] Adler D, Philipp T, Dyncall Library. 2011; Available from: http://www.dyncall.org.
[19] Festo, Robotino. 2012; Available from: http://www.festo.de.
[20] Kohrt C, Stamp R, Pipe A, et al, "A robot manipulator communications and control framework," in: Proc. IEEE Int. Conf. Mechatronics and Automation (ICMA), 2008, 846-851.

KOHRT, C., PIPE, A., KIELY, J., STAMP, R. and SCHIEDERMEIER, G. (2012) *A Cell Based Voronoi Roadmap for Motion Planning of Articulated Robots Using Movement Primitives.* International Conference on Robotics and Biomimetics.

Proceedings of the 2012 IEEE
International Conference on Robotics and Biomimetics
December 11-14, 2012, Guangzhou, China

A Cell Based Voronoi Roadmap for Motion Planning of Articulated Robots Using Movement Primitives

C. Kohrt, A. G. Pipe, J. Kiely, R. Stamp, G. Schiedermeier

Abstract— The manufacturing industry today is still focused on the maximization of production. A possible development able to support the global achievement of this goal is the implementation of a new support system for trajectory-planning, specific for industrial robots. This paper describes the trajectory-planning algorithm, able to generate trajectories manageable by human operators, consisting of linear and circular movement primitives. First, the world model and a topology preserving roadmap are stored in a probabilistic occupancy octree by applying a cell extension based algorithm. Successively, the roadmap is constructed within the free reachable joint space maximizing the clearance to the obstacles. A search algorithm is applied on robot configuration positions within the roadmap to identify a path avoiding static obstacles. Finally, the resulting path is converted through an elastic net algorithm into a robot trajectory, which consists of canonical ordered linear and circular movement primitives. The algorithm is demonstrated in a real industrial manipulator context.

I. Introduction

ROBOT use and automation levels in the industrial sector will inexorably grow in future, driven by the present need for lower item costs and enhanced productivity. Synonymous with this projected increase will be the requirement for capable programming and control technologies. Many industries employ offline programming within a manually controlled and specified work environment. This is especially true within the high-volume automotive industry, particularly when related to high-speed assembly and component handling, but also in the case of medium sized and small batch manufacture. Any scenarios, reliant on manual data input, based on real world obstructions, necessitate the complete production system being offline for an appreciable time while data is input. These production downtimes consequently cause financial losses. Published research appears to be concentrated on the application of simulation tools to generate discrete portions of the total robot trajectories [1, 2], whilst necessitating manual input to link paths associated with one particular activity with those of another. Human input to correct inaccuracies as well as errors resulting from unknowns and falsehoods in the environment is needed. In addition, simulation tools are complex and require highly skilled workers. Offline robot program generation is time intensive also due to inaccuracies; even then its correct operation is not guaranteed.

This has led to the vision of an enhanced online robot programming software application to support the robot programmer. An overview for online robot programming is given in [3] and it is stated there that only one approach has led to a commercial tool.

Investigations have been undertaken with the aim of developing an online robot software application, by considering the working production environment as a single whole workspace. Use is made of automated workspace analysis techniques and a trajectory planning algorithm, described in this paper, to realize the robot software application.

In this article, we consider the high level of complexity of typical robot-programming tasks for human operators; consequently, the robot application-software we present here, takes over the most complicated task, which is robot motion planning. The remaining manageable tasks related to the given mission, e.g. spraying, handling or painting, continue to be the responsibility of the operator. In a handling mission for example, the operator provides information about what the robot has to do, e.g. placing objects to specific positions in a specified order, while the online robot software application knows how to control the robot. This is accomplished with the help of the trajectory planning algorithm presented here.

This trajectory planning algorithm is an important integral part of the enhanced online robot programming application to find suitable robot trajectories in order to generate the robot program with the required features. A robot trajectory is a path in the working space of the robot. Each point on the path is described as a vector with the position and the time. The trajectory planning task here is to find a collision free movement of the robot from the start to the target location considering the motion constraints of the robot (e.g. a car that cannot move sideways), whilst also satisfying the requirements for readability, maintainability and changeability of the derived robot program.

Laboratory tests in Section IV have demonstrated that the so achieved trajectory represents a trade-off between path shortness of the trajectory and readability, maintainability and changeability of the resulting robot program.

Manuscript received October 15, 2012.

Christian Kohrt is with UWE - University of the West of England, Bristol, UK (e-mail: christian@kohrt.org).

Anthony G. Pipe is with UWE - University of the West of England, Bristol, UK (e-mail: anthony.pipe@uwe.ac.uk).

Gudrun Schiedermeier is with UASL – University of Applied Sciences Landshut, Germany (e-mail: gschied@fh-landshut.de).

Richard Stamp is with UWE - University of the West of England, Bristol, UK (e-mail: richard.stamp@uwe.ac.uk).

Janice Kiely is with UWE - University of the West of England, Bristol, UK (e-mail: janice.kiely@uwe.ac.uk).

978-1-4673-2127-3/12/$31.00 © 2012 IEEE

II. LITERATURE OVERVIEW

Trajectory planning is a fundamental problem and significant research has been conducted during the last decades either in static or in dynamic environments [4]. For example, roadmap methods [5] do not compute the whole configuration space, they rather try to generate a roadmap of suitable configurations. Apart from roadmap based techniques, the potential field approach [6, 7] and cell based methods [8] are two popular path planning approaches.

The cell based method in combination with the potential field has been studied in [8] and has been successfully applied to arbitrary shaped robots in dynamic environments. The computation time of the potential field has been reduced by introducing hierarchical subdivision approaches such as quadtree and octree based methods [9]. Cell based methods often generate a path connecting the midpoints of the cells. The publication [10] identifies two limitations with cell based methods. First, the detection of small passages requires high accuracy of the octree or quadtree. Second, the shortest path is not always identified since the distance calculations of the cells often use the midpoints of the cells. Thus, the paths obtained by the cell based method are not optimal because of the connectivity limitations in a grid.

The potential field approach has several limitations as outlined in [6]. In particular, the robot may get stuck at a local minimum and the reported paths can be arbitrarily long.

Voronoi based path planning methods have been studied in [11-17]. However, the quality of the path obtained directly from the Voronoi diagram is long and not smooth. In the recent years, improving the quality of the path has been an active area of research. In [18], the Voronoi diagram was combined with the visibility graph and potential field approach to path planning into a single algorithm to obtain a trade-off between safest and shortest paths. The algorithm is fairly complicated but the path length is shorter than those obtained from the potential field method or the Voronoi diagram.

Most of the algorithms have limitations in real-time path planning where the world model with unknown obstacles is updated during runtime. These algorithms work best on given maps including full knowledge of all obstacles.

III. TRAJECTORY PLANNING

The trajectory-planning algorithm plans a trajectory between two given joint positions. A linear octree [9] is used to represent the working space of the robot in a spatial world space. The octree stores its cells in a predefined maximum accuracy defined by the octree depth. Each cell contains a binary tree to store the robot joint positions and stores a reachability value, which describes if the robot can move its tool-center-point (e.g. the robot hand) into the cell area without collision. The general reachability is stored in a pre-calculation step described in Section III.B.

In addition, each cell stores an occupancy value as well. Cells are defined as fully, partly or not occupied, depending on the obstacles within the working space. This information is input by external sensors through a sensor fusion framework. A collision button and computer-aided design data of a construction process of the working cell have been utilized in the test environment to detect obstacles. The choice is based on the fact that model data is often available and the operator itself is a reliable source to detect collisions. Additional more advanced sensors, such as machine vision can be applied as well to increase the recognition performance, but this work is beyond the scope of this article.

The occupancy and the reachability information are employed to create a roadmap within the reachable free space of the octree. The roadmap forms a Voronoi diagram, which is created by a cell-based algorithm within the octree.

A search algorithm is executed on the joint positions located within the roadmap to identify the shortest path from the start to the target position. Subsequently, the so derived path is turned into a trajectory through the application of the elastic net, presented in [19].

The employed algorithm facilitates only kinematic forward calculations to avoid ambiguities and to reduce computation time of the inverse kinematic calculation.

A. World Model

Path planning is based on data about the physical environment stored within the world model. It is implemented as a linear octree [9] that stores pre-existing and dynamic information of the environment.

The computer-aided design data of a construction process of the working cell is adopted as information source whereas a collision indication button, held by a human operator, is utilized as a real-time sensor. During the execution of the path planner, the operator indicates upcoming collisions not predicted by the automated system through the real-time sensor. Robot type information is particularly important allowing the use of a simulation model of the robot to afford forward and inverse kinematic pre-calculations.

The world model handles the information and combines the CAD data and real-time data mentioned above. The deriving data fusion is carried out as a voting system [4]. Real robot applications have demonstrated that sensors may deliver wrong information [20]. Therefore, each sensory source is classified through the reliability weight between 0.0 and 1.0 and an applied simple moving average filter delivers cohesive information.

B. Reachability Calculation

The configuration space of an articulated robot is often discretized in order to execute a path searching algorithm on the discretized search space. The discretization plays an important role since the accuracy of the search algorithm is often coupled with the accuracy of the discretization. The approaches in [21-23] use hierarchical structures, capability maps or non-uniform discretization to optimize the search space to enable efficient searching.

Optimization can in general be reached by minimizing or

ordering the search space specifically for the applied search algorithm. The planning algorithm described here is executed in the constrained configuration space to improve the search algorithm, as will be seen in the next sections. The reachability of the robot is required to calculate these constraints.

The reachability of a robot in world space can be calculated by transforming the robot configurations from the tool center point coordinates to world coordinates or vice versa. This transformation can be applied with forward or inverse calculations of the robot kinematics. An efficient inverse calculation can only be achieved for world coordinates with given information about its position and orientation. Since the orientation can be arbitrarily chosen, inverse calculations lead to intensive computation.

This problem has been studied in [21, 22] and a simple pre-calculation step is proposed to generate and persist the required information in a look-up table by forward calculations of the robot arm configuration to the points in space. The look-up table may, in general, be used if the robot kinematics are static and known beforehand. Since this algorithm is used in an industrial environment, both statements are fulfilled. The aim of the look-up table is to represent the reachability with a limited number of joint positions P_{RJ} to reduce the search space for a path-searching algorithm. The number of joint positions P_{RJ} has direct impact on the running time of the path searching algorithm and the required pre-calculation time of the look-up table.

The limitation is possible because of the employed search algorithm described in Section III.E. The discretization of the configuration space has been implemented with a robot link dependent accuracy a_n (with n is the link number), which identifies the link importance and considers the sweep occupation volume of the robot links as well as the mechanical constraints of the robot joints.

The implemented linear octree O - the world model - has a defined depth d, which allows calculation of the smallest octree cell size. This can be further employed to estimate the robot link dependent accuracies a_n, which have to be carefully chosen. In order to guarantee that the path-searching algorithm will complete the search task successfully, it has to be ensured that enough discretized positions P_{RJ} are stored per octree cell on the deepest level.

The octree accuracy does not need to be very high because the employed trajectory planning methodology discussed in Section III.E only applies to the octree for path searching. The trajectory generation algorithm actively requests additional positions and operates almost independently from the octree.

Various methodologies for discretizing the configuration space are presented in the literature. An optimal discretization methodology that sets the resolution along each configuration coordinate (robot axis) according to the maximum movement of the robot end-effector at each step that the robot moves along this coordinate is described in [24]. The discretization resolution is determined with $\Delta q = (\Delta q_1, \dots, \Delta q_D)$ of a D-dimensional configuration space. A uniform discretization for all joints of the robot manipulator can be defined with $\Delta q_i = c$ for some constant c.

With a reasonable joint resolution of 1°, the uniform discretization results in huge configuration spaces. For example, a discretization of the joints of the Mitsubishi RV-2AJ with $\Delta q = (1°, 1°, 1°, 1°, 1°)$ results in a configuration space with $89.42 \cdot 10^{10}$ states.

The algorithm presented in this article is based on equation (1), where l_i is the distance between the centers of joint i to the farthest point the end-effector can reach, and $MaxMove$ is a pre-set distance the robot may move at one step along the coordinate.

$$\Delta q_i = 2 \cdot \arcsin \frac{MaxMove}{2 \cdot l_i} \quad (1)$$

The optimal discretization results in Cartesian movements Δx_i of the joint i, which meets the condition $\Delta x_{max} \leq MaxMove$ where $\Delta x_{max} = \max\{\Delta x_i, \forall i\}$.

For $MaxMove = 10mm$ of a Mitsubishi RV-2AJ industrial robot, the optimal discretization equals to $\Delta q = (0.80°, 0.98°, 1.73°, 3.33°, 5.73°)$.

The size of the corresponding configuration space considering the mechanical constraints for the utilized Mitsubishi robot is $3.42 \cdot 10^{10}$ states. This is 1.89 magnitudes less compared to the uniform discretization with $\Delta q = (0.8°, 0.8°, 0.8°, 0.8°, 0.8°)$ and $264.99 \cdot 10^{10}$ states. The configuration space is computed with forward calculations of the robot manipulator and the joint positions P_{RJ} are stored within the octree. This calculation has to be done once per robot.

C. Occupancy Calculation

The occupancy calculation is done beforehand and online by sensors during path planning to update the world model. The first information source is modeled data, which has been applied to the in-memory world model and handled as a sensor. This information source is amended by a binary collision indication button of the operator and it has turned out to be sufficient for the proposed trajectory planning algorithm.

D. Voronoi Based Roadmap Generation

Roadmap methods generally identify a set of roads, which may be safely travelled along without incurring collisions with obstacles. The method here adopted has been inspired by [16], based on the Voronoi form [25, 26]. This choice has been taken after considering two important aspects. First, the Voronoi form may be applied either in the world space or in the joint space of the robot. Second, it maximizes the clearance of obstacles, so that the path-planning algorithms do not have to be particularly accurate. The second point may be perceived as a negative characteristic too, since the derived roads are not short, smooth or continuous enough to guarantee an enhancement [18, 27] (see Section III.E). In fact, implementation tests of [16] have shown that a Voronoi

form is rarely reached. Adjustments of the parameters by trial and error, as suggested by the authors of [16], have not led to any improved results either. In addition, real-time robot control with this kind of neural network requires processing of the neurons to adapt to the environment including the obstacles. Since random positions are not available in real environments the proposed approach has not been followed here any further.

Hence, the concept at the basis of the Voronoi form has been extended and applied to a grid-based algorithm. First, the obstacle and border cells are added to an open list. Successively, all neighbor cells are iterated for all elements in the open list in order to mark them with the obstacle number according to the currently examined element of the open list. The currently examined element is moved from the open to the closed list and extended cells are added to the open list to be examined in the next iteration.

The general grid-based algorithm described in Listing 1 produces the approximated Voronoi diagram. The primary aim is to approximate a Voronoi form between the obstacles and the border cells in joint space.

The grid used in the implementation is an octree in three dimensions. It allows adding obstacles during runtime while recalculation is only necessary for neighboring areas (see III.D). The octree also provides the opportunity to utilize its hierarchy to speed up the algorithm to efficiently store environment information. Application of this cell extension methodology builds a roadmap that supports real time development of the topology and connectivity of the robot workspace.

This algorithm is applied to the tool center point of the robot. The maximum clearance of the whole robot arm to the obstacles is indirectly considered because reported collision indication positions are stored as robot joint positions into the cell. The cell occupancy is always calculated based on all postures and thus, its occupancy value is accordingly calculated.

During the execution of the path-planning algorithm, new information of the working space and the obstacles is provided by the employed sensors and information sources, which are the collision button and the computer aided design model. New joint position information is added to the data structure in the steps described in Listing 2.

The world coordinate of the position is determined by forward kinematics calculation successively storing the joint position into the octree cell that is responsible for the world position region.

The cell is marked with an occupation value in accordance to the reported and fused sensor value O_{Cell}. A probability threshold of $t_P = 0.8$ is applied in equation (2) to transform the cell occupancy value to the binary value $O_{Cell_{Binary}}$ required by the Voronoi roadmap generation algorithm.

$$O_{Cell_{Binary}} = \begin{cases} O_{Cell} \geq t_P & 1 \\ \text{else} & 0 \end{cases} \quad (2)$$

1. Store all border, obstacle and extended cells C in the open list
2. While open list element count > 0
   2.1. Take first cell C_i from the open list
   2.2. Inspect all neighbour cells of C_i and mark each extended neighbour cell according to the following conditions:
      2.2.1. If the extended cell is located between two or more obstacles
         2.2.1.1. If the cell is not reachable it is marked '0'
         2.2.1.2. Else it is marked '-1'
      2.2.2. Else copy the mark from cell C_i
   2.3. Add all neighbour cells of C_i, which are not in the closed list, to the open list
   2.4. Move cell C_i from the open list to the closed list
3. Wend

Listing 1. Cell extension algorithm.

1. Get the robot posture for a collision indication
2. Execute forward calculation to get the world position
3. Store the joint position to the responsible octree cell
4. Calculate the occupation value for the cell
5. Update the parent cells
6. Recalculate the cell region d to obtain the updated Voronoi diagram

Listing 2. Obstacle addition algorithm.

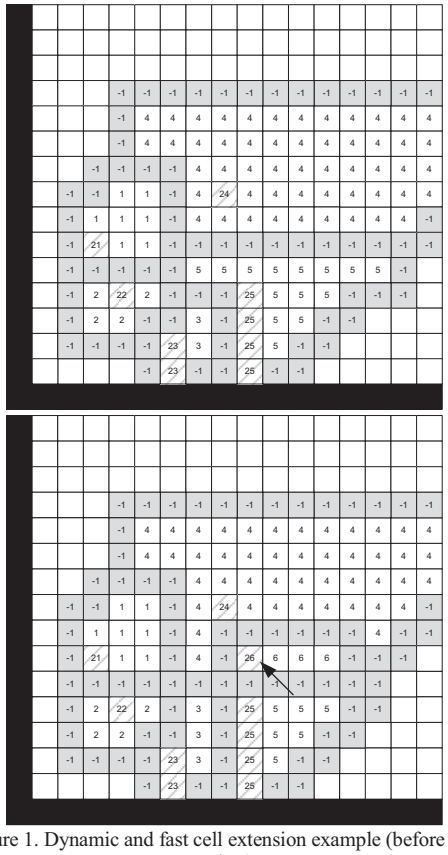


Figure 1. Dynamic and fast cell extension example (before and after update).

Parent cells are updated to either partly or fully occupied depending on the occupation of the child cells of the parent. Parts of the Voronoi roadmap have to be recalculated if new collision information is processed. A minimum distance d_{min} of the robot TCP is introduced to those obstacles, which is

used to clear surrounding extended groups of cells within the distance d_{min}. An example is illustrated in Figure 1.

The cell in position (9, 6) is updated and marked as occupied (see second figure, cell number 26). A radius of $d_{min} = 5$ cells is considered. As a result, the group information and the Voronoi path are recalculated.

The second example in Figure 2 focuses on the defined distance and shows how the distance influences the Voronoi path generation. The distance to the occupied cells shall be maximized within the given boundary of d_{min}. The occupied cell '27' (only its extended cells '7' are visible) is next to the newly added occupied cell '26' and, thus, the Voronoi path is adapted. The guaranteed space between the Voronoi path and the newly added cell is $d_{min}/2$ because the cell extension mechanism starts from the given distance and grows from both sides in order to meet in the middle of d_{min}.

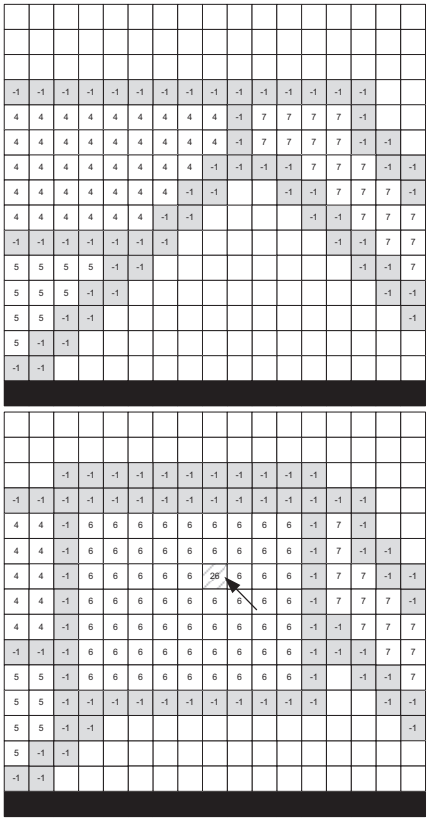


Figure 2: Defined distance influence on Voronoi path generation.

The algorithm is summed up in Listing 3, where the group information is updated for each obstacle addition.

```
1. Add new obstacle cell to open list
2. Reset and move cells within the distance d_min
   from the closed to the open list
3. Apply the algorithm from Listing 1
```

Listing 3. Cell addition for obstacles.

E. Search within the Roadmap

The robot may be seen as a Dubin's car [28] in three dimensions that may be steered from the start to the target location by real robot movements. During each movement, the world model gets updates in the form of obstacle joint positions. The algorithm finds a path to the target location based on the roadmap and accuracy of the octree, considering all the joint positions within the cells of the roads. The accuracy of the road cells are uniformly at the highest level. This also defines the minimum size of small passages that may be captured.

Path planning consists of two steps. First, within the roadmap the shortest path from the start to the goal is calculated within the joint space. As a second step, the algorithm transforms the identified path into a trajectory consisting of movement primitives, described in Section III.F. The trajectory avoids obstacles and reduces the clearance to them. This is done by forces applied on the roads within the map [19].

Information about the environment in which the robot operates and about the objects it has to avoid is captured within the roadmap. The roadmap is improved during trajectory planning. Real sensory information is delivered to the roadmap in the form of collision locations. This leads to an adaptation process of the roadmap, which primarily targets the approximation of the Voronoi form.

The A* search algorithm [29] is first used to conduct a local search and connect the start and target locations to the roadmap and, second, to search within the joint positions of the Voronoi roadmap. The start and target locations are handled as obstacles: this means that Voronoi roads are generated around them. The extended cells are added to the search space to connect the location with the Voronoi roads.

The joint distance metric is utilized as heuristic for the A* algorithm. The connectivity of the joint positions is given by the octree cell connectivity. All joint positions of one octree cell are connected to all joint positions of the neighboring octree cell. This may result in high running search times if too many joint positions are stored within the octree cells. The reachability calculation described in Section III.B has to consider this by choosing the accuracy a accordingly. This is highly dependent on the robot geometry.

The octree is an extension of the quadtree, which has shown two limitations [10] in path planning: first, the detection of small passages requires high accuracy of the octree/quadtree. Second, the shortest path is not always identified since the distance calculations of the cells always use the midpoints of the cells.

The first aspect requires the involvement of many cells; consequently, the planning stage may take a great deal of processing time. It is proposed in [10] to overcome this limitation using an obstacle dependent grid. However, in the now proposed approach the octree representation is used to interface between world and joint space coordinates. The number of cells is reduced by the transition to the joint positions, which are assigned to each cell, and by only subdividing needed cells.

The second aspect is solved using joint positions within a cell and the joint distance metric for the A* search. The joint distance between two joint positions is directly computed by the difference of these joint positions. The distance measurement is executed on the joint positions and not on the cells; therefore the octree cell size is decoupled from the distance measurements.

As mentioned in III.C, the occupancy probabilities for the cells are considered as movement costs during path planning. Since the search is not conducted within the cells but within the joint positions, each joint position is allocated the probability given by the containment cell [30]. The connectivity of the octree cells includes direct and diagonal neighbors so that each non-boundary cell has 26 neighbors.

Moreover, the application of the A* algorithm to a real robot in order to identify the shortest path often leads to re-planning of the path itself each time a shorter path is found. Since real robot movements are involved, this should not happen too often. A hysteresis on the path length is applied in order to prevent this and to allow an additional exploration of the working space: consequently, the system achieves environment information stored within the world model.

The A* path planning method together with the probabilistic occupancy map projected on joint positions always delivers the shortest roadmap Voronoi road, if one exists. The search space is reduced by the Voronoi form in world space and the reachability calculation is dependent on the robot geometry. The joint positions are carefully distributed along the roadmap paths. Through the application of this methodology, good performance of the search stage is assured.

F. Elastic Net Trajectory Generation

As mentioned before, transformation of the path to a trajectory is a necessary step carried out by the application of the elastic net. The path within the roadmap found by the A* algorithm consists of connected joint space positions. Transformation of the path into a trajectory is reached by applying equidistance, rotation and shrink forces on the joint space positions [19] in world space. For these positions, both forward and inverse kinematic calculations are used. The generated trajectory consists of canonically ordered movement primitives, which are linear and circular movements. The joint movement type is not of interest for the online path planning application and it is therefore omitted. The transformation considers the reachability and obstacles automatically, as shown in Section 3.

IV. Experimental Results

In this section, the general execution of the programming assistant is described and a scenario (see Figure 3) has been chosen to demonstrate the proposed approach. The system is executed with a real five axis industrial scale, articulated Mitsubishi RV-2AJ robot [31]. The algorithm utilizes an octree as world model (as described in Section III.A) and joint positions attached to the octree cells. During implementation, the algorithm has been tested in simulated two-dimensional space using a quadtree as world model and world positions attached to the quadtree cells. The proposed algorithm works in real surroundings. The illustrations shown in this section are simplified to support understanding of the algorithm.

In the chosen real scenario, the two obstacles O1 and O2 are given as computer-aided design (CAD) objects and imported into the in-memory environment model. One obstacle O3 shall be unknown to the system and is therefore not imported. The chosen scenario consists of a mission with the start and target positions P1 and P10.

A. The Generated Roadmap

The scenario in Figure 3 is processed to the roadmap shown in Figure 6. Each cell of the roadmap contains the produced robot positions in configuration space, as explained in Section III.B.

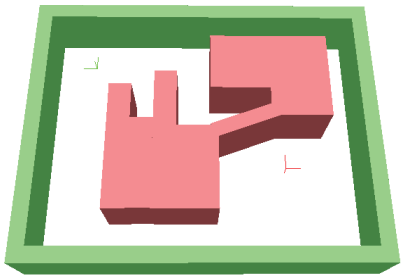


Figure 3: Illustration of the experimental scenario in the 3D world.

B. Corridor of Robot Space Positions

Using the generated roadmap in IV.A, the resulting corridor is given in

Figure 4, including the indicated configuration space positions. The search is executed on those positions and it finds a path, as illustrated. Configuration space positions are also added to the start and target positions including their extended cells, as explained in Section III.B.

C. Elastic Net Trajectory Generation

In Figure 5, the found path is processed and adapted to a feasible trajectory, shown in Figure 6. The elastic net algorithm is parameterized regarding its shrink forces. Those forces (shown as arrows in Figure 5) move the particles on a straight line and, thus, push the trajectory to the obstacles. The stronger the force, the more the trajectory is moved towards the obstacles and the more collisions may occur. The path planning system first controls the real robot along a trajectory with low shrink forces applied to reduce the number of collision indications. The real robot may now be controlled along the generated trajectory until a collision is indicated or the target is reached. After the final trajectory is found, the shrink force may be raised to optimize the trajectory.

D. Re-planning on Collision Indications

As mentioned in IV.A, the search is executed within the roadmap corridor containing configuration space positions of the robot. During the movement execution of a solution, new information about the workspace and the obstacles may be added to the world model. This normally happens when collisions are indicated. With a dynamic update of the world model, a new search is initiated.

The real robot stops its previous movements, moves back to the last common trajectory position and follows the new trajectory. Figure 7 illustrates the environment exploration and the resulting world model updates to recognize obstacle O3. As a result, the Voronoi roadmap plans a new trajectory around the newly added obstacle location. It turns out that those locations are also occupied and therefore a completely

new trajectory is found, as shown in Figure 8, which is further modified as described in IV.A.

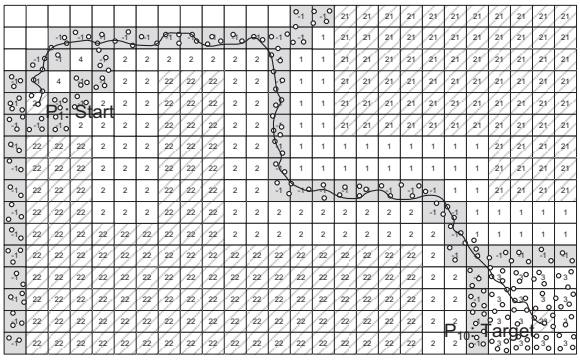


Figure 4: Roadmap corridor including configuration space positions.

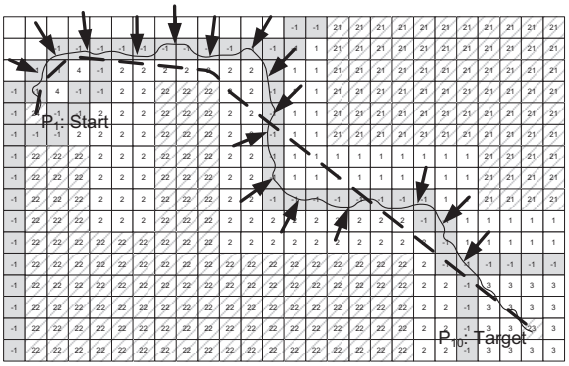


Figure 5: Elastic net trajectory generation.

E. Trajectory Planning Workflow and Robot Behaviour

The Mitsubishi RV-2AJ robot moves slowly along the planned trajectories in real time. The world model is updated on each collision indication (for dynamic and static obstacles) and does not interrupt the trajectory planning workflow. The clearance of the generated trajectories to obstacles is kept, and time consuming re-planning is rarely executed. Although unknown obstacles require re-planning more often, it still does not interrupt the workflow. It may be optimized by introducing occupancy probabilities to neighboring octree cells to reduce the number of re-planning occasions and required collision indications.

V. DISCUSSION AND CONCLUSION

The general research aim was to establish a robot programming support system which helps the robot operator to generate robot programs in an industrial production environment. The trajectory planning system is an important component and it has to satisfy all requirements as stated in the introduction. The framework is defined by the employed algorithm and the usage of the robot programming support system in real environments, together with the limited sensory input. The achieved algorithm employs Voronoi roadmaps in the first instance. This allows a high probability for collision free movements of the robot through the workspace considering a minimum knowledge of obstacles within the environment. The Voronoi roadmap supports path planning with only little sensory input, which is most often obtainable in real environments. While the robot is moving along the Voronoi path, collision information indicated by the operator or other sensors is used to improve the roadmap and, thus, exploration of the environment takes place.

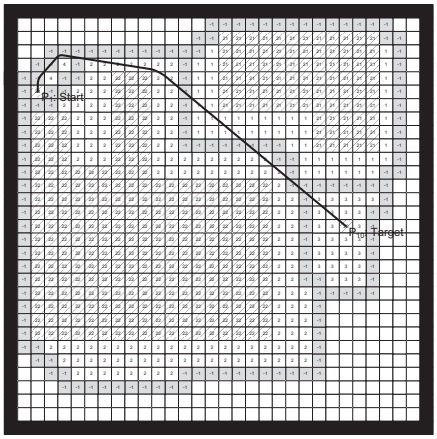


Figure 6: Trajectory through the roadmap without obstacle O3.

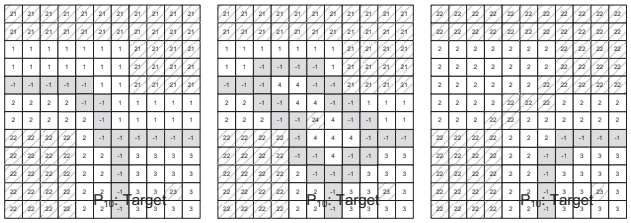


Figure 7: Every collision indication position (part of obstacle O_3) is added to the roadmap.

The application of the elastic net not only transforms the found path into a trajectory, it also optimizes that trajectory. It deforms and stretches the path to reduce the clearance to the obstacles and thereby the world model is updated. This is an important feature to stretch or shorten the generated trajectories along Voronoi edges, which are otherwise not short and smooth. The operator might directly control the elastic stretching process in future work and has been chosen by experiment in the proposed approach.

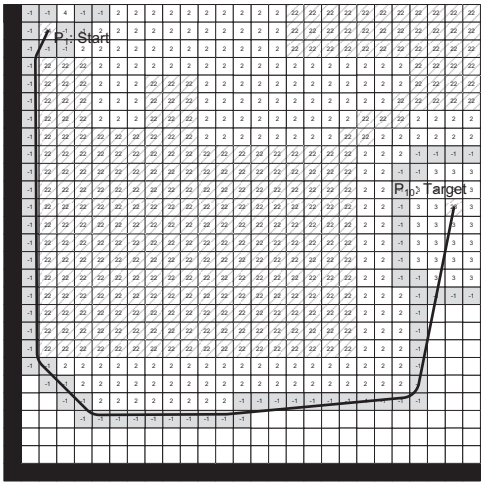


Figure 8: The new trajectory.

Shortest path planning is executed on points along the Voronoi edges and optimized in a second stage to generate the trajectory. Although other solution candidates might be

shorter after optimization, this approach presents a good approximation. This two stage approach allows the use of low accuracies in the search stage, which speeds up the algorithm. The accuracy of the octree controls the capability of the path searching algorithm to find small passages. The robot configuration space discretization in the pre-calculation step is optimized for the accuracy. Too many discretization positions lead to long path planning times, whereas too few positions prevent the path planner to find a solution.

This approach considers world and joint coordinates and joins them in the octree. The transition is an important step, since inverse calculations of target positions for articulated robots often result in non-singular robot postures. Reported collisions occur in a single posture and, thus, postures have been stored within the octree cells for obstacle avoidance.

The presented methodology considers static obstacles. An extension to dynamic obstacles requires two collision indication buttons to classify dynamic and static obstacles. Dynamic obstacles that have the same state in each time step for each production cycle can be supported. These are the only obstacle types required for the defined industrial production scenario.

REFERENCES

[1] Y. Demiris, A. Billard, Special Issue on Robot Learning by Observation, Demonstration, and Imitation, 37 (2007) 254-255.

[2] L. Qi, X. Yin, H. Wang, L. Tao, Virtual engineering: challenges and solutions for intuitive offline programming for industrial robot, in: Proc. IEEE Conf. Robotics, Automation and Mechatronics, 2008, pp. 12-17.

[3] Z. Pan, J. Polden, N. Larkin, S.V. Duin, J. Norrish, Recent Progress on Programming Methods for Industrial Robots, Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK), (2010) 1-8.

[4] H. Chen, T. Fuhlbrigge, X. Li, Automated industrial robot path planning for spray painting process: A review, in: Proc. IEEE Int. Conf. Automation Science and Engineering CASE 2008, 2008, pp. 522-527.

[5] R. Geraerts, M.H. Overmars, A comparative study of probabilistic roadmap planners, in: IN: WORKSHOP ON THE ALGORITHMIC FOUNDATIONS OF ROBOTICS, 2002, pp. 43-57.

[6] Y. Koren, J. Borenstein, Potential Field Methods and Their Inherent Limitations for mobile robot navigation, in: Proceedings of the IEEE Conference on Robotics and Automation, Sacramento, California, 1991, pp. pp. 1398-1404.

[7] C.W. Warren, Global path planning using artificial potential fields, (1989).

[8] Y. Kitamura, T. Tanaka, F. Kishino, M. Yachida, 3-D path planning in a dynamic environment using an octree and an artificial potential field, in: Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on, 1995, pp. 474 -481 vol.472.

[9] I. Gargantini, Linear octtrees for fast processing of three-dimensional objects, Computer Graphics and Image Processing, 20(4) (1982) 363-374.

[10] J.Y. Hwang, J.S. Kim, S.S. Lim, K.H. Park, A fast path planning by path graph optimization, Systems, Man and Cybernetics, Part A, IEEE Transactions on, 33 (2003) 121-129.

[11] P. Bhattacharya, M.L. Gavrilova, Roadmap-Based Path Planning - Using the Voronoi Diagram for a Clearance-Based Shortest Path, IEEE Robotics \& Automation Magazine, 15 (2008) 58-66.

[12] K. Hoff, III, T. Culver, J. Keyser, M.C. Lin, D. Manocha, Interactive motion planning using hardware-accelerated computation of generalized Voronoi diagrams, in: Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on, 2000, pp. 2931-2937 vol.2933.

[13] H. Ingaki, K. Sugihara, N. Sugie, Numerically robust incremental algorithm for constructing three-dimensional Voronoi diagrams, in: Proc. 4th Canad. Conf. Comput. Geom., 1992, pp. 334--339.

[14] J. Kim, F. Zhang, M. Egerstedt, An Exploration Strategy Based on the Constructing Voronoi Diagrams, in: IEEE Conference on Decision and Control \& Chinese Control Conference, 2009.

[15] J. Vleugels, M. Overmars, Approximating Generalized Voronoi Diagrams in Any Dimension, in, 1995.

[16] J.M. Vleugels, J.N. Kok, M.H. Overmars, Motion Planning Using a Colored Kohonen Network, (1993).

[17] S. Fortune, A sweepline algorithm for Voronoi diagrams, in: Proceedings of the second annual symposium on Computational geometry, ACM, Yorktown Heights, New York, United States, 1986, pp. 313-322.

[18] E. Masehian, M.R. Amin-Naseri, A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning, J. Robot. Syst., 21 (2004) 275-300.

[19] C. Kohrt, G. Schiedermeier, A.G. Pipe, J. Kiely, R. Stamp, Nonholonomic Motion Planning by Means of Particles, in: Proc. IEEE Int Mechatronics and Automation Conf, 2006, pp. 729-733.

[20] D.L. Hall, J. Llinas, An introduction to multisensor data fusion, Proceedings of the IEEE, 85 (1997) 6 -23.

[21] J. Yang, P. Dymond, M. Jenkin, Exploiting Hierarchical Probabilistic Motion Planning for Robot Reachable Workspace Estimation, in: J.A. Cetto, J. Filipe, J.-L. Ferrier (Eds.) Informatics in Control Automation and Robotics, Springer Berlin Heidelberg, 2011, pp. 229-241.

[22] F. Zacharias, C. Borst, G. Hirzinger, Capturing robot workspace structure: representing robot capabilities, in: Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on, 2007, pp. 3229 -3236.

[23] J.H. Reif, H. Wang, Nonuniform Discretization for Kinodynamic Motion Planning and its Applications, SIAM Journal on Computing, 30 (2000) 161-190.

[24] D. Henrich, C. Wurll, H. Worn, Online path planning with optimal C-space discretization, in: Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on, 1998, pp. 1479-1484 vol.1473.

[25] P. Bhattacharya, M.L. Gavrilova, Roadmap-Based Path Planning - Using the Voronoi Diagram for a Clearance-Based Shortest Path, IEEE Robotics & Automation Magazine, 15 (2008) 58-66.

[26] A.K. Garga, N.K. Bose, A neural network approach to the construction of Delaunay tessellation of points in Rd, Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 41 (1994) 611 -613.

[27] P. Bhattacharya, M.L. Gavrilova, Voronoi diagram in optimal path planning, in: Proc. 4th Int. Symp. Voronoi Diagrams in Science and Engineering ISVD '07, 2007, pp. 38-47.

[28] L.E. Dubins, On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents, American Journal of Mathematics, 79 (1957) 497-516.

[29] S.-J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach (2nd Edition), Prentice Hall, 2002.

[30] P. Payeur, Improving robot path planning efficiency with probabilistic virtual environment models, (2004).

[31] C. Kohrt, A. Pipe, G. Schiedermeier, R. Stamp, J. Kiely, A robot manipulator communications and control framework, in: Proc. IEEE Int. Conf. Mechatronics and Automation ICMA, 2008, pp. 846-851.

KOHRT, C., PIPE, A., SCHIEDERMEIER, G., STAMP, R. and KIELY, J. (2011) *An Online Robot Trajectory Planning and Programming Support System for Industrial Use.* Journal of Robotics and Computer-Integrated Manufacturing.

(This is a sample cover image for this issue. The actual cover is not yet available at this time.)

Robotics and Computer-Integrated Manufacturing 29 (2013) 71–79

Contents lists available at SciVerse ScienceDirect

Robotics and Computer-Integrated Manufacturing

journal homepage: www.elsevier.com/locate/rcim

An online robot trajectory planning and programming support system for industrial use

C. Kohrt [a,b,*], R. Stamp [a], A.G. Pipe [a], J. Kiely [a], G. Schiedermeier [b]

[a] Department of Engineering, Design and Mathematics, UWE—University of the West of England, BS16 1QY Bristol, UK
[b] Department of Computer Sciences, UASL—University of Applied Sciences Landshut, 84036 Landshut, Germany

ARTICLE INFO

Article history:
Received 26 September 2011
Received in revised form
3 June 2012
Accepted 26 July 2012

Keywords:
Robot
Path
Planning
Support
Program
Generation

ABSTRACT

The manufacturing industry today is still looking for enhancement of their production. Programming of articulated production robots is a major area for improvement. Today, offline simulation modified by manual programming is widely used to reduce production downtimes but requires financial investments in terms of additional personnel and equipment costs. The requirements have been evaluated considering modern manufacturing aspects and a new online robot trajectory planning and programming support system is presented for industrial use. The proposed methodology is executed solely online, rendering offline simulation obsolete and thereby reduces costs. To enable this system, a new cell-based Voronoi generation algorithm, together with a trajectory planner, is introduced. The robot trajectories so achieved are comparable to manually programmed robot programs. The results for a Mitsubishi RV-2AJ five axis industrial robot are presented.

1. Introduction

Robot use and automation levels in the industrial sector will grow in future, driven by the ever-present need for lower item costs and enhanced productivity. Synonymous with this projected increase will be the requirement for capable programming and control technologies. Many industries employ offline programming within a manually controlled and specified work environment. This is especially true within the high-volume automotive industry, particularly when related to high-speed assembly and component handling. Any scenario, reliant on manual data input, based on real world obstructions, necessitates the complete production system being offline, out of production, for appreciable time-periods while data is input. A consequent financial loss ensues.

The two main categories of robotic programming methods are online programming and offline programming. Usually, the teach pendant is used to manually move the end-effector to the desired position and orientation at each stage of the robot task. Relevant robot configurations are recorded by the robot controller and a robot program is successively written to command the robot to move through the recorded end-effector postures. Offline programming is based on models of the complete robot work cell and the robot is simulated.

Published research appears to be concentrated on the application of simulation tools to generate discrete portions of the total robot trajectories [17], whilst necessitating manual input to link paths associated with one particular activity with those of another. Human input is needed also to correct inaccuracies and errors resulting from unknowns and falsehoods in the environment.

Investigations have been undertaken with the aim of generating a robot control program, by considering the working production environment as a single, whole, workspace. Use is made of automated workspace analysis techniques and trajectory smoothing. Some non-productive time is necessitated, but unlike previously reported approaches, this is, for the most part, achieved automatically and consequently rapidly. As such, the actual cell-learning time is minimal.

2. Industry requirements to an online robot trajectory planning and programming support system

An up-to-date industry requires a modern production system, able to combine and support flexibility, high-speed and optimization [11]: the overall production time available must be maximized to guarantee the highest productivity possible. Considering the high level of complexity of several robot-programming tasks for human operators, the proposed solution consists in a support system that takes over all the most complicated sub-tasks. The left manageable

* Corresponding author at: University of the West of England, Department of Computer Sciences, Faculty of Computing, Engineering & Mathematical Sciences, Coldharbour Lane, Bristol BS16 1QY, UK
E-mail addresses: christian@kohrt.org (C. Kohrt), richard.stamp@uwe.ac.uk (R. Stamp), anthony.pipe@uwe.ac.uk (A.G. Pipe), janice.kiely@uwe.ac.uk (J. Kiely), gschied@fh-landshut.de (G. Schiedermeier).

0736-5845/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.rcim.2012.07.010

sub-tasks related to the given mission remain responsibility of the operator. The proposed methodology is executed solely online rendering offline simulation obsolete and thereby reduces costs for the offline preparation of robot programs. Supported online programming must be fast and flexible to reduce possible production downtimes. The generated trajectories must conform to the given requirements in quality and speed. Physical production parts and fixtures are often not available during online robot programming, thus, the support system must handle such situations to permit its use. Nevertheless, the human component still remains important and necessary: robot programs may be modified by human operators during their lifecycle because of possible changes. The so generated programs must be readable, maintainable and changeable.

Fig. 1. Devices overview.

3. Support system overview and architecture

The proposed support system is applied on a 5-axis industrial-scale, articulated Mitsubishi RV-2AJ robot with an additional Ethernet card installed. It is a nonlinear system with five rotary joints. The robot is equipped with the Mitsubishi CR1 controller and a teach pendant. The main areas of the robot are assembly, manufacture, pick & place and handling tasks. Communication between this system and a personal computer is possible [12]; the commercial viability has already been demonstrated [16]. The equipment is shown in Fig. 1. Both model-based and sensor-based data are considered in order to define the environment of the robot: perception functions, initiated by sensors (cameras or tactile sensors), provide the system with information about the environment. A general overview is given in Figs. 2 and 3.

The *Object Recognition* component converts the features of an image into a model of known objects. First, the scene is segmented into distinct objects. An analysis deriving from motion, binocular stereopsis, texture, shading and contour follows, so that orientation, shape and position of each object may be determined relative to the camera. Peter Corke's Machine Vision Toolbox for Matlab [4,5] allows the developer to use professional image processing capabilities [20] with ease. In addition, model-based data such as computer-aided design (CAD) data is used to present the world model more accurately. Computer-aided design derived data from simulation systems, such as RobCAD [24], are exported as drawing exchange format (DXF) files, including all locations attached and they are stored within the world model. The attached locations of computer-aided design objects are employed to acquire information concerning start, target and application paths locations.

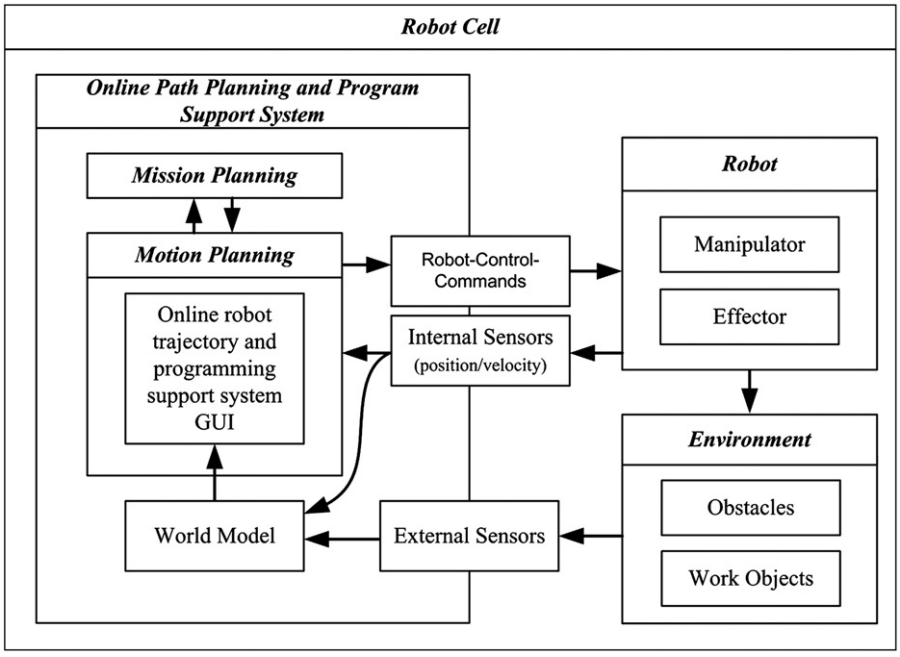


Fig. 2. Logical view of the support system.

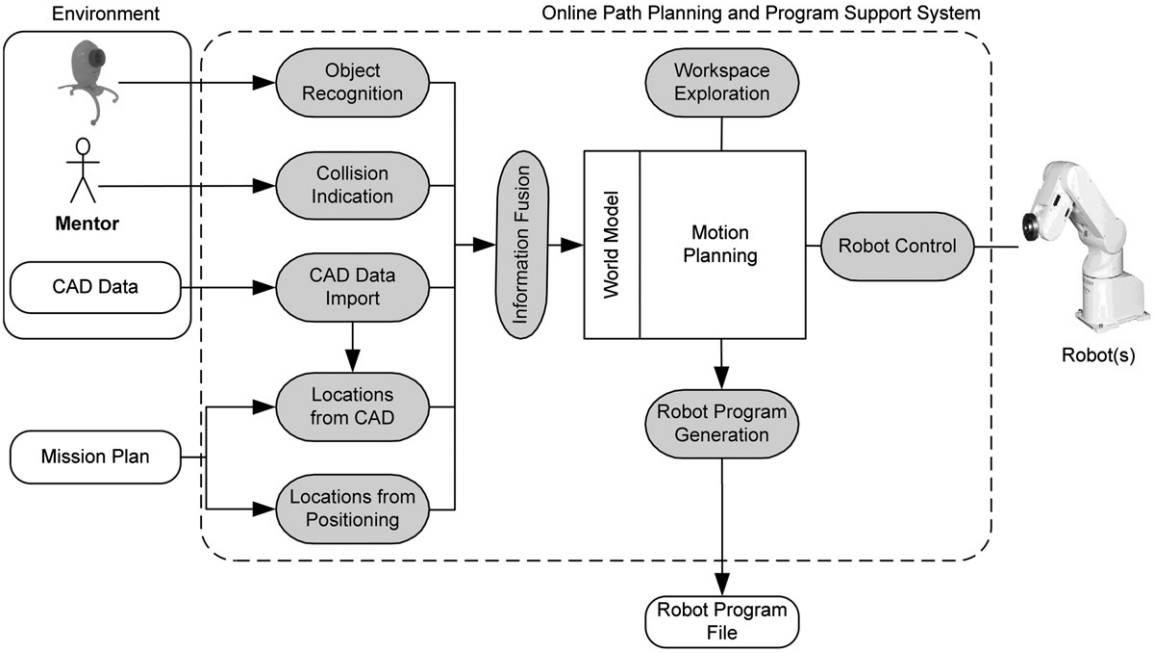


Fig. 3. Online path planning and programming system overview.

Target locations may also be determined by a manual movement of the robot or by visual servo-control. The first is possible with the support of a joystick or a teach pendant. The visual servo-control applies a pointing device to indicate the target location to the robot. The robot moves automatically towards the given location with the help of a neural network and stores the position. Successively, the network transforms picture coordinates into robot control commands as described in [18].

The *Workspace Exploration* component gathers additional environmental information, by direction of the robot to move within the workspace. This is realised through manual movements, random movements and existing robot programs. Collisions are always processed during exploration. Throughout its movements, the robot ascertains which parts of the space are free within its coordinate system, by either visual feedback or manual collision indication. This information becomes more accurate during the planning process.

The *Robot Control* component [12] grants the direct communication with the robot and enables direct robot control, serial/Ethernet connection, robot parameter editing/reading/writing, program uploading and downloading, real-time movement control, robot system backup/restore, external control over the user datagram protocol (UDP) and equipment control.

The robot path is completely stored within the support system in form of a trajectory that consists of connected particles. Its transfer to a robot specific program is achieved within the *Robot Program Generation* component in two steps: first, translation into a robot program of solely the provided trajectory; second, generation of the specific robot program enriched with additional configuration commands and specific linguistic syntax. The here described two-step generation may also be applied to support other robot types.

4. The online path planning and programming support system

Deriving from the requirements described in Section 2, a method is necessary to combine maintainability and optimality, i.e., shortest path finding and path smoothing. The proposed path planning system identifies a trade-off between both. Path finding and smoothing are actually two competitive tasks, considering also that smoothing is generally applied after the definition of the path itself. The proposed path planner allows on the contrary the concurrent execution of both tasks. Optimality here is identified in form of the trajectory length.The so generated trajectory consists only of a small number of locations and movement primitives (linear, joint and circular movements). This approximation renders the robot program maintainable, clearly structured and understandable by human robot programmers.

4.1. Overall algorithm

The content of the following paragraphs is based on the path planning workflow shown below:

(1) Set up online path planning and program generation support system including hardware.
(2) Import pre-existing data (robot geometry and computer-aided design data).
(3) Create a mission by robot movements, computer-aided design locations, pointing device or simulation.
(4) Execution of the support system.
    a. Create connectivity in form of an approximated Voronoi form.
    b. Explore the workspace and update the world model.
        i. Automatic random exploration.
        ii. Exploration by existing robot programs.
        iii. Exploration by following the Voronoi lines to the target without path smoothing.
    c. Apply the path searching algorithm in joint space.
    d. Apply the elastic net algorithm to generate the trajectory.
    e. Move along the trajectory from start to the target until either a constraint violation occurs (collision or robot kinematic constraint), a shorter path is found by the path searching algorithm or the target is reached.
        i. On collision or kinematic constraint violation.
            1. Update the roadmap and generate new roads.

2. Take back the last movement to the last common trajectory position that is unchanged.

ii. On a shorter path found in the roadmap.
1. Continue the movement to explore the workspace along the possible trajectory solution until the path length difference is larger than a hysteresis value.
2. When the path length difference is larger than the hysteresis value, do an automatic random exploration.
3. Take back movement to the last common trajectory position of the old and new trajectory and continue with 4e).

(5) Robot program generation.
(6) Robot program file upload to the robot.
(7) Remove the support system.

4.2. General workflow

A general case of the workflow is illustrated in Fig. 4 and will be described in the following sections. It may be recognised within the world model, but also within the online robot trajectory planning and programming support system (Fig. 3).

The workflow consists mainly of four actions: the linear octree [9] stores robot environment data. The application of the data creates a roadmap in form of a Voronoi diagram, in three-dimensional space and with a new cell-based methodology. The A* search algorithm is applied on joint space positions within the roadmap. It is a famous shortest path-finding algorithm [19] that uses heuristics to direct the search towards the target. The heuristic shall never overestimate the distance to the goal. Therefore, the joint distance is appropriate. With the support of the trajectory generation module (within the elastic net), a so deriving path is transformed into a trajectory.

4.3. World model

Path planning is based on data about the physical environment, the so-called world model. Pre-existing and dynamic sensor-information of the environment is stored into this specific model, defined by three main sources: computer-aided design data of a construction process of the working cell, a vision-system and the human operator. During the execution of the path planner, the operator is given the possibility to indicate a collision point through a specific button for example on the control panel or the joystick.

Both, the positions in the world model and their occupations are of interest, therefore a flag shows whether a position is safe or not. Real robot applications have demonstrated that sensors may deliver wrong information [10]: in this model, a reliability weight between 0.0 and 1.0 is defined for each information source. The world model is able to handle this additional information and combines the information types mentioned above.

The deriving sensor fusion includes sensor abstraction, algorithms and architectures [3]. Fusion is carried out as a voting system. Each sensory source is filtered through a simple moving average (SMA) filter, which delivers cohesive information. The reliability weight of each source affects the calculation of the coordinates occupation with the averaged weighted sum of the sensor values. The resulting probability of the occupation value rises with every check. Not only real obstacles are considered, but also the kinematic of the robot, consequently involving areas in space otherwise unconsidered.

4.3.1. Pre-existing and dynamic information

The world model stores information concerning the robot cell, the used robot and the environment in form of computer-aided design data. Such pre-existing information is considered beforehand. The robot type information is particularly important since it allows the use of a simulation model of the robot to afford forward and inverse kinematic calculations. These calculations and computer-aided design data stored in the world model become usable for the path planning system. Both pre-existing and dynamic information deriving from the sensors is adopted: the operator gives valuable information about upcoming collisions; a vision system delivers information about the robot position and possible collisions. Obviously, not only collisions are interesting, but also information about the position of obstacles and 'holes' in the configuration space not recognisable from the robot itself.

4.3.2. Voronoi based roadmap generation

Roadmap methods generally identify a set of roads, in graphic form, which may be safely travelled along without incurring into obstacles. The method here adopted has been inspired by the approach presented by Vleugels, Kok and Overmars [21], based on the Voronoi form [2,8]. This choice has been taken also considering two important aspects: first, the Voronoi form may be applied either in the workspace or in the configuration space of the robot. Second, it maximizes the clearance of obstacles, so that the path-planning algorithms have not to be particularly accurate. This second point may be perceived as a negative characteristic too, since the so deriving roads are not short, smooth and continuous enough to guarantee an enhancement [1,15]. In fact, implementation tests of [21] have shown that a Voronoi form is rarely reached. Adjustments of the parameters by trial and error, as suggested by the authors of [21], have not led to any different results too.

Hence, the concept at the basis of the Voronoi form has been extended and applied to a grid-based algorithm. A simplified, two-dimensional space is illustrated in Fig. 5: several obstacles, a configuration space 'hole', start and target cells are represented. The light grey cells '−1' reproduce the Voronoi approximation. The dark grey cells represent the configuration space 'hole'. White cells denote expanded nodes, 1–7 denotes expanded obstacle node cells, black cells denote border nodes, and 21–23 denote obstacles. The general grid-based algorithm consists of a simple rule, applied to produce the approximated Voronoi diagram. The primary aim is to approximate a Voronoi form between the obstacles and the border cells in the configuration space. The configuration space 'holes' are considered as obstacles which preclude the Voronoi form the possibility to maximize the clearance to physical objects.

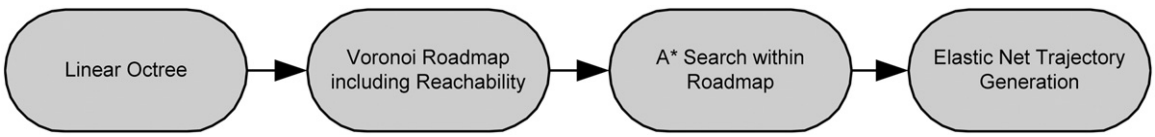


Fig. 4. General path planning workflow.

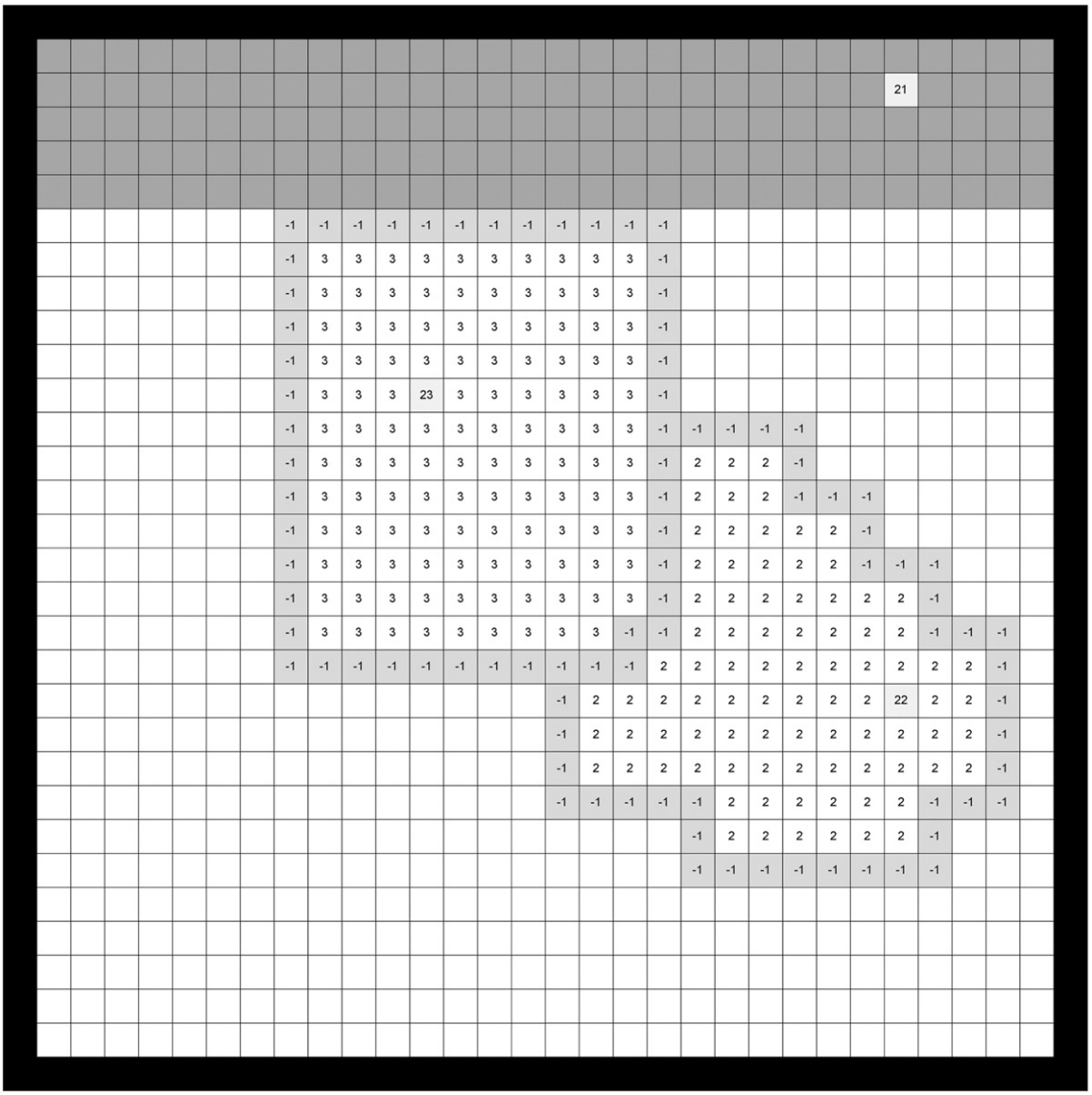


Fig. 5. Voronoi approximation in a two-dimensional uniform grid.

4.3.2.1. Rule

All direct neighbours of the cells C_i are extended. If the extended cell is located between two or more obstacles or border cells and cannot be reached by the robot, its number is set to '0'. If the extended cell is located between two or more obstacles but it is reachable by the robot, its number is set to '-1', the Voronoi approximation. In any other case, the extended cell is set to the number of the originating cell C_i.

The grid used in the implementation is an octree in three dimensions. It allows adding obstacles during runtime while only the neighbouring areas will be necessary to recalculate. The octree also provides the opportunity to use its hierarchy to speed up the algorithm. Applying this cell extension methodology, roadmaps are built so that an on-going, real time development of the topology and connectivity of the robot workspace is possible.

Roadmaps are employed by the path planning system during the mission planning in order to identify paths between two positions (see Section 4.4) and to define the trajectory (see Section 4.5) [13].

4.4. Mission planner

Given a mission, the mission planner plans multiple possible trajectories. This choice has been taken for two main reasons. First, the industry today requires a support system able to accomplish different tasks and contemporary requirements. Second, applying the here presented path planner, the system results highly optimized. Distances are calculated in shorter time (see also Section 5), the range of collected information used by the mission planner is higher and the multiplicity of possible trajectories is maintained. Any algorithm for solving the travelling salesman problem [19] may be utilised to calculate the order in which each application path is processed. The mission planner delegates the task of trajectory planning to the path planner. Both, mission and path planner have to establish an interconnection for the exchange of information that is the length of the actual planned path. The algorithm here used must be able to handle path length information during path planner execution and react to this by commanding the path planner. In the system now discussed, a simple brute force algorithm has been used and as such, it is admirable for a demonstration

system, although it is limited to operation with few application paths.

4.5. Path planner

As described in Section 4.4, path planner and mission planner must interact. While the path planner is focused on the creation of the trajectory, the mission planner handles a higher level of the planning. A mission is defined by the start- and target-locations of the paths, combined with path application information, for example handling, adhesive bonding or painting. The path planner calculates a path and controls the robot along that path until a collision is detected, the kinematic constraints are not met or the target is reached. In each case, the mission planner is informed by sensor inputs and acts appropriately by initiating the roadmap and trajectory generation algorithm.

Fig. 6 shows the path planner together with its interfaces. It is defined in the Enterprise Architect UML tool and the hull including the connection to the communication framework is automatically generated as Java code.

The path planner interfaces with the following systems: control port (component life cycle), robot position (actual robot position), mission planner (mission information), environment model (sensor input), robot kinematic (forward and inverse robot geometry calculations) control application (path planning user control), robot movement control (direct movement execution) and world model (topology of the workspace through connected roads).

The robot is considered as a Dubins airplane [7], steered from the start to the target by real robot movements. Given the target, the path planner identifies the shortest path within the roadmap. During the planning of the trajectory, an improvement of the roadmap takes place: data about the environment are collected and obstacles within the configuration space are better approximated. Finally, a joint distance metric is determined as heuristic for the A* algorithm and the path is converted into a trajectory able to avoid static obstacles and to reduce the clearance to them.

Transformation is therefore a necessary step and it is realised through the application of an elastic net. The roads of the roadmap, identified by the A* algorithm, consist of connected configuration space positions. Those positions create a Voronoi diagram (considering the free area in the joint space) and the elastic net. Transformation of the elastic net into a trajectory is achieved by applying equidistance, rotation and shrink forces on the joint space positions [13]. The result is a trajectory formed by canonically ordered movement primitives, which are linear, circular or joint movements. Joint movements are not of interest in this study and are therefore omitted. Moreover, the transformation process takes the configuration space 'holes' (by kinematic calculations) and the obstacles (by collision detection) into consideration.

Finally, the A* algorithm leads to the identification of the shortest path and this often generates a re-planning of the path itself if a shorter path is recognised. Since real robot movements are involved, this should not happen too often. A hysteresis is applied in order to prevent this. This application has been included in order to allow an additional exploration of the workspace: consequently, the system may rely on a wider knowledge provided to the world model.

4.6. Robot program generator

As described in Section 4.5, a trajectory is composed of movement primitives. Movement primitives are in its turn composed of a list of particles, mainly linear, circular and joint movements. Each particle forming the movement primitive knows its own position (stored in Cartesian coordinates) and orientation. Successively the robot program generator transfers the given set of particles to a robot readable format, either robot program files, for example in Melfa Basic language, or direct movement commands transferred to the robot controller.

5. Experiment

The online path planning and programming support system proposes an approach able to reduce the robot programming time including preparation and installation. It generates acceptable robot programs and regards the modern industrial basic goals (flexibility, speed and optimization; see also Section 2). It finds a trade-off between shortest path finding and trajectory forming and maintainability. Finally, it generates a downloadable robot program file.

In this section, the general execution of the support system is described and a scenario (see Fig. 7) has been chosen to illustrate the proposed approach.

5.1. Pre-existing data import

In the chosen scenario the two obstacles, O_1 and O_2, are given as drawing exchange format files and imported to the environment model. One obstacle O_3 is 'unknown' for the system (not imported).

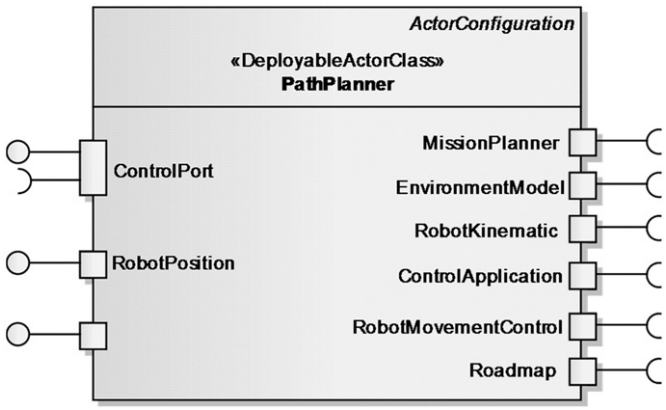


Fig. 6. Path planner interfaces.

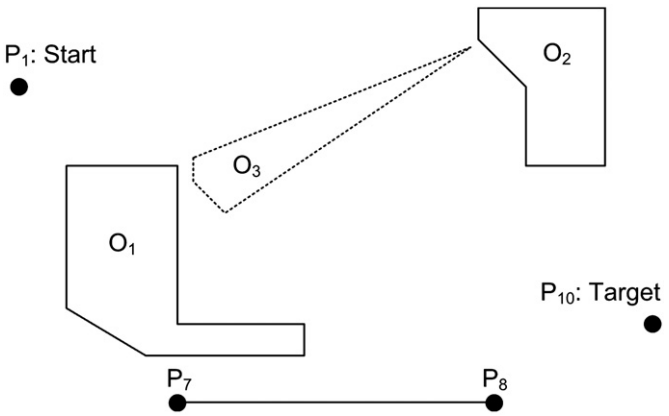


Fig. 7. Experimental scenario (2D example in 3D world).

5.2. Mission preparation

The chosen scenario consists of a mission with positions P_1 and P_{10} and the application path P_7 to P_8, a straight line with the hand tool equipment of the robot closed. The pointing device has been used to store the locations of the application paths in co-operation with the support system.

5.3. Robot program generation

Once the mission has been planned successfully, the robot program file may be generated. The program generation is template-based. Thus, only the dynamic content of the file is shown in Listings 1 and 2 and Listing . The results are shown in Figs. 8 and 9.

The movement primitives circular and linear are identified, respectively as *MVR* and *MVS* robot commands. The program in Listing 2, composed of 6-movement primitives, is still readable by a human. The final movements of the robot are comparable to the manually programmed ones. Manual modifications may still be carried out within the program even on larger missions.

The overall time for the proposed system to generate a robot program file for the scenario was about 20 min (see Table 1, row 9), including mission preparation, data import and program file generation. The proposed system has been compared with offline programming and conventional online programming. Both programming methods include the use of tools such as the Mitsubishi programming tool COSIROP/MELSOFT [16] or RobCAD. Offline programming and conventional online programming requires highly skilled operators, while only a basic knowledge is required

```
10 MOV P2
20 MVR P2,P3,P7
30 HOPEN 1
40 MVS P8
50 MVR P8,P9,P10
```

Listing 1. Melfa Basic IV programmed file created online .

```
10 MOV P2
20 MVR P2,P3,P4
30 MVS P5
40 MVR P5,P6,P7
50 HOPEN 1
60 MVS P8
40 MVR P8,P9,P10
```

Listing 2. Melfa Basic IV robot program file created with the support system .

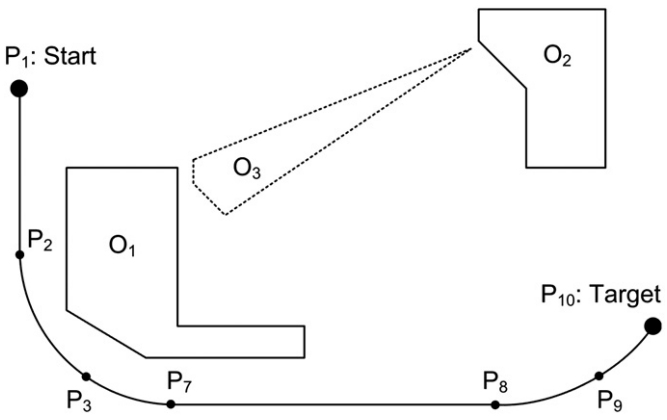


Fig. 8. Manually planned path (schematic).

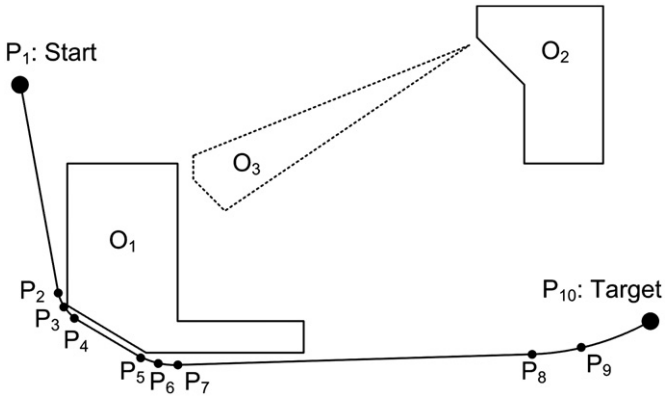


Fig. 9. Automatically planned path (schematic).

for supported robot programming. Online programming only considers available physical objects whereas offline and supported programming support models of these objects. The time for each process step is given in Table 1:

The times within Table 1 may be divided into fixed and task-dependent times. Usually within an industrial setting, there is not the necessity to place numerous models into the workspace; therefore, they may be seen as fixed. Moreover, setting the locations should not be relevant too, though program generation is highly dependent on the size of the program (see rows 4–6 in Table 1).

Table 2 illustrates the online programming time only and Table 3 represents the overall programming time for each programming method. Offline programming must be separated into minimum, maximum and normal values, which represent the online modification of the offline generated program within the robot cell. The normal values may vary within the minimum and maximum values, depending from the quality of the offline generated robot program. Online programming can be applied very quickly and should be used for small program sizes since the programming time deeply increases compared to the program size. Supported online programming requires an equal amount of time and a small fixed installation time compared to normal values of the offline programming method.

Table 3 focused on the preparation times and it shows an additional preparation time for offline programming also mentioned in Table 1, row 0. The offline preparation time can be omitted entirely to save offline programming expenses, since the speed of programming, comparing offline and supported online programming is equal. Certainly, this is highly dependent from the quality of offline generated programs and may affect the "offline (normal)" values in Tables 2 and 3. In the given small example scenario, 2 h offline programming including the operator and the simulation tool could be omitted saving costs. Therefore, supported online programming is recommended especially for small batched manufacturing but also for high-volume production.

6. Discussion

The complexity of programming remains one of the major hurdles preventing automation using industrial robots for small to medium sized manufacturing. Offline programming with a simulation system has been introduced for large volume manufacturing but the additional efforts in offline programming makes it inefficient for small to medium sized manufacturing. Although online programming methods have been researched in the past [25] to make online programming more intuitive, less reliant on

Table 1
Path planning execution times (* if locations are stored within DXF).

Id	Program execution time**Task	Online [s]	Supported [s]	Offline [s]
0	Offline programming	0	0	7200
1	System installation	10	600	10
2	DXF import	0	30	0
3	DXF placement	0	300	0
4	Set start/goal locations	60	60/0*	0–60
5	Set application locations	60	60/0*	0–60
6	Program or modify path	240	60	0–240
7	Save/upload program	20	10	20
8	Sum (online):	390	1120/1000	30–390
9	Sum (overall):	390	1120/1000	7230–7590

Table 2
Comparison of the online programming times.

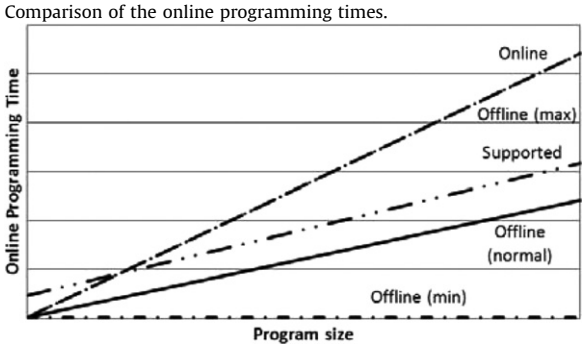
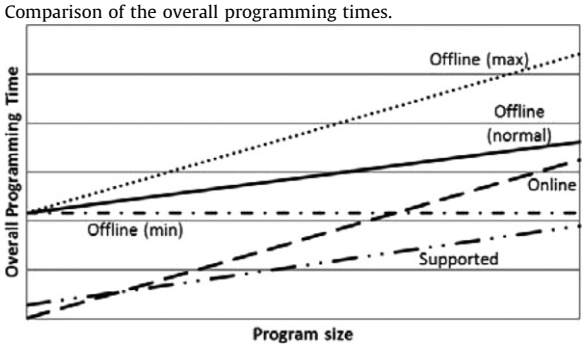


Table 3
Comparison of the overall programming times.



operator skill and more automatic, most of the research outcomes have not become commercially available. This is partially because most of these methods are limited to their specific setups and are yet to be applied to general applications. Compared to those methods, the presented methodology differs in two points. First, the human operator reports collisions and, thus, it is generally available and cost efficient. Second, the applied trajectory planning algorithm is able to handle the information type and intelligently controls the robot within the robot cell to compute the robot trajectory.

7. Conclusion

Aim of this paper was to introduce a new online path planning and programming support system. The tool is applicable to real industrial scale, where articulated robots work in multi-dimensional space. One of the main benefits deriving from this application is its real time capability. Creating the opportunity to work successfully online, offline simulation systems becomes obsolete; moreover, the overall time required for larger missions decreases. This support system is based on two specifics: the Voronoi roadmap and the elastic net, both co-operating for the planning of missions with multiple goals. The new approach transforms the user interaction into a simplified task that generates acceptable trajectories, applicable for industrial robot programming. In addition, it works successfully with basic knowledge of the operator and asks to use the software application only. The trade-offs optimality, path planning & smoothing and maintainability are considered in the new approach. The new criteria maintainability and reusability have been introduced and the shown experiment has demonstrated that the system successfully faces and satisfies the modern requirements coming from the industrial market. The process is optimized, offline programming time may be saved and online programming becomes easier. Nevertheless, there is still space for further development, concerning dynamic obstacle avoidance and application of the system to multiple robots working conjunctly. The standard A* algorithm here used may, in the future, be extended to the AD* algorithm [14]. Mission and task specific extensions to the software have not been incorporated yet. These are for example application path information for welding, adhesive bonding or handling.

References

[1] Bhattacharya P, Gavrilova ML. Voronoi diagram in optimal path planning, In: Proceedings of the fourth international symposium Voronoi diagrams in science and engineering ISVD, 2007, pp. 38–47.
[2] Bhattacharya P, Gavrilova ML. Roadmap-based path planning—using the Voronoi diagram for a clearance-based shortest path. IEEE Robotics and Automation Magazine 2008;15:58–66.
[3] Chen H, Fuhlbrigge T, Li X. Automated industrial robot path planning for spray painting process: a review. In: Proceedings of the IEEE international conference automation science and engineering CASE, 2008, pp. 522–527.
[4] Corke PI. A Robotics Toolbox for Matlab. IEEE Robotics and Automation Magazine 1996;3:24–32.
[5] Corke PI. Machine vision toolbox. IEEE Robotics and Automation Magazine 2005;12:16–25.
[7] Dubins LE. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. American Journal of Mathematics 1957;79:497–516.
[8] Garga AK, Bose NK. A neural network approach to the construction of Delaunay tessellation of points in Rd. In: IEE transactions on circuits and systems I: fundamental theory and applications, Vol. 41, 1994, pp. 611–613.
[9] Gargantini I. Linear octtrees for fast processing of three-dimensional objects. Computer Graphics and Image Processing 1982;20(4):363–74.
[10] Hall DL, Llinas J. An introduction to multisensor data fusion. Proceedings of the IEEE, 1997; 85: 6–23.
[11] S.D. IFR, World Robotics Report, 〈http://www.worldrobotics.org〉, 2005.
[12] Kohrt C, Pipe A, Schiedermeier G, Stamp R, Kiely J. A robot manipulator communications and control framework. In: Proceedingsof the IEEE international. conference mechatronics and automation ICMA, 2008, pp. 846–851.
[13] Kohrt C, Schiedermeier G, Pipe AG, Kiely J, Stamp R. Nonholonomic motion planning by means of particles. In: Proceedings of the IEEE internetional mechatronics and automation conference, 2006, pp. 729–733.

[14] Likhachev M, Ferguson D, Gordon G, Stentz A, Thrun S. Anytime dynamic A*: An anytime, replanning algorithm. In: Proceedings of the international conference on automated planning and scheduling (ICAPS), 2005.
[15] Masehian E, Amin-Naseri MR. A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning. In: Journal of Robotic Systems, Vol. 21 (6), 2004, pp. 275–300.
[16] Mitsubishi-Electric, 〈http://www.mitsubishi-automation.com〉, 2008.
[17] Qi L, Yin X, Wang H, Tao L. Virtual engineering: challenges and solutions for intuitive offline programming for industrial robot. In: Proceedings of the IEEE Conference robotics, automation and mechatronics, 2008, pp. 12–17.
[18] Ritter H, Martinetz T, Schulten K. Neuronale Netze 1994:3486243446.
[19] Russell S-J, Norvig P. Artificial Intelligence: a modern approach (2nd Edition), 0-13-080302-2, 2002.
[20] TheMathworks, 〈http://www.mathworks.de〉, 2006.
[21] Vleugels JM, Kok JN, Overmars MH. Motion planning using a Colored Kohonen Network, in: technical report RUU-CS, Issue 93-38, 1993.
[24] Siemens, RobCad Simulation Software, in, 〈http://www.plm.automation.siemens.com〉, 2011.
[25] Pan Z, Polden J, et al., 2010. Recent progress on programming methods for industrial robots, in: 41st international symposium on robotics and sixth German conference on robotics, 2010, pp. 1–8.

KOHRT, C., PIPE, A., SCHIEDERMEIER, G., STAMP, R. and KIELY, J. 2008. A robot manipulator communications and control framework. Proc. IEEE Int. Conf. Mechatronics and Automation ICMA.

Proceedings of 2008 IEEE International Conference on
Mechatronics and Automation

FA1-1

A Robot Manipulator Communications and Control Framework

Christian Kohrt, Anthony Pipe, Gudrun Schiedermeier, Richard Stamp, Janice Kiely

Abstract— **The use of industrial scale experimental machinery robot systems such as the Mitsubishi RV-2AJ manipulator in research to experimentally prove new theories is a great opportunity. The robot manipulator communications and control framework written in Java simplifies the use of Mitsubishi robot manipulators and provides communication between a personal computer and the robot. Connecting a personal computer leads to different communication modes each with specific properties, explained in detail. Integration of the framework for scientific use is shown in conjunction with a graphical user-interface and within Simulink as a Simulink block. An example application for assisted robot program generation is described.**

Index Terms—**Manipulator, communication, robot programming, manipulator motion-planning**

I. Introduction

PATH-PLANNING for industrial robots in complex environments where collision avoidance, in cooperation with the presence of a human worker in the robot work space is a research area which merits significant attention. The levels of automation within the automotive industry are expected to increase in future, so as to enhance the economics of manufacture. It is to be expected that in a future factory, human employees will co-exist alongside active industrial robots, to perform such tasks as body-part assembly and sealant application. For example, in car industry a moving conveyor is often used and separation of human employees and robot systems can hardly be realized. An increase in productivity can only be gained with either shorter production cycles or longer production times. Manufacturing industry must have a flexible production to offer highly diversified product mixes in a short delivery time, based on just-in-time small batched production [13]. With changing products production robots must also be programmed more often while the overall production time must be maximized to guarantee a high productivity.

The proposed framework is used in an ongoing project leading to foundations and algorithms for the industrial path-planning task which is the creation of a robot program within static industrial surroundings. The programming task will change from explicit to implicit programming.

A system overview in the next section gives a brief summary of physical devices involved. Section III describes the assisted robot program generation application, which makes use of this framework. Subsection IV.A gives detailed information of the communication modes possible with a Mitsubishi robot system connected to a CR 1 controller. An extension to the built-in communication modes is the data link control mode explained in subsection B. An overview of communication modes and their use is given in subsection C. The next subsection shows how the framework interacts with Matlab/Simulink followed by the *Visualization* component with collision detection and a visual servo control example. A conclusion is given in section V, showing the main use of the presented framework and important results.

II. System Overview

The equipment is shown in Fig. 1. External devices are the pointing device, vision system, robot controller, Teachpendant, robot and personal computer.

Fig. 1. System overview.

The robot manipulator communications and control framework is executed on the personal computer, which has an Ethernet and serial port connection to the robot controller. The Teachpendant and the robot are connected to the controller. The vision system and the pointing device are plugged in to the personal computer. The framework is

Manuscript received May 31, 2008. This work was supported in part by the Bavarian Research Foundation.

Christian Kohrt is with Berata GmbH, Munich, Germany (phone: +49-179-2921307; e-mail: christian.kohrt@berata.com).

Anthony G. Pipe is with UWE - University of the West of England, Bristol, UK (e-mail: anthony.pipe@uwe.ac.uk).

Gudrun Schiedermeier is with UASL – University of Applied Sciences Landshut, Germany (e-mail: gschied@fh-landshut.de).

Richard Stamp is with UWE - University of the West of England, Bristol, UK (e-mail: richard.stamp@uwe.ac.uk).

Janice Kiely is with UWE - University of the West of England, Bristol, UK (e-mail: janice.kiely@uwe.ac.uk).

978-1-4244-2632-4/08/$25.00 ©2008 IEEE

verified with a visual servo control application including collision detection and Matlab/Simulink integration.

III. Implementation of the Framework

The operation of an industrial robot is generally restricted to a small set of commands. Research was undertaken to integrate those commands that control movement with data from the path planning system.

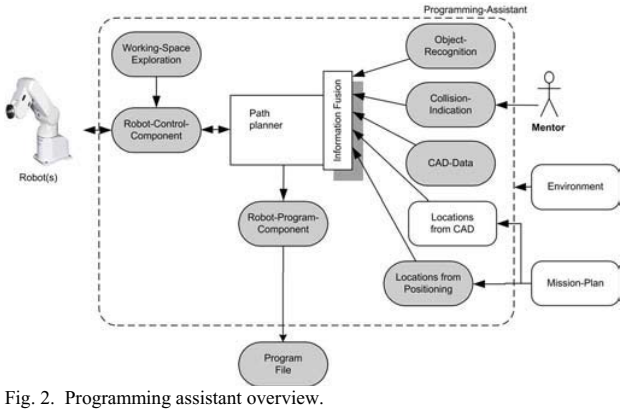


Fig. 2. Programming assistant overview.

An overview of this approach is given in Fig. 2. A programming assistant is proposed, which aims at creating a robot program for industrial robots.

Path-planning in robotics considers model-based and sensor-based information to capture the environment of the robot. Perception, initiated by sensors, provides the system with information about the environment and also interprets them. Those sensors are, among others, cameras or tactile sensors often used for robot manipulators.

The *Object Recognition* component converts the features of an image into a model of known objects. This process consists of segmentation of the scene into distinct objects, determining the orientation and pose of each object relative to the camera and determining the shape of each object. Those features are given with motion, binocular stereopsis, texture, shading and contour. Peter Corke's Machine Vision Toolbox for Matlab [10, 11] allows the user to easily use professional image processing capabilities of Matlab. In addition to the detection of the real environment, model-based data such as CAD-data is used to render the world model more accurate. CAD derived data from simulation systems for example RobCAD are exported as DXF files with all physical locations attached and stored within the world model. Attached locations of CAD-objects are employed so as to acquire information concerning the start and target locations. The path information, such as gluing, painting etc. must be given within the programming assistant. Target locations can also be defined by manual movement of the robot and visual servo control. The latter uses a pointing device to show the robot system the target location. The robot automatically moves to the shown location and stores the position.

The *Working Space Exploration* component gathers information of the environment by moving the robot by manual movement, by random movement with collision indication and by running a robot program, also with collision indication. During motion, the robot ascertains which parts of the space are free in the robot coordinate system either with visual feedback or manual collision indication. This knowledge will become more accurate during the planning process.

The kinematic of a robot is often not precisely known. A neural network based approach is employed to ascertain the dynamic model of a robot by the adoption of a learning process. Visual servo control and working space exploration is used to autonomously learn the robot kinematic.

The *Robot Control* component communicates with the robot and provides direct robot control, serial/Ethernet connection, parameter edit/read/write, program upload and download, real-time movement control, robot system backup/restore, external control over UDP[1] and equipment control.

The robot program is implicitly stored within the robot programming assistant as a trajectory. A transfer to a robot specific program is done in the *Robot Program* component in two steps: First, translation to a pseudo robot program with no other information than the provided trajectory, and, second, generation of the specific robot program with additional configuration commands and specific syntax of the robot programming language. The two step transfer can also be adopted to support other robot types and manufacturers.

The employment of industrial robots without modification of the robot and its controller is necessary for a rollout in industry. The Mitsubishi industrial scale experimental machinery robot system used is the RV-2AJ robot connected to a CR1 controller. It is well documented, industrially proven and communication with a personal computer is possible. Its commercial viability has already been demonstrated in the manufacture of car sub-assemblies, semiconductor memories and other industrial/consumer goods [12]. The main areas of application are assembly, manufacture, pick & place and handling.

The robot control framework is used to verify a path planning algorithm developed at the University of Applied Sciences Landshut in cooperation with the University of the West of England.

The Mitsubishi CR1 controller employed is equipped with a built-in RS232 communication port and an external Ethernet extension box. Both ports are used for communication in the framework.

The tool Matlab/Simulink from TheMathworks [17] is often used in the area of robot control. To use the robot control framework in Simulink it must be encapsulated in Simulink blocks. This brings the benefit to have all of the control within the model and opens the use of Matlab/Simulink for robot control applications in a model driven architecture design.

[1] User Datagram Protocol

IV. THE FRAMEWORK

The aim of the robot control framework is to simplify the usage of robot control and to cover all of the needs originated from robot control applications. It consists of the components described in the sections robot communication, data link control mode, overview of communication modes and their use, connecting the framework to Matlab/Simulink and visualization. An example is given in section F.

A. Robot communication

The available communication modes are controller link mode (CL), data link mode (DL) and real-time external control mode (RTEC).

Controller communication mode (CL)

The controller communication mode is used to set parameters, send robot control commands and read the robot status. Getting the status information during movement of the robot and controlling the robot in real time is not possible in this mode. The data sent over Ethernet is not encoded and can be read in plain text. Thus, it is possible to listen to the Ethernet communication between the controller and the personal computer. Generally, the protocol format for sending commands is the following:

[<Robot No.>];[<Slot No.>];<Command> <Argument>

Each command is followed by a message sent by the controller with status information and the result. Table I states the pattern of such returning messages, where each star stands for a digit:

TABLE I
STATUS OF SENT COMMANDS

Commands	Contents
QoK****	Normal status
Qok****	Error status
QeR****	Illegal data with error number
Qer****	Error status and illegal data with error number

Real-time external control mode (RTEC)

Real-time external control of the robot is useful for direct robot control, where the trajectory is calculated manually. The real-time external control mode is based on the UDP networking protocol. A UDP datagram is a simple and very low-level networking interface, which sends an array of bytes over the network. Even though they are not reliable their low overhead protocol makes datagram transmission very fast. Since the connection is a single point to point connection between personal computer and controller, UDP can be handled easier. Sending and receiving of packets is monitored and a timeout exception is thrown if the communication does not meet the requirement in time.

Mitsubishi provides a simple C communication program example. Running time is crucial, since every communication cycle has a time period of 7.1ms, which is dependent of the robot hardware. A plain Java port is not capable to communicate with the robot controller in time and leads to a loss of UDP packages. Movement of the robot was not continuous any more.

A dynamic link library for RTEC mode created in C is connected with JNI to Java. The library could also be used in Simulink to build a “hardware-in-the-loop” low level robot control application. This gives full control of the robot and code generation from Matlab/Simulink is possible.

Data link mode (DL)

The data link mode connects a controller with a personal computer or vice versa. Usually, it is used to send robot status information from internal robot sensors or other data to its receiver.

B. Data link control mode

The data link mode is extended by a control component, which gives the opportunity to control the robot while getting status information. The personal computer and the robot controller are arranged in a cascaded control system, where the robot controller calculates the trajectory given by the personal computer in form of movement commands. Those commands can be sent at any time over Ethernet or the serial port and the robot manipulator follows the trajectory without stopping between the buffered movement commands.

Multitasking

Multitasking is used to run the data link control programs in parallel. Multitasking is executed by placing the parallel running programs in slots. Data is passed between programs being executed in multitask operation via program external variables and user defined external variables.

Multitasking configuration

The main control program MULTITASK is executed first in slot 1. It sets the variable M_01 and M_02 to zero and starts the programs DATALINK and CONTROLLINK in slot 2 and slot 3. In line 80 and 90 the program waits for the variables M_01 and M_02 to be set from the other programs to stop execution. The main program multitask.mb4:

```
10 RELM
20 M_01=0
25 XLOAD 2,"DATALINK"
30 XRUN 2,"DATALINK"
40 WAIT M_RUN(2)=1
50 M_02=0
55 XLOAD 3,"CONTROLLINK"
60 XRUN 3,"CONTROLLINK"
70 WAIT M_RUN(3)=1
80 WAIT M_01=1
90 WAIT M_02=1
100 XSTP 2
110 WAIT M_WAI(2)=1
120 XSTP 3
130 WAIT M_WAI(3)=1
140 GETM 1
180 HLT
190 END
```

The DATALINK program in slot 3 shown below sends the timestamp, current joint position, current speed of the tool

center point and current position.

Sending is looped over lines 100 to 130 and it sends until a zero value is received. After closing the communication port, the program notifies the MULITASK program in slot 1 that the signal is turned on by means of the external variable M_02. The communication program datalink.mb4:

```
10 WAIT M_02=0
20 M_TIMER(1)=0
30 OPEN "COM2:" AS #2
35 INPUT #2,DATA
40 IF DATA = "0" THEN 160
100 PRINT#2, M_TIMER(1), "|", P_CURR, "|", J_FBC,
"|", J_CURR, "|",M_RSPD(3)
130 GOTO 100
160 M_02=1
170 WAIT M_02=0
180 END
```

The CONTROLLINK program moves the robot manipulator by receiving and executing movement commands. This program runs in a cyclic mode and no user interaction such as moving the robot with the Teachpendant or by robot commands in controller communication mode is possible. For control communication the RS232 port is used, which is a slow connection but fast enough for direct robot control commands. The data link mode is extended by a movement command and leads to the data link control mode. The movement program controllink.mb4:

```
10 WAIT M_01=0
20 OVRD 100
30 GETM 1
40 CNT 1, 300
50 SERVO ON
60 OPEN "COM1:" AS #1
70 DEF JNT JNTPOS
80 INPUT #1, JNTPOS
90 MOV JNTPOS
100 GOTO 80
```

With the CNT command, the robot continuously moves to multiple movement positions without stopping at each movement position.

C. Overview of communication modes and their use

Use cases for robot control are defined in table III. Since it is not possible to send control commands and information requests over one connection, a second connection is always needed to get actual status information during motion.

TABLE II
COMMUNICATION MODES

Mode	Phys. layer	Command type	Feed back type	UC 1	UC 2	UC 3	UC 4	UC 5	UC 6
RTEC	ETH	SDO	SDO	X	-	-	-	-	-
DL	ETH	SD	SD	-	-	-	X	X	X
DL	RS232	SD	SD	-	-	-	X	X	X
CL	ETH	Robot command		-	X	-	-	-	-
CL	RS232	Robot command		-	-	X	-	-	-
CL	ETH	Robot program		-	-	-	X	-	-
CL	RS232	Robot program		-	-	-	-	X	-

(RTEC – Real Time External Control; DL – Data Link; CL – Control Link; ETH – Ethernet; SDO – Serialized Data Object; SD – Serialized Data; UC – Use Case)

An overview of communication modes and use cases of table III is given in table II.

TABLE III
USE CASES

Use-case	Description
1	Direct robot control over Ethernet with feedback. Either the mentor or the path-planning-system can move the robot manually. No controller calculations are involved.
2	Robot operation with singular movement commands over Ethernet. The controller calculates the path. Feedback data can be retrieved by Ethernet connection after finishing movement.
3	Robot operation with singular movement commands over serial port. The controller calculates the path. Feedback data can be retrieved by serial port connection after finishing movement.
4	Robot operation with robot programs over Ethernet. The controller calculates the path. Feedback data can be retrieved either by Ethernet or by serial port connection.
5	Robot operation with robot programs over serial port. The controller calculates the path. Feedback data can be retrieved either by Ethernet or by serial port connection.
6	Robot operation by two data-link channels. One sending channel over serial port and one receiving channel over Ethernet. The robot has to be programmed so that it is possible to send movement-type and data.

D. Connecting the framework to Matlab/Simulink

This section shows the mature steps and important key issues to integrate the Java framework with a SWT[2] user interface to Matlab/Simulink.

Matlab/Simulink integration

The framework must be executed in its own thread to avoid a freeze of the Matlab thread. The implementation as a singleton of the framework GUI[3] guarantees that only one single instance of the GUI is running per Matlab instance.

Communication must be established between Matlab and Java. Calling Java classes from Matlab is supported by default. To communicate back to Matlab/Simulink, two cases of software usage are possible: As a standalone client and as a plug-in. A standalone client is running outside of Matlab/Simulink, whereas a plug-in is started within. This has a great impact on the communication of Matlab and Java. While the standalone client must have interprocess communication, a plug-in does not require this.

Fig. 3. Interprocess communication.

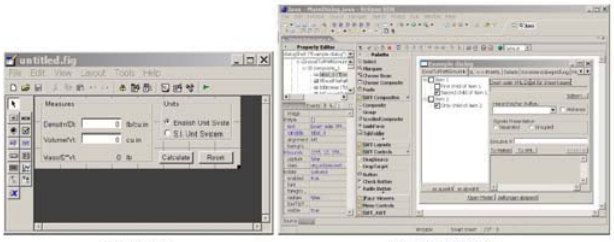
Generally, DLL[4] libraries of Matlab/Simulink can always be called by native system calls. JNI[5] is a wrapper for such

[2] Standard Widget Toolkit
[3] Graphical user interface
[4] Dynamic Link Library
[5] Java Native Interface

system calls and thus can be used. A more convenient possibility is COM[6] or DDE[7] communication. Matlab supports both, the COM and the DDE technology. COM technology is to be used, because the DDE communication server must be switched on in newer versions of Matlab (R14 onwards). In contrast, a plug-in does not need an interface for interprocess communication. In fig. 3 COM/DDE communications is illustrated for standalone clients.

The graphical user interface

Simulink is a platform for multidomain simulation and model based design for dynamic systems. It provides a customizable set of block libraries. Models are built from these blocks that can be connected to solve a given engineering challenge. Usually such systems are quite complex and users not familiar with the model will have difficulties to modify model parameters and to control the model. Therefore, a centralized user input to the model can be realized through a GUI. A GUI development environment (GUIDE) is shipped with Matlab and, thus, becomes the standard tool for GUI creation. A Java application within Matlab/Simulink has greater functionality, i.e. interconnection to a server. It also allows the use of another GUI library such as SWT or Swing for standardized development of complex GUIs.



(a) GUIDE (b) SWT Designer

Fig. 4. The GUI editors GUIDE and SWT.

SWT-based applications integrate seamlessly into the host environment. The library is an adapter to the native widgets. The design of SWT as an adapter makes a small library possible. These libraries must be available on the target computers.

The Simulink Blockset

To allow experiments in model based design methodology the Java robot control framework is integrated into a Simulink blockset. However, code generation is not possible with those blocks. For simulation, additional blocks for forward and backward calculation are needed. A Simulink block usually supports simulation and code generation. Since Java is used, this feature cannot be supported.

The blockset consists of the blocks listed in table IV. It is not possible to use more than one robot control block at the same time because every block needs its own explicit connection.

A stopped robot movement is a movement with stops between two movement commands. It is also a blocking command, which means movement finish must be awaited to send the next command. A non blocking continuous movement is a movement that can do continuous movements also between two commands and the movement command can be sent at any time. The status block always uses the data link communication mode.

TABLE IV
SIMULINK BLOCKS

Block name	Operation mode	Description
Status	data link	continuous measurement
RelJoint	data link control	continuous, non blocking
RelCart	data link control	continuous, non blocking
CircularMov	controller operation	stopped, blocking
LinMovJoint	controller operation	stopped, blocking
LinMovCart	controller operation	stopped, blocking
JointMovJoint	controller operation	stopped, blocking
JointMovCart	controller operation	stopped, blocking

E. Visualization

Visualization is done with a Java3D scenegraph. But not only viewing the robot but also collision testing should be done combined with ODEJava, a physical simulation system.

Collision detection

The built-in Java3D testing does only tests in every frame. Collisions of fast moving objects could take place between two frames that leads to an unrecognized collision. The Open Dynamics Engine (ODE) library written in C and its Java binding ODEJava is used to do collision detection. The ODEJava project allows using ODE with Java. ODE is a free, industrial quality library for simulating articulated rigid body dynamics in virtual reality environments. It has built-in collision detection. The Project also contains tools for binding ODEJava into Xith3D, jME and Openmind scenegraph projects. Since Java3D scenegraph is used, development of a graphics engine is necessary to combine ODEJava with Java3D.

The *Viewing* component also supports visual display of DXF CAD-data as well as any Java3D object. The ODEJava library is used for collision detection of basic geometric objects. DXF-data cannot be used with ODEJava, but can be viewed with Java3D. Collision detection with complex DXF data is therefore rudimentary supported. Collision points and vectors are hard to calculate from DXF-data, but can be done manually. The requirements for DXF-data collision detection are fulfilled. The basic geometric objects are fully supported and have higher requirements in terms of accuracy, because the neural net used in the *Path Planner* component is simulated by those objects [14].

F. Visual servo control

Visual servo control is used for user interaction with the robot system by a pointing device for example used in the

[6] Component Object Model
[7] Dynamic Data Exchange

Working Space Exploration or *Location Positioning* components.

The transformation of picture coordinates of the camera views to robot coordinates by a neural net is learned. The system interprets then the pointing device of the mentor and controls the robot so that it moves in the direction of that point.

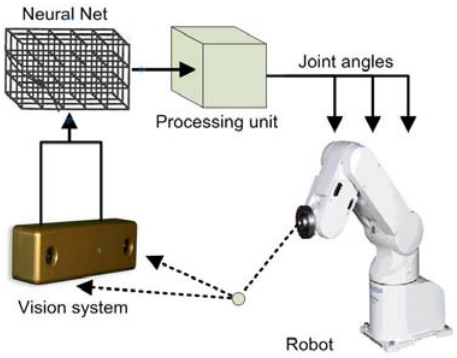


Fig. 5. Example application

An extension to Kohonen's model [9] is implemented to autonomously learn the positioning of a robot arm to a visually given point (Fig. 5). To get information of the position of the objects in space, the robot cell is equipped with two cameras, which monitor the robot cell. During training, the position of the target location within the working-space is randomly chosen. The target location is monitored from the cameras and their signals are applied to the neural net. Every neuron is responsible for a subspace of the robot cell. If a target location is chosen, this neuron becomes activated and provides control signals to the robot controller. The position of a robot arm with five joints is not only a five dimensional vector, but every camera delivers its two dimensional point of the viewing pane. The neural net has to transform that position information to control signals for the five robot joints.

More information about the robot, the cell, the cameras or its positions in space is not needed. Moreover, this must be learned by the neural net.

At the beginning, the robot will move to incorrect robot positions. The difference to the target position is used to train the net. Then the robot will be given the next target position which gives the system the opportunity to learn a second time and so on. The robot is an autonomous learnable system.

V. Conclusion

Robot control applications need a connection to the real robot system. Sending robot control commands as well as receiving information of the robot status and position is necessary especially for path-planning applications, where the focus is on algorithm development. This framework offers the possibility use a standard industry robot system. The framework extends the Mitsubishi CR1 controller family robot system to send robot commands during movement of the robot manipulator without stopping between two commands and to receive robot information during movement. All communication modes over serial port and Ethernet are discussed. Besides the use of the robot control framework as a standalone application, it can also be used with Matlab/Simulink and interconnected within Simulink models to support a wide range of robot control applications.

Acknowledgment

We thank Stefan Holzer for his support and valuable work of an earlier project at the UASL University of Applied Sciences Landshut.

References

[1] Jim Tung, "The impact of model based design on product development," in Model-Based Design Conference, 2005.
[2] *Mitsubishi-Electric Manual*, Connection with personal computer, 2005.
[3] *Mitsubishi-Electric MELFA Industrial Robots Instruction Manual (Functions and Operations) CR1/CR2/CR3/CR4/CR7/CR8 Controller*, Mitsubishi-Electric, 2003.
[4] *Mitsubishi-Electric MELFA Industrial Robots Instruction Manual CRn-500 Expansion Serial Interface*, Mitsubishi-Electric, 2003.
[5] *Mitsubishi-Electric Mitsubishi Industrial Robot CRn-500 Series Personal Computer Support Software Instruction Manual*, Mitsubishi-Electric, 2003.
[6] *Mitsubishi-Electric MELFA Industrial Robots Instruction Manual Controller CR1*, Mitsubishi-Electric, 2002.
[7] *Mitsubishi-Electric Ethernet Interface CRn-500 series Manual*, Mitsubishi-Electric, 2002.
[8] *Mitsubishi-Electric MELFA Industrial Robots Instruction Manual RV-1A/2AJ Series,* Mitsubishi-Electric, 2002.
[9] H. Ritter, T. Martinetz, K. Schulten, "Neuronale Netze," Oldenbourg, 1994.
[10] P. I. Corke, "The Machine Vision Toolbox," in IEEE Robotics and Automation Magazine, 12(4), pp 16-25, November 2005.
[11] *Programming - Matlab Image Processing Toolbox Version 2*, Mathworks, 1997.
[12] http://www.mitsubishi-automation.com, 2008.
[13] University of Karlsruhe, "EURON II Research Roadmap," www.euron.org, 2005.
[14] C. Kohrt, G. Schiedermeier, A. G. Pipe, J. Kiely, R. Stamp, "Nonholonomic Motion Planning by Means of Particles," in IEEE International Conference on Mechatronics and Automation, Luoyang, China, pp 729-732, June 2006.
[15] C. Kohrt, T. Reicher, R. Rojko, "With Model-Based Design to Productive Solutions: Professional GUIs for Simulink by Utilizing the Java SWT Library," in Design & Elektronik, Stuttgart, Germany, May 2006.
[16] P. I. Corke, "Visual Control of Robots: High-Performance Visual Servoing," New York: Wiley, 1996.
[17] http://www.mathworks.com, 2008.

KOHRT, C., SCHIEDERMEIER, G., PIPE, A. G., KIELY, J. and STAMP, R. 2006. *Nonholonomic Motion Planning by Means of Particles.* International Mechatronics and Automation Conference. Luoyang, China: IEEE.

Proceedings of the 2006 IEEE
International Conference on Mechatronics and Automation
June 25 - 28, 2006, Luoyang, China

Nonholonomic Motion Planning by Means of Particles

C. Kohrt[*], G. Schiedermeier[**], A. G. Pipe[***], J. Kiely[***], R. Stamp[***]
[*] Berata GmbH,
80807 Munich, Germany christian.kohrt@berata.com
[**] Department of Computer Sciences, UASL – University of Applied Sciences Landshut,
84036 Landshut, Germany gschied@fh-landshut.de
[***] Faculty of Computing, Engineering & Mathematical Sciences, UWE – University of the West of England,
BS16 1QY Bristol, UK {anthony.pipe, janice.kiely, richard.stamp}@uwe.ac.uk

***Abstract* - In this article a new approach to planning of a nonholonomic motion is presented. A flexible, intelligent planner based on a static map and the topology of the robot's environment has been developed. The approach uses 'particles' to construct automatically a path between two given locations. The generated path is a smooth trajectory, where the length of the path is kept at a minimum and obstacles are avoided. This concept applies to robots meeting the restrictions of a Dubin's car (nonholonomic robot that can only move forward). After the basic concepts of the approach has been described, simulations will be presented.**

Index Terms – Pathplanning, autonomous, nonholonomic, particles, elastic.

I. Introduction

This paper presents a new motion planner for nonholonomic mobile robots. Such robots have dependent degrees of freedom so that the motion is restricted. In this paper, nonholonomic mobile robots refers to car-like robots. The problem is to find a feasible trajectory for the robot, enroute from its start position to its goal position, without collision with static obstacles. Boundary conditions imposed and dynamics of the robot's kinematic model must be satisfied.

In the geometric formulation of this problem, the robot is reduced to a point on the two dimensional surface with the behavior similar to Dubin's car [7]. This car is able to drive forward only and the radius of steering is bounded.
The resulting paths must be smooth (differentiable) and feasible for a car-like robot. The tangent direction is continuous and they respect a minimum turning radius constraint. These paths can be followed by a real vehicle without stopping and thus have a continuous curvature profile in their motion.

Existing path-planning methods can be found in [3]. Roadmap methods calculate a collection of path segments around static obstacles. This path is calculated by connecting the initial and the goal configuration of the robot with a roadmap that can be built in several ways. For example, the Visibility Graph is built by connecting the initial and target configuration with the edges of all obstacles in the given map. The Voronoi diagram leads through the middle of available corridors between obstacles.

Cell Decomposition methods divide the robot's free space into several regions, so called cells. The connectivity graph is built by connecting adjacent cells. A channel leading from initial to goal configuration through the graph can then be computed. A path can be chosen as, for example, leading through the midpoints of the intersections of two successive cells.

Potential field methods divide the free space into a fine regular grid and search this grid for a free path. Different potentials are assigned to the cells of the grid, where 'attractive' potentials are given to cells close to the robot's goal, 'repulsive' potentials are assigned to obstacles. A path is constructed along the most promising direction.

In a nonholonomic planner, the path is created as a set of maneuvers, which take into account the geometric and kinematic constraints of the robot. Different approaches have been developed using a random planner [4] or nonholonomic graphs.

The main contribution of this paper is to form the paths of an already connected roadmap to conform to the robot's constraints. This goal will be captured by means of 'particles'. The calculations will be done locally with no global knowledge.

The remainder of the paper is organized as follows. In Section II, assumptions are formulated for the path-planning problem. In Section III, the new path-planning concept is explained in detail. Section IV proposes a strategy for how to vary the parameters to achieve good results. Examples are given in section V. And finally the paper is concluded with brief remarks in Sections VI and VII.

II. Assumptions

With a predefined roadmap, generated by another path-planner, a Voronoi diagram can be created (see Fig. 1). Thus, the topology of the working space can be obtained. The robot under consideration is shown in Fig. 2. The steering angle is bounded to a maximum absolute angle of e. The car is able to drive around curves with a minimum radius of r. No other maneuvers are allowed and the car can drive forward only.

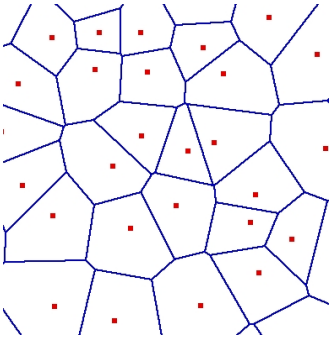


Fig. 1 Voronoi diagram.

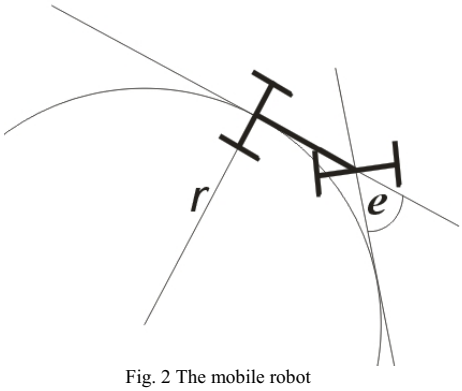


Fig. 2 The mobile robot

The robot operates in a two dimensional environment with static obstacles. Obstacles are represented by polygons. These obstacles are stored in the roadmap. The robot knows its start and goal positions.

III. Theory

A. Correlation between the radius of a curve and the steering angle e

The correlation of e and the radius r is shown for two cases, a regular polygon and a circle. The former will be used later, where e corresponds to g_1, g_2 and g_3 in the ideal case. In Fig. 4, the steering angle e of the real robot from Fig. 2 can be compared.

The formulas for the correlation of e and r are mentioned in (1), (2) and (3).

$$a = \pi - 2 \cdot d = g_n = e \tag{1}$$

$$e = g_n = 2 \cdot \operatorname{atan}\left(\frac{l}{2 \cdot r}\right) \tag{2}$$

$$r = \frac{l}{2 \cdot \tan\left(\frac{e}{2}\right)} \tag{3}$$

B. Installed forces

'Particles', forming a Voronoi diagram of the working space, are utilized to solve a planning problem of a car-like mobile robot. By minimization of newly installed forces at the

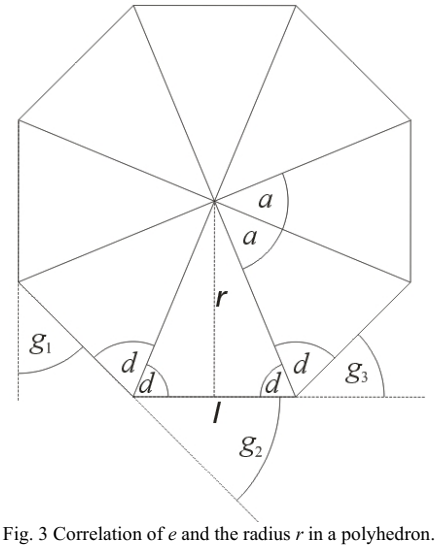


Fig. 3 Correlation of e and the radius r in a polyhedron.

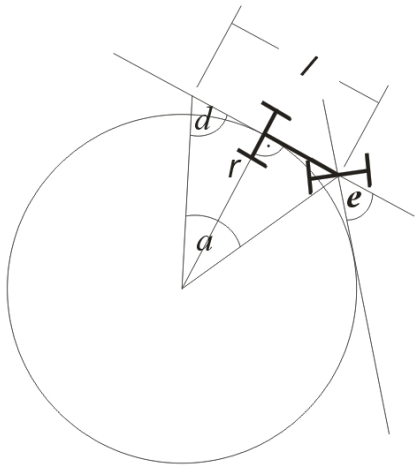


Fig. 4 Correlation of e and the radius r in a circle.

'particles', a path-planning solution can be found.

As can be seen in Fig. 5, three forces are installed. The first force $F_{\text{Equidistance}}$ keeps the distances between the 'particles' equidistant. The second force F_{Rotation}, actually the summary of the four forces $F_{\text{Rotation g}_1}$, $F_{\text{Rotation g}_3}$, $F_{\text{Rotation g}_4}$ and $F_{\text{Rotation g}_6}$, moves the 'particles' on a circle with the neighboring 'particle' as the midpoint. The last one, F_{Shrink}, lets the path shrink in the direction of a straight line.

The direction and value of the forces are influenced by the three neighboring angles g_1, g_2 and g_3 (see Fig. 3 and 5).

C. Equidistance forces

These forces push the 'particles' in a tangential direction. $F_{\text{Equidistance}}$ influences the other forces, especially the rotational forces as little as possible. To reach equidistance of all points the tangential force is utilized. The absolute value of the force is the difference of the distance to the neighboring points (4).

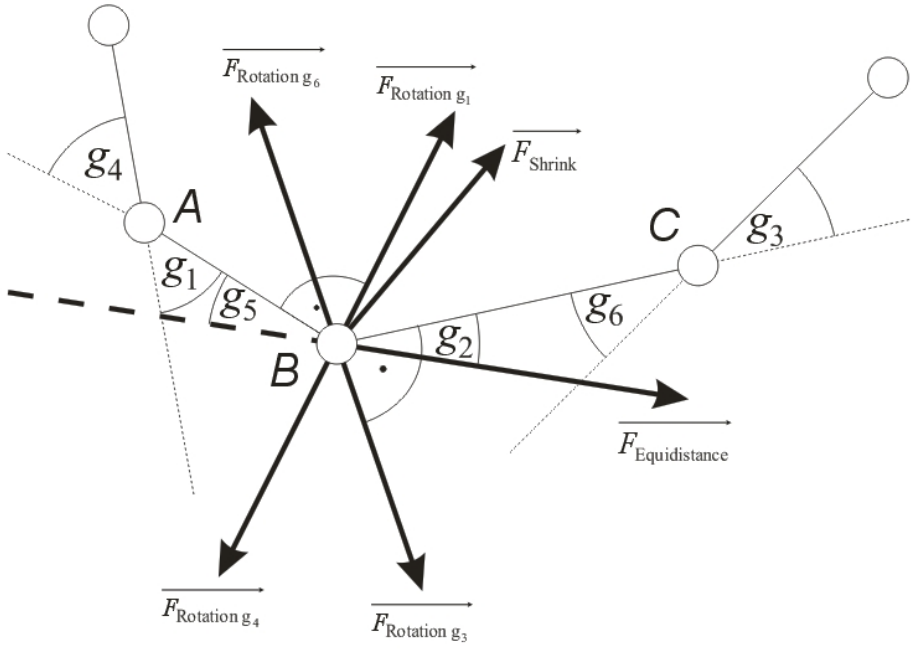


Fig. 5 Installed forces.

$$\overrightarrow{F_{\text{Equidistance}}} = \left(\overline{BC} - \overline{AB}\right) \cdot \frac{\overrightarrow{F_{\text{Tangent}}}}{\left|\overrightarrow{F_{\text{Tangent}}}\right|} \tag{4}$$

D. Rotational forces

The steering angle e (see Fig. 2) can be changed at any time within its boundaries. Curves with a fixed e would result in circular curves. To build a circle of 'particles', it can be seen as a polyhedron, such as shown in Fig. 3. A polyhedron has straight lines between the neighbors and a circle can be approximated by more 'particles'. F_{Rotation} tries to keep the angles of three neighboring 'particles' the same.

Every line tries to minimize the difference of the angles g_1 , g_2 and g_3 with a small rotation (see Fig. 6).

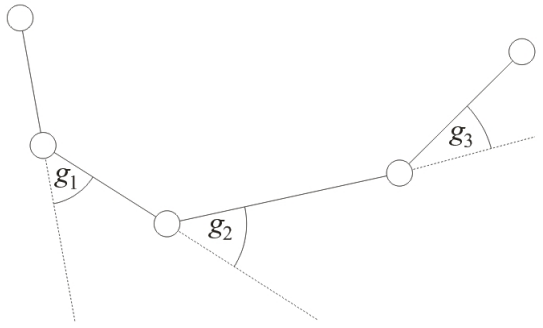


Fig. 6 Angles of the rotational force.

The force of the rotation is orthogonal to its rotation axis. This leads to the formulas for the motion of point B:

$$e_1 = \frac{g_1 + g_2}{2} \tag{5}$$

$$e_2 = \frac{g_2 + g_3}{2} \tag{6}$$

$$\overrightarrow{F_{Rotation\ g_1}} = \left(\frac{\overrightarrow{F_{g_1}}}{\left|\overrightarrow{F_{g_1}}\right|}\right) \cdot \left(e_1 - g_1\right) \tag{7}$$

$$\overrightarrow{F_{Rotation\ g_3}} = \left(\frac{\overrightarrow{F_{g_3}}}{\left|\overrightarrow{F_{g_3}}\right|}\right) \cdot \left(e_1 - g_2\right) \tag{8}$$

$$\overrightarrow{F_{Rotation\ g_4}} = \left(\frac{\overrightarrow{F_{g_4}}}{\left|\overrightarrow{F_{g_4}}\right|}\right) \cdot \left(e_2 - g_2\right) \tag{9}$$

$$\overrightarrow{F_{Rotation\ g_6}} = \left(\frac{\overrightarrow{F_{g_6}}}{\left|\overrightarrow{F_{g_6}}\right|}\right) \cdot \left(e_2 - g_3\right) \tag{10}$$

$$\overrightarrow{F_{Rotation}} = \overrightarrow{F_{Rotation\ g_1}} + \overrightarrow{F_{Rotation\ g_3}} + \overrightarrow{F_{Rotation\ g_4}} + \overrightarrow{F_{Rotation\ g_6}} \tag{11}$$

E. Shrink forces

F_{Shrink} is a constructed force at each 'particle' to build a straight line. This can be done by a simple vector addition of

the two position vectors of the neighbors of each 'particle' (see Fig. 5) while keeping the equidistance constraint.

$$\overrightarrow{F_{\text{Shrink}}} = \overrightarrow{BC} - \overrightarrow{AB} \tag{12}$$

F. Forming lines

Every 'particle's' position lies on an edge of the polyhedron. The overall force leads to a curved connection, where all 'particles' are ordered equidistant and the steering angle e always lies within its boundaries. The path has no straight line, yet.

If the steering angle e is very small, the radius of the curve is very large and can be seen as a straight line. The algorithm considers this with a switch for the calculation of the positions of each 'particle': Shrink forces can be used to form a line. It is a simple vector addition of the two neighboring lines of B to A and C (see (12) and Fig. 5).

A radius threshold r_{max} is introduced, which controls when the formulas for a line or a curve are used. r_{max} is the value for the maximum radius. The angle threshold $t_{a,min}$ can be obtained from (2). If $|g_n| \le t_{a,\min}$ for n=1,2,3 then the 'particles' will be shrunk to a line. Otherwise the rotational forces take place.

To enhance the algorithm, a second constraint leads to faster convergence: If the absolute value of the angles g_n exceeds $t_{a,max}$, calculated with (2) from the threshold r_{min}, the shrink algorithm can be used.

G. Overall force

With both types of motions, it is possible to construct a path from a start position to a goal position with straight lines and curves automatically. The only parameter which is responsible for the decision of whether a line or a curve has to be build is the threshold $t_{a,min}$. The overall formula is now:

$$\vec{F} = C_1 \cdot \overrightarrow{F_{\text{Equidistance}}} + \begin{cases} C_2 \cdot \overrightarrow{F_{\text{Rotation}}}, \text{if } |g_n| > t_{a,\min} \\ C_3 \cdot \overrightarrow{F_{\text{Shrink}}}, \text{if } |g_n| \le t_{a,\min} \end{cases} \tag{13}$$

C_n are parameters to normalize each of the forces and can be utilized to weight each force dependent to the stage of planning. This is utilized in the strategy in section IV.

IV. Strategy

The best strategy is first to start with a high shrink force F_{Shrink} and set the rotational force to zero while using (14). No threshold is applied, yet.

$$\vec{F} = C_1 \cdot \overrightarrow{F_{\text{Equidistance}}} + C_3 \cdot \overrightarrow{F_{\text{Shrink}}} \tag{14}$$

The connections will be as short as possible and the 'particles' are aligned on a line. The motion constraints are not fulfilled at this stage.

On the second stage, F_{Shrink} is getting smaller and the other forces start to grow up to a defined value. Now, the threshold $t_{a,min}$ is applied and (13) is used, which causes the 'particles' to form a canonical path of linear and circular motions.

V. Examples

The topology of the map is obtained by another algorithm, such as a Voronoi diagram. An A* algorithm can be used to find a suitable path. Often, the shortest path is chosen. In these examples a path has been found within the topology map, which has to be optimized from a random state of the 'particles'.

In Fig. 7 $t_{a,min}$ is set to zero and therefore the minimal steering angle e is zero. The path is a smooth curve and there is no straight line. In contrast to Fig. 7, the parameter $t_{a,min}$ in Fig. 8 is set to a value greater than zero. Thus, the path tends to have more straight lines and narrow curves.

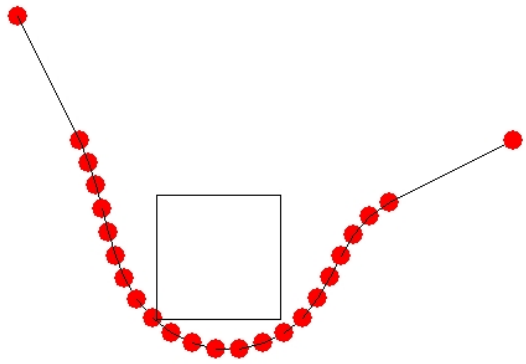


Fig. 7 Path with $t_{a,min} = 0$.

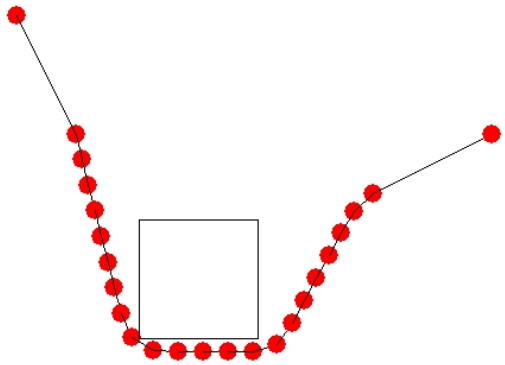


Fig. 8 Path with $t_{a,min} > 0$.

VI. Conclusion

In this paper a new motion planning algorithm is presented for nonholonomic mobile robots in two dimensional configuration space. Planned trajectories are smooth and feasible for car-like robots with a continuous curvature profile. The algorithm works with only local knowledge and can be extended for higher degrees of freedom.

VII. Future Work

The 'particles' have the tendency to build up if the number of 'particles' is high enough. Also, a large number of 'particles' converges very slowly. One of the proposed ideas are additional forces on the equidistance force of straight lines

to lengthen the distance between two ‘particles’. An additional effect could be used if the distance is a function of the velocity of the robot. Another idea is the injection of new ‘particles’ so as to start with a low number of ‘particles’ and increase this number as required by the algorithm.

REFERENCES

[1] H. Jaouni, M. Khatib, J. P. Laumond, “Elastic Bands For Nonholonomic Car-Like Robots: Algorithms and Combinatorial Issues,” 3rd International Workshop on the Algorithmic Foundations of Robotics, Houston, 1998.

[2] Jiang, Kaichun, L. D. Seneviratne, PP. W. E. Earles: “A shortest Path Based Path Planning Algorithm for Nonholonomic Mobile Robots,” Journal of Intelligent and Robotic Systems, 24th Ed, pp. 347-366, 1999.

[3] J.-C. Latombe, “Robot Motion Planning”, Kluwer Academic Publishers, UK, 1996.

[4] S. M. LaValle, J. J. Kuffner, ”Rapidly Exploring Random Trees: Progress and Prospects,” Proceedings of the Workshop on the Algorithmic Foundation, 2000.

[5] S. Quinlan, O. Khatib, “Elastic bands: connecting path planning and control,”, vol. 2, pp. 802-807, 1993.

[6] B. Graf, J. M. Hostalet Wandosell, C. Schaeffer, “Flexible Path Planning for Nonholonomic Mobile Robots,” Fourth European Workshop on Advanced Mobile Robots, EUROBOT '01, Lund, Sweden, September 2001.

[7] L. E. Dubins, “On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents,” American Journal of Mathermatics, vol. 79, pp. 497-516, 1957.

[8] A. Scheuer, T. Fraichard, “Collision-free and continuous-curvature path planning for car-likerobots,”, vol. 1, pp. 867-873, 1997.

KOHRT, C., ROJKO, R., REICHER, T. and SCHIEDERMEIER, G. 2006. *With Model Based Design To Productive Solutions Professional GUIs For Simulink By Utilizing The Java SWT Library*. In: FACHZEITSCHR.-VERLAG, W. (ed.) KFZ-Elektronik.

With Model-Based Design to Productive Solutions: Professional GUIs for Simulink by Utilizing the Java SWT Library

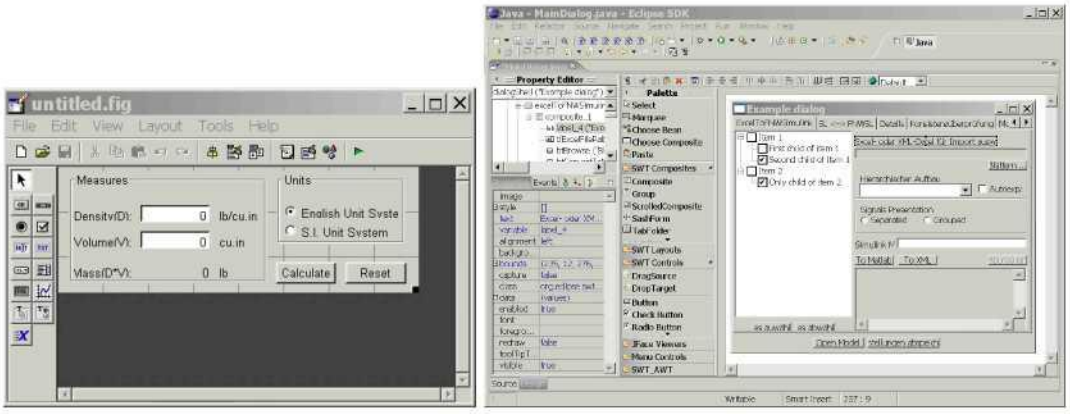
Authors
Dipl.-Ing(FH) Christian Kohrt[1], Dr. Thomas Reicher, Dr. Roman Rojko
Christian.Kohrt/Thomas.Reicher/Roman.Rojko@berata.com
Berata GmbH
Frankfurter Ring 127
80807 Munich, Germany
Tel: +49-89143259-0
Fax: +49-89143259-59

Abstract
The Model-Based Design (MBD) approach is a widely used method to solve scientific engineering challenges [1]. Matlab/Simulink as a representative of MBD is a tool capable of exploiting the advantageous aspects of a graphical user interface (GUI). The latter is created with a tool named GUIDE, which is shipped with the Matlab/Simulink software. Unfortunately, user interfaces created with GUIDE have some drawbacks. Thus, new approaches are needed to overcome these drawbacks to improve the design of the GUI. It is surprising, that the Java SWT library (Standard Widget Toolkit) is not used for such user interfaces. Although not supported by Mathworks, this article compares the features of an SWT based GUI to the GUIDE, explains the practical implementation of SWT GUIs by examples and gives an outlook to the wide field of applications taking benefit.

1 Introduction

The V-Modell [2] is a standard IT product development method publicly available and is used by many companies. It manages required tasks and outcomes, defines methods and functional tool requirements and guarantees a uniform procedure for software development. The functional tool requirements define the functional properties of the tools for software development. These tools are usually organized in a toolchain, that is a set of tools linked together. On each stage in the toolchain, software tools are used, which are designed specifically for the given task.
For example, Matlab/Simulink as a representative for the Model-Based Design approach is used frequently to solve modern engineering challenges in the field of Technical Computing, Control Design, Signal Processing and Communications, Image Processing, Test and Measurement, Analysis, etc. [3].
The requirements of the tools used in such toolchains are of course different from those used as standalone software. While in standalone software input of information are explicitely given to solve a task (and therefore are mostly related to the task), tools in a toolchain must also be able to manage not task related information. They also have to provide suitable interfaces to allow communication to other tools.
Simulink is a platform for multidomain simulation and Model-Based Design for dynamic systems. It provides a customizable set of block libraries. Models are built of these blocks that can be connected to solve a given engineering challenge.
Usually such systems are quite complex and users not familiar with the model will have difficulties to modify model parameters and to control the model. Therefore, a centralized user input to the model can be realized through a GUI. A GUI development environment (GUIDE) is shipped with Matlab and, thus, becomes the standard tool for GUI creation.

[1]also lecturer



(a) GUIDE (b) SWT Designer

Figure 1: GUIDE and SWT Designer

Although GUIDE covers simple graphical objects for GUI design, customer requirements go far beyond this. GUIDE also does not support a seamless transition from Simulink GUIs to those of software developed in another programming language such as Java. Further, the use of the same technology would result in GUIs that are easily exchangeable. Unfortunately, third party tools are not available on market.

The aim of our customer project was to have GUIs with the same 'Look & Feel' in both Simulink and Java developed software and to integrate Simulink in a toolchain. A further aim was to have the right tools to design complex GUIs, i.e. to control a robot path-planning model. We identified the Standard Widget Toolkit as a possible way to achieve those aims. The Java SWT library uses native library calls to create the GUIs. The appearance is indistinguishable from user interfaces of native applications and the user is already familiar to those graphics.

2 Related Work

Although it is possible to embed ActiveX documents into GUIDE and thus extending its functionality, we decided not to use it. Matlab Version R13 does not support ActiveX by default. It is supported since version R14, but we found that it is not flexible enough to satisfy our expectations in GUI design. Usually, the GUI is of a highly complex structure. Compared to the Model-View-Control (MVC) concept, the Model and the View are not separated, which leads to unstructured code.

3 Theory

The Standard Widget Toolkit is an open source framework for developing graphical user interfaces in Java. Matlab/Simulink supports Java by default. Calling Java classes can be done on the command line as well as in a Matlab script. Some options must be set in order to run Java classes.

First, the path to the Java classes must be specified. This can be done in a static or dynamic way. Through dynamic loading of Java classes, also known as 'Hot Deployment', software development is much improved. In the development process, this results in significant time savings. The user is not forced to restart Matlab after each development iteration. Unfortunately, Hot Deployment has got some restrictions in Matlab version R14SP2 whereas other versions such as R14SP1 and R14SP3 work properly.

Second, the Java classes must be copied to a specified location within the file system, where Matlab's Java Path can be directed to. While in development process, these changes can be deployed to Matlab automatically by ANT ('Another Neat Tool') after each development iteration.

ANT is also useful to control the ej-technologies tool exe4j (Java Exe Maker) or an installer such as the Nullsoft Scriptable Install System (NSIS). The requirements to be installed on other 'clean' machines is a suitable Matlab and Java version installed.
As mentioned in section 1, SWT-based applications integrates seamless into the host environment. The library is an adapter to the native widgets. Swing for example emulates the native user interface and, thus, mimic it. Sometimes, the right skins are not available and differences between the native and Swing based user interfaces are apparent.
The design of SWT as an adapter makes a small library possible in contrast to the Swing library. These libraries must be available on the target Computers. Therefore, the installer has to copy those files to the host computer and thus has greater size.
Another advantage of the SWT design is the improved interaction compared to Swing. Since SWT uses native event processing, the inner structure does not vary from the native system and the behavior is thus comparable to the native system. In addition, SWT is less resource-hungry than Swing.
Because SWT is only an adapter to the native host, a more robust and tolerant system can be expected in regard to heterogeneous hardware and the various accelerator settings of the graphics subsystem.
All in all, compared to other GUI libraries SWT has got the greatest advantages.

4 Practical implementation

This section shows the mature steps and important key issues to integrate SWT interfaces to Matlab/Simulink.

4.1 Key issues

4.1.1 Threads

The SWT interface must be run in an own thread. The reason is quite simple: Matlab hangs in its thread waiting for the GUI to end. Thus, no user input in Matlab/Simulink is possible.
Dependent to the application and customer requirements, the implementation as a Singleton of the GUI guarantees that only one single instance of the GUI is running in one Matlab instance.

4.1.2 Calling Matlab/Simulink

Communication must be established between Matlab and Java. Calling Java classes from Matlab is supported by default. But, more investigations were needed to communicate back to Matlab/Simulink. The results are explained in this section.
Two cases of Software usage are possible: As a Standalone-Client and as a Plug-In. A Standalone-Client is running outside of Matlab/Simulink, whereas a Plug-In is started within. This has great impact on the communication of Matlab and Java. While the Standalone-Client must have interprocess communication, a Plug-In does not require this.

Figure 2: Interprocess-communication

Generally, DLL libraries (Dynamic Link Library) can always be called by native system calls. The Java Native Interface (JNI) is a wrapper for such system calls and thus can be used. A more convenient possibility is COM (Component Object Model) or DDE (Dynamic Data Exchange) communication.
Matlab supports both, the COM and the DDE technology for communication. It is planned in future releases to drop DDE and to support COM, only. The DDE communication server must be switched on in newer versions of Matlab (R14 onwards). Only COM communication will be supported in future, so we decided to use this technology. In contrast, a Plug-In does not need an interface for interprocess communication. In figure 2 on the preceding page, COM/DDE communication is illustrated for Standalone-Clients.

4.2 Example applications

In this section, two examples, namely the integration of Simulink into a toolchain and controlling a robot with a complex user interface, will be introduced to give an idea how to implement the mentioned technologies and methods.

5 Conclusion

The application of SWT interfaces is very widespread. Besides the normal use as interfaces for complex models or applications, it also applies to interfaces of Simulink blocks or offer the opportunity to integrate Simulink into a toolchain.
Thus, Simulink is prepared to be used in a toolchain linked to a set of other tools. Those tools maybe deliver information not used by the model. But nevertheless, these data must be given in the correct format to the following tool in the toolchain to guarantee a seamless transition along the toolchain.
The used type of GUI technology is dependent to the kind of data, complexity of the user interface and other requirements.
Using professional SWT interfaces in Simulink is easily possible, once everything is configured. We found, that the benefit of having professional user interfaces, that satisfy customer requirements outweigh the additional time needed for configuration.

6 Acknowledgment

We thank Phillip Ewert for reviewing this proposal and giving valuable comments.

References

[1] Jim Tung. The impact of model based design on product development. In Model-Based Design Conference, 2005.
[2] Koordinierungs-und Beratungsstelle der Bundesregierung fur Informationstechnik in der Bundesverwaltung KBSt. http://www.v-modell-xt.de. 2006.
[3] TheMathworks. http://www.mathworks.de. 2006.
[4] Berthold Daum. Professional Eclipse 3 for Java Developers. Wrox, 2004.