Open-hardware e-puck Linux extension board for experimental swarm robotics research

Wenguo Liu*, Alan FT Winfield

Bristol Robotics Laboratory, University of the West of England, Bristol, UK, BS16 1QY

Abstract

In this paper we describe the implementation of a Linux extension board for the e-puck educational mobile robot, designed to enhance the computation, memory and networking performance of the robot at very low cost. The extension board is based on a 32-bit ARM9 microprocessor and provides wireless network support. The ARM9 extension board runs in parallel with the dsPIC microprocessor on the e-puck motherboard with communication between the two via an SPI bus. The extension board is designed to handle computationally intensive image processing, wireless communication and high-level intelligent robot control algorithms, while the dsPIC handles low-level sensor interfacing, data processing and motor control. The extension board runs an embedded Linux operating system, along with a Debian-based port of the root file system stored in a Micro SD card. The extended e-puck robot platform requires minimal effort to integrate the well-known open-source robot control framework Player and, when placed within a TCP/IP networked infrastructure, provides a powerful and flexible platform for experimental swarm robotics research.

Keywords: Swarm robotics, Mobile robotics, Embedded Linux

1. Introduction

Research on swarm robotics has gained much attention in recent decades as a novel biologically-inspired approach to the coordination of large groups of relatively simple robots, following simple rules [1, 2, 3]. Generally, in order to carry out real robot experiments in research labs we require a robot which is small, reliable and inexpensive, in order to minimise physical space and maintenance for running a relatively large number (several tens) of robots. Traditionally research labs have designed and built their own robot platforms for swarm robotics research, such as the Linuxbot [4], Alice [5], Jasmine [6] and Swarm-Bot [7]. There are also a number of commercially available mobile robots suitable for swarm robotics research, such as the widely used Khepera II and III from K-Team, Lego Mindstorms from the Lego company and Create from iRobot. However, the open-hardware e-puck educational mobile robot developed at the *École Polytechnique Fédérale de Lausanne* (EPFL) has become very popular within the swarm robotics research community within the last three years [8]. The e-puck combines small size – it is about 7cm in diameter, 6cm tall and 660g weight – with a simple and hence reliable design[1]. Despite its small size the e-puck is rich in sensors and input-output devices, and experience shows the e-puck robots have relatively long MTBF (Mean time between failures) and need little maintenance.

The basic configuration e-puck is equipped with 8 Infra-Red (IR) proximity sensors, 3 microphones, 1 loudspeaker, 1 IR remote control receiver, a ring of 9 red LEDs + 2 green body LEDs, 1 3D accelerometer and 1 CMOS camera. Bluetooth provides the facility for wirelessly uploading programs and general monitoring and debugging. All sensors and motors are processed and controlled with a dsPIC30F6014A microprocessor

*Corresponding author

Email addresses: Wenguo.Liu@brl.ac.uk (Wenguo Liu), Alan.Winfield@uwe.ac.uk (Alan FT Winfield)

[1]The open-hardware design can be found at http://www.e-puck.org

from Microchip^{TM}. Extension sockets provide for connecting additional sensors or actuators. Since the first release of the e-puck project in 2006, various extension boards have been developed at different institutions including: a Fly-Vision Turret, Ground Sensors, Colour LED Communication Turret, Omnidirectional vision, ZigBee Communication Turret [9], Range and Bearing Turret [10] and see-Puck display module [11]. These boards physically connect to the basic e-puck robot through its extension sockets, interfacing via an I2C bus or RS232 serial ports. Typically they have separate microprocessors to manage their own sensors or actuators and function as slave modules to the e-puck's dsPIC.

The e-puck's default equipped MCU (Micro Processor Unit) dsPIC30F6014A has certain limitations. One obvious drawback is its limited on-board memory and computation; the dsPIC30F6014A has only 8KB of RAM, 144KB of flash memory and provides 16 MIPS of peak processing power. This constrains the number of sensors the MCU can process in parallel; in particular for vision capture, although the e-puck's CMOS camera has a resolution of 640x480 pixels, only a small fraction of the image can be grabbed by the dsPIC30F6014A. Furthermore the image processing frame-rate is limited by the processing power of the MCU. Also we often require a robot to store data during an experimental run, such as running status, sensor data etc, for monitoring and post-analysis. Although the Bluetooth radio can partially ease the problem of monitoring and data logging for single robots, for more than one robot its limited data rate and network topology severely constrains the number of robots. At the time of writing none of these problems have been addressed by the existing extension boards mentioned above, since they have been designed primarily for specific projects in order to extend the basic e-puck robot with additional sensing, limited range communication or user interfaces. These limitations could be overcome by a higher-performance processor and communications extension board, thus making the e-puck platform more suitable for large-scale swarm robotics research.

Current embedded systems technology offers a number of low-cost solutions for expanding the capability of the basic e-puck platform. The solution presented in this paper was to develop a custom extension board for the e-puck, as a general platform for swarm robotics research, and to make this available to the research community as open-hardware. In contrast with the e-puck extension boards outlined above, an embedded Linux system will be mounted on this board as the primary operating system for the whole robot. High level robot control and management algorithms will run on this board, while lower level sensor processing and motor control will be executed on the e-puck's dsPIC. The extension board will interface directly with the robot's camera in order to overcome the basic e-puck's severe limitations on image processing. The open-hardware extension board will be equipped with a WiFi module in order to provide fast and scalable wireless networking for large robot swarms, supporting both robot-robot communication and a remote terminal interface for programming and monitoring. Thus, the extended e-puck robot offers not only enhanced processing power, memory and communications, but a new and more powerful control architecture for the robot. For instance, it will provide much more flexibility in how the robot may be programmed natively (in contrast with, for instance, cross-compilation on a PC), with standard Linux tools, high level languages (instead of C/ASM only) and frameworks, including Player/Stage [12]. This paper describes both hardware and software design, and implementation, of the e-puck Linux extension board and its new control architecture. The paper is organised as follows. Section II presents the hardware architecture and the specification of each module. Section III describes the operational system considerations for the extension board. Section IV introduces the software design for inter-processor communication and user interfaces, including Player server support. Section V describes and evaluates the hardware implementation then concludes the paper by describing the infrastructure of an experimental swarm robotics research project making use of the Linux extended e-pucks.

2. Hardware

We make use of the extension sockets provided on the e-puck motherboard to connect our Linux extension board. These sockets provide the extension board with power and provide communication with the robot via multiple buses. Figure 1 shows a block diagram of both the extension board (upper) and the e-puck motherboard (lower). In this design the dsPIC of the e-puck motherboard is intended to act as a slave processor handling low-level motor control and sensor data processing including, for instance, AD sampling

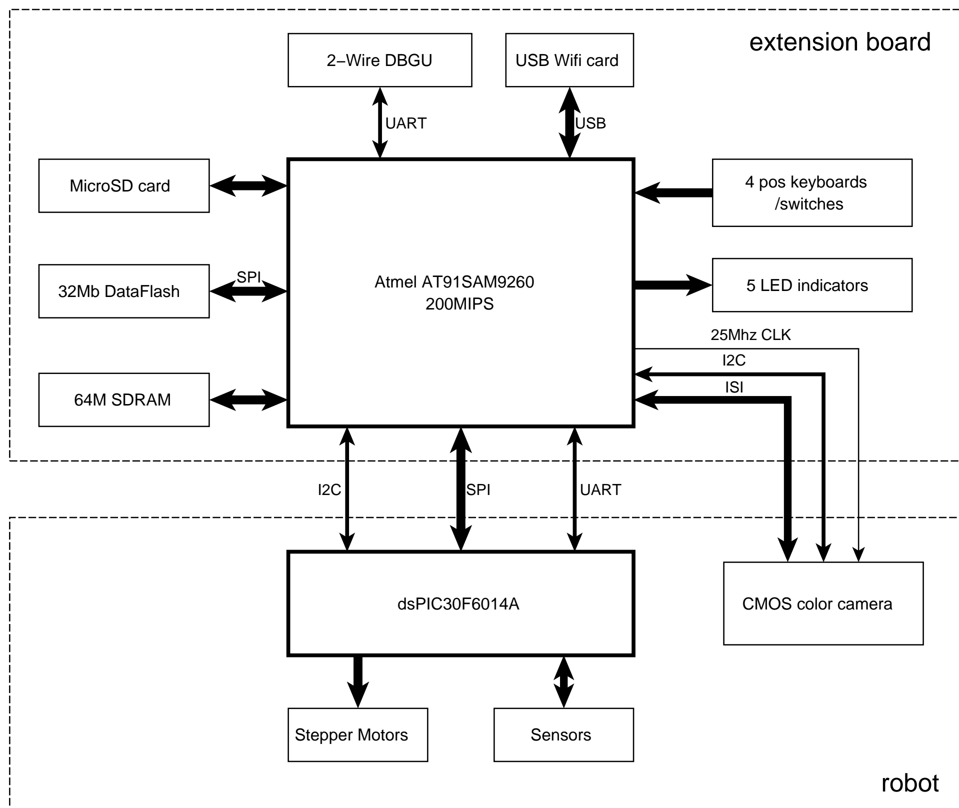

Figure 1: Hardware block diagram showing the Linux extension board (upper) and its interfaces with the e-puck robot motherboard (lower).

and digital filtering. The Atmel MCU on the extension board acts as the master processor, running the robot's operating system (Linux), network communications and high-level intelligent control algorithms. The two processors communicate primarily via the 4-wire SPI bus but, in addition, I2C and UART interfaces are also optionally available for inter-processor communication. To overcome the memory bottleneck of the dsPIC for image capture from the e-puck's camera, the extension board provides a direct connection with the e-puck's CMOS camera via the Image Sensor Interface (ISI); in addition the configuration register of the camera is accessible to the Atmel MCU via another I2C interface. The extension board's USB interface is intended for a USB WiFi card, to provide the robot with wireless networking. The MCU, memory and debugging interfaces are further detailed below.

2.1. Micro Controller Unit

We have selected the AT91SAM9260 32bits MCU from Atmel™(http://www.atmel.com) as the micro controller unit for the extension board. It integrates a 200MIPS ARM926EJ processor with 8KB Data Cache, 8KB Instruction Cache, 32KB Internal ROM and two 4KB Internal SRAMs. The MCU has memory management units and its external bus interface supports SDRAM, Static Memory, ECC-enabled NAND Flash and Compact-flash. The Multimedia Card Interface (MCI) supports SD Memory Card Specification V1.0. It provides a USB 2.0 full-speed device/host interface, two 2-wire UARTs, two SPI and one TWI for inter-processor or general purpose communication. It also has an Image Sensor Interface (ISI) which supports a colour CMOS image sensor, allowing us to connect the extension board directly to the e-puck's CMOS camera. The two-wire UART Debug Unit (DBGU) of the MCU offers a simple interface for programming and debug, i.e. monitoring communications. However, most importantly for meeting our needs, the MCU is Linux compatible and has been well supported by the Linux kernel since version 2.6.19.

2.2. Memory

Two 32MB MT48LC16M16A2 SDRAMs from Micron™technology (http://www.microchip.com) provide a total of 64MB for the Atmel MCU on the extension board. As no internal flash memory is available on the MCU a 4MB dataflash device, the AT45DB321D from Atmel™, is connected to the MCU via an SPI bus for storage of the bootloaders and Linux kernel image. Main solid-state disk storage is provided by a Micro SD card for the Linux file system, development tools, robot applications and data. The Micro SD card is interfaced using the MultiMedia Card Interface (MCI) and fully supported by the Linux kernel. The card can be partitioned and formatted to meet our requirements and, with typical storage capacities of 2GB and greater, it offers an inexpensive and reliable option for the robot's main non-volatile memory.

2.3. Low-level debugging interfaces

The extension board provides a 2-wire UART debug port for uploading code (i.e. bootloaders and the Linux kernel image) to the dataflash memory. No specialist programmer is required as the DBGU can be connected to the standard serial COM port of a desktop computer. The same desktop computer may be used for debugging, using for instance the Linux command-line console. A four-way key switch is connected to the MCU through the GPIO interface; this can be configured as a switch or a simple keyboard for the embedded Linux system. Any status changes of these switches can be captured as interrupt events in order to, for instance, trigger pre-loaded robot control programs without the need for manually logging into the Linux operating system. Another important function of these switches is to trigger safe Linux shutdown in the event that the wireless connection is lost and manual shutdown becomes impossible. Five user-defined LEDs are also available for indicating the status of the Linux system, for example indicating SD card access, activation of the wireless connection, or the running status of the robot controllers.

3. Linux Operating System

We have selected Embedded Debian (EmDebian [13]) as the Linux distribution for our extension board. EmDebian is an official sub-project of Debian and optimised for size. It provides a complete Debian-based environment with minimised Debian packages. Our extension board runs the Armel port of EmDebian [14],

optimised for the ARM architecture. This provides the most common utilities, applications and development environment familiar to anyone used to developing for x86 Linux on a PC.

It is straightforward to set up the software development environment, e.g. editor, compiler and version management tools using the standard Debian command line package management program 'aptitude' and, following appropriate configuration, we can remotely access the on-board Linux operating system via the extension board's WiFi network. The Armel Emdebian port provides a complete solution for programming the ARM9 processor using C, C++, Python and other programming languages. Editing and compiling code can be done on the robot by logging into the system either via SSH through the wireless network or using the serial DBGU port. The use of standard operating system and development environment clearly offers a significant benefit in reduced learning time and a very large toolset for programmers of the extended e-puck.

3.1. Boot sequence

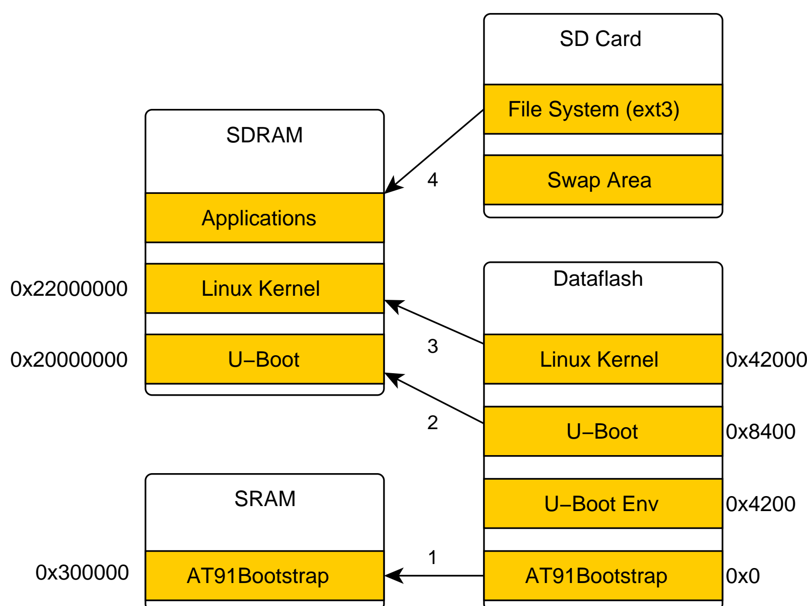

Figure 2: Boot sequence and memory mapping for the Atmel AT91SAM9260 MCU

The Atmel AT91SAM9260 MCU has a boot program located in its internal ROM. This boot program first initialises the DBGU and USB ports for the programmer, then checks if a valid application is present in the FLASH memory (here the DataFlash). If a valid application is found, the code is downloaded into the internal 4KB SRAM and executed. Figure 2 shows the contents of the Dataflash and the steps for loading the Linux system. In order to boot a Linux kernel from Dataflash, two bootloaders must be executed in sequence. The AT91bootstrap [15] is the first level bootloader from Atmel$^{\text{TM}}$. This bootloader is located at address 0x0 and downloaded into the SRAM by the embedded boot program, it performs processor initialisation (PLLs, PIOs, SDRAM, SPI), loads the high level bootloader from DataFlash sectors to SDRAM and then jumps to it. textitDas U-Boot [16] is the second stage bootloader for the extension board, located at address 0x8400. It is loaded to SDRAM by the AT91bootstrap bootloader and responsible for configuring the main interfaces, e.g. FLASH, network, USB keys for downloading and launching the Linux system. This bootloader also provides a command shell and some tools for uploading new kernel binary images to the board's SDRAM via the DBGU port and copying binary images from SDRAM to FLASH memory. There is a small space on the DataFlash (from 0x4200 to 0x8400) which stores U-boot environment variables and boot arguments for Linux kernels. The U-Boot is configured to automatically load and decompress the Linux kernel from the

Dataflash (from address 0x42000) into the SDRAM and then jump to it. The Linux kernel then initialises each of the hardware devices, mounts the root filesystem from the SD card and executes the initialisation process for networking and other system services.

3.2. Root file system

The Linux root file system is stored in the Micro SD card and automatically mounted by the Linux kernel. Although the exact contents of the root file system varies in different Linux distributions, it must include a minimum set of sub-directories: /dev, /proc, /bin, /etc, /lib, /usr, /tmp, plus some basic utilities, configuration files, applications and run-time libraries [17]. For an embedded Linux system the root file system is often created on a desktop computer using cross-compiling tools and then transferred to the target media.

EmDebian presents an easy way of setting up the root file system for our e-puck extension board. The basic root file system can be created for a Micro SD card using the open source processor emulator QEMU [18] on a desktop computer. Further customisation can then be done on the target system using the widely used 'apt-get' program. All packages in EmDebian are synchronised with the main Debian distribution which means it is very easy to keep the system up to date.

3.3. Device drivers

Although the Atmel MCU is well supported in the mainstream of the official Linux kernel distribution since version 2.6.19, the default configuration is customised for the evaluation board of the AT91SAM9260 released by Atmel^{TM}. Several device drivers required by some hardware on the extension board are absent, including drivers for the SPI bus, the keyswitch and the CMOS camera. To add support for these devices we have added relevant code to the kernel sources, enabled the corresponding option flags for the kernel build configurations and then rebuilt the Linux kernel. Some devices are statically built into the Linux kernel, e.g. SD card, SPI, I2C, keyswitch, and status LED, while others are dynamically linked modules which can be loaded at runtime, such as the CMOS camera. All devices on the Linux extension board are accessible from Linux user space for ease of code development. Two types of USB WiFi card, for ZyDAS chipset ZD1211B or Ralink chipset RT73, are natively supported by the extension board, but other USB WiFi cards can be used via the ndiswrapper [19] tools with their windows driver. There is no special requirement for the choice of Micro SD card since most are natively supported by the Linux kernel after enabling its MTD (Memory Technology Device) driver.

4. Software Interfaces

The Atmel MCU on the extension board is intended to act as the robot's master processor; it will be responsible for image processing, wireless communication and high-level intelligent robot control. The dsPIC then works as a slave processor dealing with low-level sensor data processing and motor control. These two processors run in parallel with process synchronisation via inter-processor communication buses. Although multiple interfaces are available on this extended platform, we choose the SPI as the inter-processor communication solution because of its high data rate and full duplex operation. This section outlines the inter-processor communications, the robot programmer's API and, finally, Player server support.

4.1. SPI communication protocols consideration

Given that the Atmel and dsPIC are running in parallel to control the robot, certain synchronisation mechanisms between these two processors are required. We use a message-based approach via the SPI bus to enable the two processors to communicate in master/slave mode. The Atmel MCU initialises communication by pulling the slave selection line low and then issues the clock cycles for data transfer. Each transmission consists of 16-bits words. Multiple transmissions can be issued by the Atmel with the same slave selection line low. As all robot sensor (except for the camera) and motor interfaces are direct to the dsPIC, we need to carefully design the communication protocols to ensure delays in transferring data between processors are within acceptable levels. Since SPI communications is full duplex, data can be sent and received

simultaneously. On each side of the SPI bus two equal-sized buffers are defined, one for transmission and the other for reception. The Atmel and dsPIC each prepare the contents of their transmission buffer before the clock cycles are issued. During data transfer, the messages stored in the transmission buffers are exchanged into the respective receive buffers on each side. We pre-define the structure of these buffers as shown in Listings 1 and 2.

Listing 1: Definition of communication buffers on the extension board side

```
struct txbuf_t
{
    struct cmd_t cmd;          // first two bytes for commands
    int16_t left_motor;        // speed of left motor
    int16_t right_motor;       // speed of right motor
    struct led_cmd_t led_cmd;  // command for leds
    int16_t led_cycle;         // blinking rate of LEDSs
    int16_t reserved[19];      // reserved, to make two buffers the same size
    int16_t dummy;             // leave it empty
};
struct rxbuf_t
{
    int16_t ir[8];             // IR Ranges
    int16_t acc[3];            // Accelerometer x/y/z
    int16_t mic[3];            // microphones 1,2,3
    int16_t amb[8];            // Ambient IR
    int16_t tacl;              // steps made on left motors
    int16_t tacr;              // steps made on right motors
    int16_t batt;              // battery level
};
```

Listing 2: Definition of communication buffers on the e-puck side, note 'int' is 16-bits on this processor

```
struct txbuf_t
{
    int ir[8];                 // IR range values
    int acc[3];                // Accelerometer x/y/z
    int mic[3];                // microphones 1,2,3
    int amb[8];                // Ambient IR
    int tacl;                  // steps made on left motors
    int tacr;                  // steps made on right motors
    int batt;                  // battery
};
struct rxbuf_t
{
    int dummy;                 //leave it empty
    struct cmd_t cmd;          //first two bytes for commands
    int left_motor;            //speed of left motor
    int right_motor;           //speed of right motor
    struct led_cmd_t led_cmd;  //command for LEDs
    int led_cycle;
    int reserved[19];          //reserved, to make two buffers the same size
};
```

Communication via the SPI bus involves two 16-bit shift registers, one master and one slave. As the dsPIC has no DMA facility, updating the shift registers has to be handled by the CPU and can take a few micro seconds. Since there is no hardware flow control or hardware slave acknowledgement mechanism available for the SPI communication, it is possible that the slave may miss the chance to load new data to the shift register when transmission starts. In this case the master will receive the wrong data. To cope with this problem, on the master side we set the clock rate of the SPI to 20MHz and add a delay (2.4 micro seconds) between two consecutive transmissions. On the slave side an interrupt routine is called after each transmission is completed, as shown in Listing 3. Each time one word (16 bits) in the transmission buffer is transferred, the data in the shift register will be saved into the reception buffer and then new data copied to the shift register.

4.2. Robot programmer's API

Under Linux every hardware device is treated as a device file. On the extension board device files are generated dynamically using the 'udev' program. For example, the SPI device interfacing with the e-puck motherboard is displayed as `/dev/spi1.0`, the I2C interface as `/dev/i2c0` and CMOS camera as `/dev/video0`. The device files allow transparent communication between user space applications and computer hardware using standard input/output system calls. Based on these system calls we have developed a

Listing 3: Handling the SPI communication on the dsPIC side

```
void __attribute__((__interrupt__)) _SPI2Interrupt(void)
{
    _SPI2IF = 0;                          // Reset interrupt flag

    index++;
    if (index == BUFFSIZE)
        index = 0;

    SPI2STATbits.SPIROV = 0;              // Clear the rx overflow flag
    ((int*)&rx_buf)[index] = SPI2BUF;     // Read the buffer
    SPI2BUF = ((int*) & tx_buf)[index];   // Write the buffer

}
```

Table 1: API function calls for access of hardware devices

Function	Description
`int initSPI()`	configures the SPI bus interface
`int initCamera()`	configures the CMOS camera interface
`int grabFrame(char *buf)`	grab a frame of image data
`int transfer(rxbuf_t* rx, txbuf_t *tx)`	full duplex data transfer across the SPI bus
`int get_switches()`	get ext. board keyswitch status
`int wait_btn_trigger(unsigned short code)`	wait for the specific keyswitch to be pressed
`void set_speed(int16_t lspeed, int16_t rspeed)`	set speed of left and right robot motors
`void get_ir_data(int16_t *ir, int16_t size)`	get robot IR proximity sensor readings
`int16_t get_battery()`	get battery voltage
`void reset_robot(int fd)`	reset robot, and stop its motors

simple Application Programmer's Interface (API) for initialisation, configuration and access of these hardware devices, and robot programming, as listed in Table 1. Referring to Table 1, basic robot control can be achieved using the functions `get_ir_data()` to read the robot's IR proximity sensors and `set_speed()` to set the robot's left and right motor speeds. The robot's camera can be accessed using the function `grabFrame()`.

4.3. Player server support

The robot server Player is perhaps the most widely used open source robot control interface in robotics research [20]. Running the Player server on the robots provides a network interface for the robot's sensors and actuators. The user client program can then subscribe to these interfaces and control the robot remotely from any computer that has a network connection to the robot. Player defines a set of standard interfaces and allows multiple devices to present the same interface. Such devices don't have to be physically realised; they can be simulated virtual devices. This is extremely useful when Player is combined with the open source simulator Stage [12], in which the standard Player interfaces are presented by the simulated devices. Robot controllers written for Stage generally work on real robots with no, or very few, changes. Thus we can test and debug a controller quickly in simulation before then transferring it to real robots.

In order to port the Player server to the extension board, we have implemented the Player device driver 'lpuck' using the API outlined in the previous section. Listing 4 shows the configuration file for running the Player server on the Linux extended e-puck robot. The interfaces provided by the lpuck driver provide the client program with full access to every sensor and actuator of the robot. Also shown in Listing 4 is the 'cmvision' driver, this is one of the abstract drivers provided by Player; it implements a number of algorithms for finding coloured blobs in the images grabbed using the robot's CMOS camera. The fact that Player includes many such abstract drivers provides further justification for installing Player on the robot.

Listing 4: Configuration file for Player server

```
(
  name "lpuck"
  plugin "liblpuck"
  provides ["position2d:0" "camera:0" "ir:0" "power:0" "aio:0" "blinkenlight:0"]
  load_gps 0
  camera_device "/dev/video0"
  image_size [640 480]
  save_frame 0
)

driver
(
  name "cmvision"
  provides ["blobfinder:0"]
  requires ["camera:0"]
  colorfile "colors.txt"
)
```

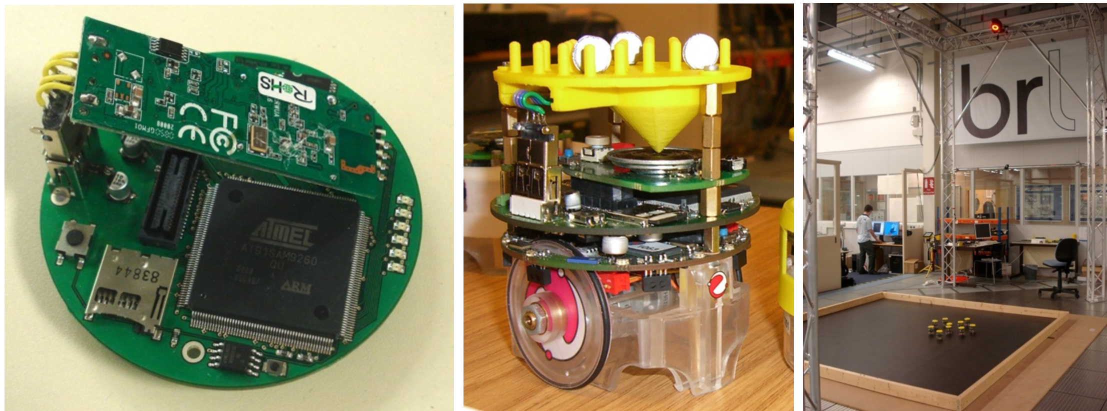

Figure 3: Left: e-puck Linux extension board with USB WiFi card (casing removed). Middle: An e-puck with Linux board fitted in between the e-puck motherboard (lower) and the e-puck speaker board (upper). Also note the yellow 'hat' which here serves three different functions: (1) it provides a matrix of pins for the reflective spheres which allow the tracking system to identify and track each robot; (2) it provides a mounting for the USB WiFi card which slots in horizontally (the wires connecting to the WiFi card are above the USB connector); and (3) it provides an inverted cone to reflect sound from the e-puck's speaker horizontally so that it can be heard by other e-pucks. Right: experimental swarm robotics arena with 10 Linux extended e-pucks.

5. Implementation, Evaluation and Infrastructure

Figure 3:left shows the Linux e-puck extension board. The board is 75mm in diameter, slightly larger than the e-puck motherboard at 70mm, in order to accommodate large footprint packages and manual soldering. A 2GB SanDisk Micro SD card provides external storage. Two partitions are created in the SD card; one is allocated as a swap area (64MB) and the other is formatted ext3 for the Linux root file system. A USB 802.11G WiFi card (with Ralink chipset RT73) provides wireless networking. The extension board is connected to the e-puck motherboard via two connectors on its underside, providing power supply, communication buses and other signals. The extension board has also one socket on its upper side so that the original e-puck speaker board can be re-fitted (after removal of one of its redundant connectors). Figure 3:middle shows the e-puck robot with the extension board installed. The primary function of the yellow 'hat' at the top of the robot is to allow us to mount reflective markers for the vision tracking system, but additionally the USB WiFi card (with its cover removed), is fitted into a slot on the underside of the hat. We have extended in total 50 e-puck robots in this way. The cost for building 50 extension boards (outsourced), including the Micro SD and WiFi cards, was around £4000 in total, i.e. a unit cost of about £80.

5.1. Performance evaluation and limitations

The power consumption of the extension board is about 350mA (of which 190mA is consumed by the USB WiFi card). Although the extension board reduces the battery life (for a comparable motor on/off duty cycle) from typically 3 hours to 1.5 hours, this is still more than sufficient for complex swarm robotics experiments.

The performance of the wireless LAN communications (WiFi) has been tested using the commonly used network testing tool *Iperf* (http://iperf.sourceforge.net/), and demonstrates a data throughput of 5.6Mbps between the Linux-extended e-puck robot and a 802.11b/g Access Point (AP). The same test between a desktop PC, fitted with the same USB WiFi card as the robot, and the Access Point yields a 20Mbps throughput. The poorer wireless network performance of the robot is almost certainly due to the lower speed of its USB interface (12Mbps) and the lower CPU speed (200MHz) of the Atmel MCU. Clearly, the performance of the WiFi network varies with the number of nodes connected to the AP, and a similar test between two robots produces a transfer speed of 4.6Mbps while the speed between two PCs is 6.4Mbps. This implies that the relative effect of low speed USB and lower CPU speed on wireless performance is much smaller for robots communicating via the WiFi network.

We have tested the image acquisition capability within the Linux extension board using the *v4l2* utilities (http://linux.bytesex.org/v4l2/). The 24MHz clock of the camera module is provided by the Atmel MCU chip, and trials show that the system can acquire colour images, with 640 x 480 resolution, at up to 16 frames per second (FPS). The frame rate drops to 10FPS if a blob detection algorithm, for instance CMVision [21], is applied to these images.

Not all the sensory modalities of the extended e-puck robot are fully supported at the time of writing, in particular audio processing. Because of the lack of DMA support for the SPI module in the dsPIC MCU, the current implementation places a limitation on the number of concurrent data streams between the dsPIC and Atmel processors. Although a simple test shows the current SPI inter-processor communication can achieve a 5Mbps data rate, this rate has to be slowed when the dsPIC needs to process all of the robot's sensors at the same time. For future swarm robotics experiments that require audio processing (i.e. for signalling between robots with sound), we plan to replace the dsPIC30F6014A on the standard e-puck board with a pin-compatible dsPIC33F series chip which has DMA support for the SPI module, for example the dsPIC33FJ128GP708. This work will also include exploiting multiple inter-processor communication channels by, for instance, using the SPI interface as the main bus for real-time microphone sampling data and audio output, and the I2C interface for low rate sensor data and motor commands.

5.2. Swarm robotics infrastructure

Programming, initialising, starting and stopping experimental runs of a large swarm of mobile robots, then monitoring and logging data from those runs, is problematical if it has to be done manually. However,

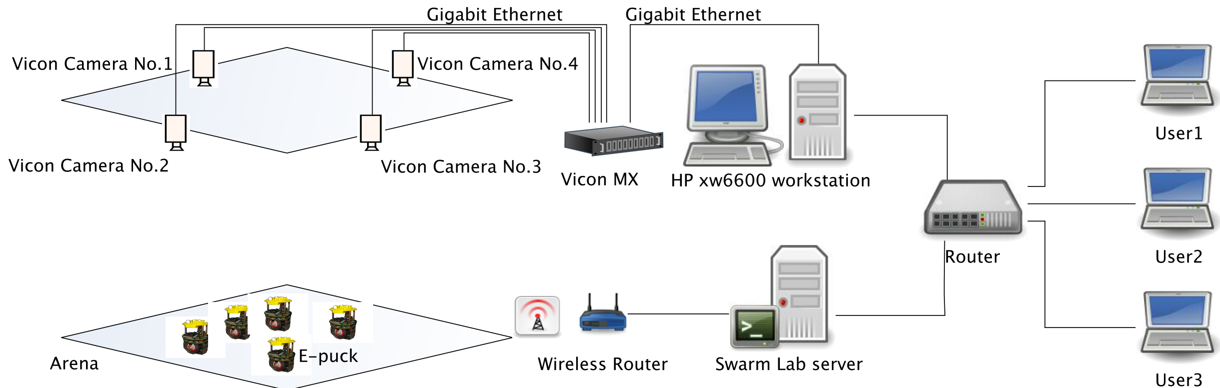

Figure 4: Experimental infrastructure for swarm robotics research based on the Linux extended e-puck. The Swarm Lab Server provides a data logging capability that combines and time stamps position tracking data collected by the Vicon system with robot status and sensor data from the e-pucks via WiFi, into a log file for post-analysis of experimental runs.

with the Linux extended e-pucks and wireless networking, we have been able to set up a powerful infrastructure for programming, controlling and tracking swarm experiments much more conveniently. Figure 4 illustrates the overall structure of the experimental infrastructure. Each e-puck robot is configured and identified with a static IP address. They connect to the LAN through a wireless router and can be accessed from any desktop computer connected to the network using the SSH protocol. A 'swarm lab server' is configured as a central code repository and data logging pool for the swarm of robots. The server also functions as a router to bridge the swarm's wireless subnet and the local network. In addition, as there is no battery-backed real time clock (RTC) on the extension board, the server provides a time server for synchronisation of the robots' clocks and time stamping log data.

A vision tracking system from Vicon^{TM}(http://www.vicon.com) provides high precision position tracking for robot experiments. This consists of four Vicon MX400 cameras, one Vicon MX and one HP xw6600 workstation. Each robot is identified uniquely by the vision tracking system from the pattern of reflective markers mounted on the matrix pins of the yellow hat, as shown in Figure 3:right. The tracking system is connected to the local network and broadcasts the real-time position of each tracked robot through a standard TCP/IP port. We use the position tracking data for two different purposes: firstly, for logging and post-analysis of experimental runs and, secondly, to provide real-time position feed-back to robots. For the latter, in order to reduce the latency the 'swarm lab server' runs a process that decodes the data stream from the Vicon system and broadcasts the processed position information together with the robots' ID tag over the wireless sub-network. In this way each robot can access its position, with minimum delay, by simply matching its own tag with the broadcast data stream. This creates a GPS-like device for the extended e-puck robot which we have also integrated as a virtual device in the Player driver.

In conclusion, the Linux extension board described in this paper provides the e-puck robot not only with improved computation, memory and communications but also a flexible control architecture, that allows us to develop and test more demanding embedded robot controllers and swarm algorithms than is possible with the basic e-puck. One swarm robotics project benefiting from the extended e-pucks, control architecture and infrastructure is investigating social learning in a robot collective, in which robots imitate each others' behaviours [22]. Robot-robot imitation requires embedded image processing so that robots can imitate movements observed through their cameras. We need to capture and record, for later analysis, the complex interactions of many such imitation events and the flow of imitated behaviours across the robot collective, but this would not be possible without the experimental infrastructure presented here and the Linux extended e-puck at its heart.

Acknowledgements

This work has been supported by the UK Engineering and Physical Sciences Research Council (EPSRC), grant reference EP/E062083/1. The authors gratefully acknowledge the contributions of co-workers Paul O'Dowd, Mehmet Erbas and Jean-Charles Antonioli for help in testing the Linux extended e-puck.

Open source materials

All hardware design files including schematics, gerber files and bills of materials are released under a GPL license at http://www.brl.ac.uk/projects/culture/epuck_linux.html. This web page includes also all source code for the device drivers, user API library and programming examples for the Linux extended e-puck robots.

References

[1] M. Dorigo and E. Şahin, Eds., *Autonomous Robots: Special issue on Swarm Robotics.* Springer, 2004, vol. 17, no. 2-3.

[2] E. Şahin and W. Spears, Eds., *Swarm Robotics Workshop: State-of-the-art Survey*, ser. Lecture Notes in Computer Science. New York: Springer, 2005, vol. 3342.

[3] E. Şahin and A. F. T. Winfield, Eds., *Swarm Intelligence: Special Issue on Swarm Robotics.* Springer, 2008, vol. 2, no. 2-4.

[4] A. F. T. Winfield and O. E. Holland, "The application of wireless local area network technology to the control of mobile robots," *Microprocessors and Microsystems*, vol. 23, no. 10, pp. 597–607, 2000.

[5] G. Caprari, T. Estier, and R. Siegwart, "Fascination of Down Scaling — Alice the Sugar Cube Robot," *Journal of Micro-Mechatronics*, vol. 1, no. 3, pp. 177–189, 2002.

[6] S. Kornienko, O. Kornienko, and P. Levi, "Collective AI: context awareness via communication," in *IJCAI'05: Proceedings of the 19th international joint conference on Artificial intelligence.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005, pp. 1464–1470.

[7] R. Gross, M. Bonani, F. Mondada, and M. Dorigo, "Autonomous Self-assembly in Swarm-Bots," *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1115–1130, 2006. [Online]. Available: http://www.swarm-bots.org

[8] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a robot designed for education in engineering," in *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, vol. 1, no. 1, 2009, pp. 59–65.

[9] C. M. Cianci, X. Raemy, J. Pugh, and A. Martinoli, "Communication in a swarm of miniature robots: the e-puck as an educational tool for swarm robotics," in *Simulation of Adaptive Behaviour (SAB-2006), Swarm Robotics Workshop*, ser. Lecture Notes in Computer Science, vol. 4433, 2006, pp. 103–115.

[10] Álvaro Gutiérrez, A. Campo, M. Dorigo, J. Donate, F. Monasterio-Huelin, and L. Magdalena, "Open e-puck range & bearing miniaturised board for local communication in swarm robotics," in *2009 IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009, pp. 3111 – 3116.

[11] M. Jacobsson, Y. Fernaeus and L.E. Holmquist, "GlowBots: Designing and Implementing Engaging Human-Robot Interaction," *Journal of Physical Agents*,vol. 2, no. 2, pp.51–60, 2008

[12] R. Vaughan, "Massively multi-robot simulation in Stage," *Swarm Intelligence*, vol. 2, no. 2–4, pp. 189–208, 2008.

[13] EmDebian. (2010) The embedded Debian project. [Online]. Available: http://www.emdebian.org/ (last accessed date: 15/03/2010)

[14] Debian Wiki. (2010) The ARM Eabi port. [Online]. Available: http://wiki.Debian.org/ArmEabiPort (last accessed date: 15/03/2010)

[15] Atmel. (2006) AT91Bootstrap,. Application Note. [Online]. Available: http://www.atmel.com/dyn/products/tools (last accessed date: 15/03/2010)

[16] DENX Software Engineering. (2010) Das U-boot – the universal boot loader. [Online]. Available: http://www.denx.de/wiki/U-Boot (last accessed date: 15/03/2010)

[17] D.P. Bovet, M. Cassetti , A. Oram, *Understanding the Linux Kernel*, O'Reilly & Associates, Inc., Sebastopol, CA, 2000

[18] QEMU. (2010) QEMU - open source processor emulator. [Online]. Available: http://wiki.qemu.org/Index.html (last accessed date: 15/03/2010)

[19] NDISwrapper Software Community. (2010) The NDISwrapper project. [Online]. Available: http://ndiswrapper.sourceforge.net/ (last accessed date: 15/03/2010)

[20] B. P. Gerkey, R. T. Vaughan, K. Stoy, A. Howard, G. S. Sukhatme, and M. J. Matarić, "Most valuable Player: A robot device server for distributed control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2001, pp. 1226–1231.

[21] J. Bruce, T. Balch and M. Veloso, "Fast and Inexpensive Color Image Segmentation for Interactive Robots", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Japan, 2000, pp. 2061–2066.

[22] A. F. T. Winfield and F. Griffiths, "Towards the emergence of artificial culture in collective robot systems," pp 431-439 in *Symbiotic Multi-robot Organisms: Reliability, Adaptability, Evolution*, P. Levi and S. Kernbach, Eds. Springer, 2010.