The Pi-puck extension board: a Raspberry Pi interface for the e-puck robot platform

Alan G. Millard[1], Russell Joyce[2], James A. Hilder[1], Cristian Fleșeriu[1], Leonard Newbrook[1], Wei Li[1], Liam J. McDaid[3], and David M. Halliday[1]

***Abstract*— This paper presents the Pi-puck extension board – an interface between the e-puck robot platform and a Raspberry Pi single-board computer that enhances the processing power, memory capacity, and networking capabilities of the robot at a low cost. It allows high-level control algorithms, wireless communication, and computationally expensive operations such as real-time image processing to be handled by a Raspberry Pi, while the e-puck's microcontroller deals with low-level motor control and sensor interfacing.**

Although two similar extension boards for the e-puck robot platform already exist, they are now out-dated and expensive in comparison. Our open-source hardware design and supporting software infrastructure offer an inexpensive upgrade to the e-puck robot, transforming it into the Pi-puck – a modern and flexible new platform for mobile robotics research.

I. Introduction

The e-puck robot [1] was first developed as an educational platform in 2004 at the École Polytechnique Fédérale de Lausanne, but was soon adopted as a tool for mobile robotics research. Thanks to its commercial availability and simple open-hardware design, the e-puck continues to be used by many institutions across the world in various fields including evolutionary robotics [2], reinforcement learning [3], and swarm robotics [4]. The platform also benefits from being supported by a number of simulation tools such as Enki [5], Webots [6], V-REP [7], and ARGoS [8].

Despite its small size, the base e-puck (with jumper board connected) features an array of sensors and actuators: two stepper motors, eight infra-red (IR) proximity sensors, a 3D accelerometer, three microphones, a speaker, a CMOS colour camera (640x480 pixel resolution), an IR remote receiver, and a number of LEDs. The e-puck also has a Bluetooth radio, which allows control programs to be wirelessly uploaded to the robot, and facilitates debugging via remote monitoring.

The success of the e-puck robot platform is partly attributable to the fact that extension boards, which connect to the robot via its expansion sockets, can be developed by third-parties. To date, a number of extension boards have been designed, including a ground sensor and colour LED communication turret [1], and range-and-bearing turret [9].

Although such extension boards augment the robot's sensing and actuation capabilities, the base e-puck has limited computational resources and networking abilities, thus restricting its usefulness as a research platform. The e-puck's dsPIC30F6014A microcontroller has just 8 KB RAM and 144 KB flash memory, and is clocked at a slow 60 MHz. This is particularly problematic for image processing applications, because only small segments of the on-board camera images can be stored in memory, and they can only be processed at low frame rates. Also, it is often necessary to log the robot's internal state and/or sensor data, for real-time monitoring and post-experiment analysis. In single-robot experiments, this can be achieved by connecting the e-puck to a computer via Bluetooth, but inherent constraints of the communication protocol cause problems for swarm-scale experiments.

In this paper we present the Pi-puck extension board – a novel hardware design that interfaces the e-puck robot with the popular Raspberry Pi single-board computer [10]. This extension board overcomes the aforementioned limitations of the e-puck by improving its computation, memory, networking, storage, and image processing capabilities. The open-source hardware design and supporting software infrastructure described in this paper offers a cheap and simple way of upgrading e-puck robots, which many institutions already own, transforming them into Pi-puck robots that have greater utility as a platform for both research and education.

The rest of this paper is structured as follows. In Section II we discuss existing extension boards for the e-puck robot that serve a similar purpose to the Pi-puck extension board, before motivating our use of the Raspberry Pi in Section III. Section IV describes the hardware design of the extension board, and how it interfaces with the e-puck hardware, while Section V details the supporting open-source software infrastructure we have developed. Potential applications of the extension board are discussed in Section VI, before closing with concluding remarks in Section VII.

II. Related Work

Two similar extension boards for the e-puck robot platform are known to exist – the Linux extension board developed at the Bristol Robotics Laboratory [11] (hereafter referred to as the BRL Linux extension board), and the Gumstix Overo COM turret [12]. Both of these boards run the Linux operating system, which confers a number of benefits. Firstly, the e-puck's dsPIC microcontroller can only be programmed in low-level C or assembly code, whereas a Linux platform can be programmed using many high-level languages such as C++ and Python. Moreover, the Linux operating system allows libraries and packages to be easily installed that

This work is funded by EPSRC grants EP/N007050/1 and EP/M025756/1

[1]York Robotics Laboratory, Department of Electronic Engineering, University of York, UK. alan.millard@york.ac.uk, https://www.york.ac.uk/robot-lab

[2]Department of Computer Science, University of York, UK

[3]School of Computing and Intelligent Systems, Ulster University, UK

can enhance user applications. A Linux environment also facilitates the use of high-level Wi-Fi protocols, allowing for remote terminal access to the robot via `ssh`, or file transfer via `scp`, and lifts the network topology constraints imposed by Bluetooth.

A. BRL Linux extension board

The BRL Linux extension board features an Atmel AT91 System-on-Chip (SoC), with an ARM processor that runs in parallel with the dsPIC microcontroller on the e-puck motherboard. Synchronous communication between the two is achieved via an SPI bus, allowing high-level control algorithms to be executed on the ARM processor. The extension board also provides the e-puck with a high-level network interface through the use of a USB Wi-Fi adapter. In addition, the board connects directly to the e-puck camera via the Atmel Image Sensor Interface and I^2C bus, so that full resolution images (640x480 pixels) can be processed, thus overcoming some of the image-sensing limitations of the base e-puck platform.

Since its inception, the BRL Linux extension board has proven to be a useful addition to the e-puck robot – supporting research into evolutionary adaptation [13], behavioural imitation [3], ethical robotics [14], and fault detection [15]. Its success demonstrates the utility of Linux-enabled robotic platforms, and has contributed to the longevity of the e-puck as a tool for scientific research.

Unfortunately, the hardware is now quite dated compared to modern standards – the ARM9 processor housed within the SoC only runs at 180 MHz, and the board features just 64 MB SDRAM. Also, the Linux kernel must be customised to support the extension board's hardware. Liu and Winfield [11] published their own modifications to kernel version 2.6.26 (released in 2008) for use with the board, but sadly the software is no longer actively maintained, so has fallen far behind the latest kernel version (4.12 at the time of writing). Similarly, the EmDebian Linux distribution that the board originally used was discontinued in 2014, so the software infrastructure provided is now also stuck in the past.

B. Gumstix Overo COM turret

In contrast to the integrated BRL Linux extension board, the Gumstix Overo COM turret mostly acts as an interface between the e-puck and a Gumstix Overo computer-on-module (COM) [16], which is a self-contained single-board computer. The turret is compatible with all Gumstix Overo COM models, but is sold with an EarthSTORM COM that features an ARM Cortex-A8 clocked at 800 MHz, and has 512 MB DDR. It is therefore significantly more powerful than the BRL Linux extension board.

Like the BRL Linux extension board, the Gumstix Overo COM turret has enhanced the e-puck's usefulness as a research platform, and has been used for the automatic design of robot controllers [17], implementing virtual sensors [18], and running ARGoS controllers on real robots [19]. However, it suffers from similar software maintenance issues – the turret connects to the e-puck's camera via the OMAP Camera Interface Subsystem, necessitating customisations to the Linux kernel to provide drivers for the camera sensor. GCtronic published the requisite modifications to kernel version 2.6.32 (released in 2009), but have not kept up with newer releases of the Linux kernel.

Being stuck with an outdated operating system makes it difficult to install versions of packages and libraries that modern software is dependent on, or drivers for contemporary hardware such as USB Wi-Fi adapters produced in recent years. While it is certainly possible to modify the latest version of the Linux kernel to work with old hardware, it is a laborious process that requires specialist knowledge of kernel customisation and driver development. Major revisions of Linux that significantly reorganise the structure of the kernel exacerbate the problem, as they preclude straightforward incremental modifications. Unless the creators of a Linux extension board (or the research community) are able to actively maintain the necessary customisations to the Linux kernel and apply them to new releases, the system will inevitably become outdated.

III. Raspberry Pi

Instead of designing another integrated solution like the BRL Linux extension board using modern hardware components, we have developed the Pi-puck extension board to interface an existing single-board computer with the e-puck, in the spirit of the Gumstix Overo COM turret. Although many different single-board computers exist, we have opted for the Raspberry Pi because it is very popular and well-supported. Since its introduction in 2012 various models have been released, including the Raspberry Pi Zero W, which we have specifically chosen for use with the Pi-puck platform due to its low cost, minimal power consumption, integrated wireless capabilities, and small physical footprint.

The Raspberry Pi Zero W uses a Broadcom BCM2835 SoC that houses a 1 GHz ARM11 processor, and has 512 MB RAM, so is similar in specifications to the Gumstix Overo EarthSTORM COM. However, a major advantage of the Raspberry Pi Zero W is its MIPI Camera Serial Interface that supports the Raspberry Pi camera module v2, which uses a Sony IMX219 8 megapixel sensor. In combination with the on-board GPU, this makes the Raspberry Pi Zero W a superior hardware platform for real-time image processing. It also features built-in 802.11n Wi-Fi and Bluetooth 4.1, so USB adapters are not required for wireless communication.

Like the BRL Linux extension board and Gumstix Overo COM turret, the Raspberry Pi uses a microSD card for non-volatile storage, upon which the Linux kernel and root file system reside. One of the major advantages of using the Raspberry Pi is that the Raspberry Pi Foundation actively maintains an up-to-date version of the Linux kernel that is customised to support their single-board computers, thus freeing us from the burden of modifying the kernel for our own hardware. The Pi-puck runs Raspbian Jessie Lite (currently on kernel version 4.9), which is a headless Debian-based distribution of the Linux operating system that provides hardware floating-point support.

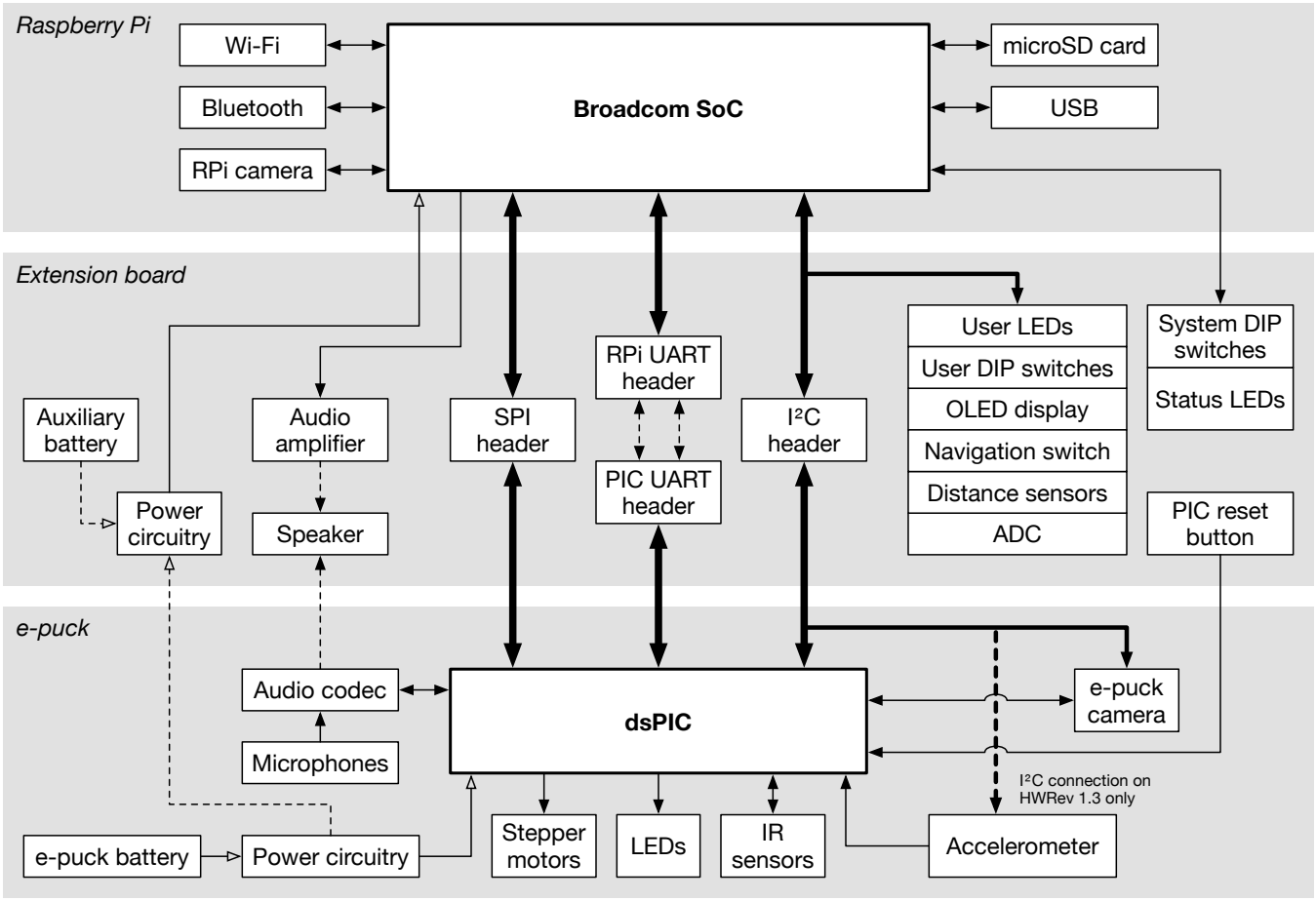

Fig. 1. Overview of the interfaces between the Raspberry Pi, Pi-puck extension board, and e-puck hardware. Dashed lines denote optional connections.

IV. Pi-puck extension board

The Pi-puck extension board provides connections between a Raspberry Pi and an e-puck, using a format like other extension 'turret' boards. All of the necessary circuitry is included to effectively support a Raspberry Pi, and allow it to control the robot. The hardware design files necessary for other institutions to manufacture their own extension boards are open-source, and freely available online [20].

A. Overview

The extension board is the same diameter as the e-puck's main circuit board, with three screw holes allowing it to be mounted above the robot and any other extension boards, and connects to the e-puck through the two 40-pin expansion sockets on its base. The Raspberry Pi Zero W connects to the top of the extension board via a 40-pin, 2.54 mm pitch header, which suspends it horizontally above the robot, with its SoC, Micro-USB and Mini-HDMI sockets, and camera connector on top, as shown in Figure 2.

The board is designed to be used as the topmost board in a stack of extension turrets that sit above the e-puck, and therefore mimics some of the features of the standard jumper board that is usually placed at the top of the stack, including a speaker and a dsPIC reset button. All unused signals on the e-puck's JE1 expansion connector (everything except UART `RS_RX/TX`) are also bridged across, allowing the dsPIC to continue to use them as if the jumper board were connected.

Although our extension board is primarily intended to be used with the Raspberry Pi Zero W, any board that conforms to the same pin-out as the standard 40-pin Raspberry Pi connector will be compatible. For example, the more powerful Raspberry Pi 3 may be used, or even the ZynqBerry [21], which would augment the e-puck with an FPGA for specialised hardware acceleration. We have designed a separate PCB that allows these larger boards to be mounted vertically on top of the robot, without any need to modify their default pin headers, as shown in Figure 2.

B. Communication interfaces

The Pi-puck extension board enables the Raspberry Pi to communicate with the dsPIC microcontroller and the e-puck's other hardware, as well as additional extension boards via I^{2}C, SPI and UART serial connections, as illustrated in Figure 1.

The I^{2}C and SPI buses allow the Raspberry Pi to act as a master device for sending commands to, and reading data from, slave software on the dsPIC. Both connections are provided so that the user may choose whichever communication method best suits their needs. Through the I^{2}C bus,

the Raspberry Pi can directly interface with the LSM330 accelerometer/gyroscope on e-puck HWRev 1.3, the e-puck's camera configuration registers, and other extension boards such as the ground sensor [1] and range-and-bearing [9] boards. The I^2C and SPI buses are broken out to a 12-pin PicoBlade connector, allowing additional peripherals to be connected, or for easy access when debugging signals.

For simple asynchronous communication, such as that used by the ARGoS controllers described in Section V, 3.3 V UART signals can be sent between the Raspberry Pi and the dsPIC. Transmit and receive UART signals from both devices are broken out to adjacent 2.54 mm pitch header pins on the extension board, which can either be connected to external hardware for monitoring and debugging, or bridged together with jumpers to facilitate communication between the Raspberry Pi and dsPIC. These pins supplant the RS-232 UART header on the base e-puck board.

C. Power circuitry

The Raspberry Pi is powered from the Pi-puck extension board using the 5 V power pins on its 40-pin header. Due to the low energy consumption of the Raspberry Pi Zero W, the entire system can be powered from the e-puck battery without any need for external cables or batteries. The user can control and monitor the power supply to the Raspberry Pi, independently to that of the e-puck, via the extension board's power switch and LEDs.

1) Voltage regulators: Like the Gumstix Overo COM turret, the voltage of the e-puck battery is stepped-up to a regulated 5 V supply, in order to reliably power the Raspberry Pi. The voltage regulator circuitry is controlled by the `ENABLE_LDO` signal from the e-puck, so power will be shut-off if the battery voltage falls below a safe level. A simpler alternative would be to attach the Raspberry Pi directly to the e-puck battery power, but this would cause it, and any attached USB devices, to operate below their rated voltages, potentially causing them to malfunction.

The Pi-puck extension board also features a regulated 3.3 V supply (independent to that of the Raspberry Pi) for powering its own hardware, which we have broken out to the 12-pin PicoBlade connector, along with ground and 5 V connections, so that auxiliary peripherals may be powered.

2) Battery voltage monitoring: The Pi-puck extension board features an analogue-to-digital converter (ADC) that allows the Raspberry Pi to monitor the voltage of the e-puck battery, and/or that of an auxiliary battery. This ADC is an I^2C device, so the e-puck's dsPIC may also communicate with it directly via the I^2C bus to monitor the battery voltages, if desired. We have broken out two more ADC inputs, with 3.3 V and ground, to a 4-pin PicoBlade connector, so that further analogue signals can be monitored.

The `BATT_LOW` signal from the e-puck's battery control circuitry can optionally be connected to a GPIO pin on the Raspberry Pi, to allow basic power monitoring from software on the ARM without needing to monitor the voltage by polling the ADC. This gives the user a way to easily detect when the e-puck determines its battery to be 'low' (below 3.3 V), allowing a graceful Linux shut-down to be performed before the power is cut off, as explained in Section V.

3) Auxiliary power connection: In addition to powering the Raspberry Pi from the e-puck battery, an auxiliary battery can be connected to the input of the voltage regulator via a standard 2-pin JST connector on the Pi-puck extension board, as shown in Figure 2. This feature can be used to provide power-hungry boards such as the Raspberry Pi 3 with more current than the e-puck can supply, or simply to decrease load on the e-puck battery for longer experimental runs.

A dedicated chip is used to monitor the voltage of the auxiliary battery, and generates a `BATT_LOW` signal when the voltage drops below a certain threshold. There is no need for another explicit `ENABLE_LDO` signal, as the voltage regulator will automatically turn off its output when the voltage of the auxiliary battery drops below a safe level. The extension board is designed such that the user may switch the voltage input, `ENABLE_LDO` signal, and `BATT_LOW` signal between the e-puck or auxiliary power supplies using a set of three jumpers positioned beneath the Raspberry Pi.

D. LEDs and DIP switches

The extension board features eight system status LEDs, as shown in Figure 2. The 5 V and 3.3 V power LEDs are connected to the outputs of the two voltage regulators, while another LED displays the status of the `BATT_LOW` signal that is being sent to the Raspberry Pi from either the e-puck or the auxiliary battery controller. The remaining five LEDs are directly connected to GPIO pins on the Raspberry Pi, four of which are configured in software to display microSD card activity, Wi-Fi transmit and receive status, and a heartbeat signal once Linux has booted. The behaviour of the final LED is left for the user to define.

A set of four system control DIP switches is also directly connected to GPIO pins on the Raspberry Pi. We provide scripts that configure the LEDs and DIP switches for system control functionality (as described in Section V), but the user may configure them to perform other functions if desired.

In addition to the system LEDs and DIP switches, a further four user LEDs and DIP switches are connected to the Raspberry Pi via a GPIO expander, to provide additional input and output signals. The GPIO expander is an I^2C device, so these LEDs and switches can also be interfaced with the e-puck's dsPIC microcontroller via the I^2C bus.

E. OLED display and menu controls

The Pi-puck extension board features a 96x16 pixel OLED display (shown in Figure 2), which can be used to provide feedback to the user. It can be configured via the I^2C bus to display information such as the ID of the robot, the IP address of the Raspberry Pi, the voltage of the e-puck or auxiliary batteries (read via the ADC), or real-time sensor readings, for example. The adjacent navigation switch and two push buttons can be used in conjunction with the display to implement a simple menu system. The user may then use these controls to select the information to be displayed, or even choose between different control programs stored on the microSD card.

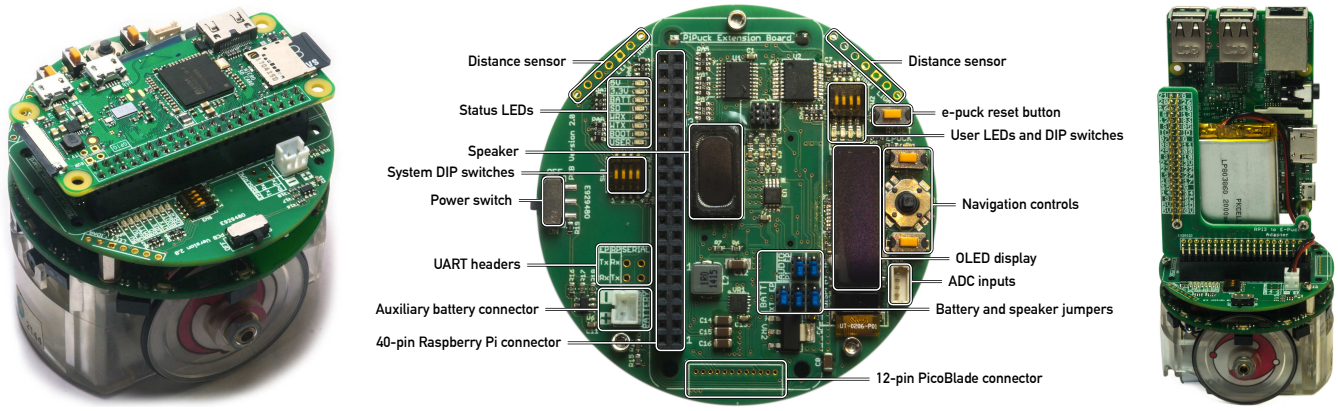

Fig. 2. *From left to right:* Pi-puck with Raspberry Pi Zero W; Prototype of the Pi-puck extension board (some optional components left unsoldered); Pi-puck with Raspberry Pi 3 and auxiliary battery.

F. Speaker

The PWM audio output of the Raspberry Pi is connected to a speaker on the extension board via an audio amplifier, so that the user can easily play sound directly from Linux. This uses the same audio output as the 3.5 mm connector on a full-sized Raspberry Pi, allowing the standard analogue audio driver to be used. Alternatively, the speaker can be driven by the the e-puck's audio codec, in order to mimic the functionality of the jumper board. The user can switch between these two input sources with jumpers located beneath the Raspberry Pi, as shown in Figure 2.

G. Long-range distance sensors

Like the Gumstix Overo COM turret, the Pi-puck extension board optionally features two long-range distance sensors positioned at the front of the robot. We use VL53L0X time-of-flight sensors that emit pulses of IR laser light to measure the distance to the nearest surface. These sensors interface with the Raspberry Pi via I^2C, and can measure the distance of objects up to 2 m away. This allows the Pi-puck to detect obstacles at a far greater distance than is possible with the e-puck's IR proximity sensors, or to construct an internal map of its surroundings by taking repeated measurements from a series of different positions and orientations.

H. Power consumption

To assess the power consumption of the Pi-puck platform, a bench power supply was used to provide a fixed 4.2 V to the battery connector of an e-puck that had a Pi-puck extension board and Raspberry Pi Zero W attached. The total current draw of the system was then measured while various tasks were performed on the Raspberry Pi, with the base e-puck robot in three different states of operation – off, on (idle), and moving while performing simple obstacle avoidance. The results from these tests are presented in Table I.

The current draw when the e-puck is turned off demonstrates the power consumption of the Pi-puck extension board and Raspberry Pi Zero W in isolation. The current draw when the e-puck is idle corresponds to best-case power consumption for the Pi-puck, while figures for when the e-puck is performing obstacle avoidance represent the expected current draw in a more realistic use-case. When the e-puck is moving, the stepper motors account for the majority of the power consumption (around 450 mA combined), causing the base e-puck current draw to outweigh that of the Pi-puck extension in all but the most demanding scenarios (for example, streaming video).

The Raspberry Pi Zero W was considered to be idle once it had finished booting into Raspbian, and was connected to Wi-Fi except where indicated. The difference in current draw between an idle state with Wi-Fi off and one with Wi-Fi on shows that the wireless hardware can incur around a 40 mA overhead simply when enabled and connected to an access point. With the CPU at maximum utilisation, up to an additional 160 mA is used compared to when it is idle.

The power consumed by transmitting and receiving data over Wi-Fi was tested using Linux's `netcat` utility to send and receive packets via UDP. As shown in Table I, receiving data draws very little current in comparison to idle. Conversely, transmitting consumes a lot of power, due to the high CPU utilisation caused by constructing packets to send.

The power consumption of the Raspberry Pi camera module and GPU were also tested by recording a 1080p H.264 video stream to the microSD card, using the `raspivid` command. This drew more current than the CPU utilisation test, although CPU usage was not at 100% while recording video. As a final power stress-test, a 1080p H.264 video from the camera was streamed via HTTP over Wi-Fi to a remote computer using VLC, causing very high usage of CPU, GPU and network, as well as powering the camera module. Predictably, this consumed more power than any other test, but the e-puck battery is still able to provide sufficient current to allow high-quality video to be streamed for off-board processing, if necessary.

In addition to measuring the instantaneous current draw under various conditions, we tested the battery life of a standard e-puck robot versus the Pi-puck. An e-puck robot fitted with a 1600 mAh battery performing simple obstacle avoidance (continually driving) will survive for approximately 210 minutes. Running the same experiment with the

TABLE I

POWER CONSUMPTION OF THE PI-PUCK WITH A RASPBERRY PI ZERO W. VALUES ARE IN mA.

	Off	Idle (Wi-Fi off)	Idle	Max CPU	UDP transmit	UDP receive	Recording video	Streaming video
e-puck off	0	160	200	360	430	220	430	665
e-puck idle	110	260	300	460	520	310	540	745
e-puck moving	560	700	715	875	960	750	965	1210

Pi-puck extension board and Raspberry Pi Zero W attached, while the Raspberry Pi is connected to Wi-Fi but otherwise idle, the system lasts for approximately 130 minutes.

If an auxiliary battery is used, the Raspberry Pi and other extension board peripherals will run on their own power source and will not drain the e-puck battery. Although this will increase the longevity of the e-puck battery, the Pi-puck's lifespan will remain limited by the maximum battery life of the base e-puck.

While the power consumption figures presented here are limited in scope, they are comparable to those quoted for the BRL Linux extension board [11] and Gumstix Overo COM turret [12]. Although the actual power usage and battery life will vary depending on the specific e-puck and battery used, the application, and many other factors, they offer an indication of what can be expected when using a Pi-puck robot compared to the standard e-puck platform.

I. Cost

A major advantage of the Pi-puck extension board is that the production cost is far lower than the retail price of the Gumstix Overo COM turret, which is approximately £640 [22]. The cost is more comparable to that of the BRL Linux extension board, which had a unit cost of £80 (based on out-sourcing the manufacture of 50 boards) [11].

We have assembled our boards in-house using PCBs produced by Eurocircuits [23], who charge £35.21, £12.30, £8.08, or £4.57 per board for batches of one, five, ten, or thirty boards, respectively. The components that are soldered onto each PCB cost around £40 per board. However, many of them are optional and may be omitted to cut costs.

In addition to the cost of manufacturing the extension board, the user must also purchase a Raspberry Pi Zero W (available for £9.60), and a microSD card (recommended minimum capacity of 4 GB) from which the Linux operating system runs. Finally, if desired, the official Raspberry Pi camera module can be purchased for £21.

Note that these are only indicative costs – the actual costs will vary depending upon the suppliers and scale of production. However, a single complete extension board with all of the necessary peripherals (including the camera module) can be constructed for around £110, excluding the cost of labour for assembly.

V. SOFTWARE INFRASTRUCTURE

In addition to the hardware design, we also provide an open-source software infrastructure [20], to make it easier for other users to get started with the extension board. This includes the files required to support the hardware, and example code for controlling the e-puck robot from the Raspberry Pi in different ways.

A. Hardware support

As discussed in Section III, there is no need to modify the Linux kernel or root file system to support the Raspberry Pi hardware, as this is already handled by the Raspbian Linux distribution provided by the Raspberry Pi Foundation. However, some optional software customisations can be made to fully utilise the hardware interfaced with the Raspberry Pi. For example, although the user-programmable LEDs and DIP switches on the Pi-puck extension board can be interfaced with directly as generic I^2C or GPIO devices via the `sysfs` pseudo file system, device tree overlays can be used to associate them with specialised Linux kernel drivers.

We provide an example device tree overlay that associates the status LEDs with the `leds-gpio` kernel driver, and configures them to display system status information, as described in Section IV. Similarly, we include another example device tree overlay that associates the DIP switches with the `gpio-keys` kernel driver, causing them to generate keyboard events when they change state.

Alternatively, the state of the DIP switches can simply be continually monitored in software. We provide a sample Python script that triggers a graceful Linux shut-down when one of the DIP switches is set to the *on* position, which is useful when remote terminal access to the robot is unavailable. A second Python script is used to enable/disable the `getty` service on the Raspberry Pi UART via another DIP switch, so that the user can alternate between serial terminal access to the Raspberry Pi, and allowing the dsPIC to communicate with the Raspberry Pi via the UART.

As discussed in Section IV, the `ENABLE_LDO` signal will cut power to the Raspberry Pi when the battery is critically low, so we also provide a Python script that monitors the `BATT_LOW` signal, and initiates a graceful Linux shut-down before this happens to prevent corruption of the file system on the microSD card.

Further device tree overlays are used to remap the PWM output of the Raspberry Pi to the GPIO pins connected to the audio amplifier, to set up the display navigation controls, and to associate the ADC with a kernel driver. We provide sample code that demonstrates how to interact with these devices, and how to communicate with the VL53L0X long-range distance sensors and OLED display via I^2C.

The example device tree overlays and Python scripts may simply be copied to the appropriate locations on the microSD card, so are very easy to customise and update.

B. Robot Operating System

ROS (Robot Operating System) [24] is an open-source collection of libraries and tools designed to facilitate the development of software for robot platforms. Its architecture

comprises a distributed network of processes (called nodes) that can be designed and developed in isolation and then integrated at run-time, thus encouraging code re-use within the robotics research community.

Both Python and C++ ROS drivers already exist for the base e-puck platform, which provide a hardware abstraction layer for the robot's sensors and actuators. This allows robot controller code to be written in a high-level language, and enables easy integration with other ROS nodes. These drivers comprise firmware for the dsPIC that sends sensor data via Bluetooth to a desktop PC running a ROS node, which publishes the data as a ROS topic. The ROS node also subscribes to a ROS topic that receives actuator commands, and forwards them to the dsPIC.

By interfacing a Raspberry Pi with the e-puck, we are able to run this ROS node directly on the Linux-enabled robot, rather than on a separate machine. We have ported the C++ driver developed by GCtronic [25] to work with our extension board, such that the ROS node communicates with the dsPIC directly through the UART, instead of remotely via Bluetooth. This ROS node may then integrate with other nodes running on separate machines, via Wi-Fi. To demonstrate the usefulness of this software infrastructure, we provide example code that uses the `rviz` and `gmapping` ROS nodes to build a 2D occupancy grid from IR proximity sensor data collected by the e-puck robot.

C. ARGoS controllers

ARGoS [8] is a multi-robot simulator that is widely-used within the swarm robotics research community, and has built-in support for the e-puck robot platform. Garattoni et al. [19] have recently extended this support to allow ARGoS controllers to be executed on real e-pucks that are fitted with a Gumstix Overo COM turret. This software infrastructure is immensely beneficial, as it allows an experimenter to quickly prototype robot controller code in simulation, and then deploy it to real robot hardware without modification.

In order to reap the same benefits, we have ported this software infrastructure to run on e-puck robots fitted with our Pi-puck extension board. The ARGoS controller code (written in C++) runs under Linux, and communicates with slave firmware running on the dsPIC via the UART. Like ROS, this provides a hardware abstraction layer for the e-puck's sensors and actuators, allowing controller code to be written at a higher level than can be achieved when programming the dsPIC directly.

VI. Applications

The Pi-puck platform's combination of capable hardware and the Linux operating system facilitates the implementation of useful experimental infrastructure, including: remote operation, data logging for real-time monitoring and post-experiment analysis, integration with a tracking system for the implementation of virtual sensors, and inter-robot communication via Wi-Fi, for example.

In addition, there are a few applications that the Pi-puck extension board is better-suited to than the BRL Linux extension board and Gumstix Overo COM turret.

A. Image processing

While the e-puck camera's parallel interface is incompatible with the Raspberry Pi hardware, the Raspberry Pi features a more modern serial camera interface that supports higher quality sensors, as discussed in Section III. Thanks to hardware acceleration from the on-board GPU, use of the Raspberry Pi camera module incurs little CPU overhead. If desired, the e-puck camera can still be used via the dsPIC.

The Sony IMX219 sensor can capture still images up to a resolution of 8 megapixels (3280x2464), or video at 1080p (1920x1080 pixel resolution). Processing images of such a high resolution in real-time is computationally expensive, and may be beyond the capabilities of the Raspberry Pi Zero W, depending on the application. However, images can be efficiently downscaled using the on-board GPU before being processed, to reduce computational expense. Alternatively, compressed video can be streamed to an external machine via Wi-Fi for off-board processing, the results of which can be transmitted back to the Pi-puck to provide feedback to a control program. If intensive on-board image processing is absolutely necessary, a Raspberry Pi 3 may be used with the Pi-puck extension board to provide more computational power.

The Raspberry Pi camera module may be mounted in a front-facing configuration, or combined with an appropriate lens or mirror to produce an omnidirectional field of vision. Omnidirectional images can either be unwrapped in real-time to create a 2D panorama, or processed directly for simple range-and-bearing calculations, using libraries such as OpenCV [26]. Additionally, the high-quality image sensor allows fiducial markers such as ArUco tags [27] to be decoded at a greater distance than would be possible with the e-puck's low-resolution camera, providing the Pi-puck with the ability to uniquely identify objects in its environment.

B. Hardware acceleration

The Raspberry Pi's VideoCore IV GPU comprises 12 vector processors called Quad Processing Units (QPUs), offering the possibility for hardware acceleration via the exploitation of GPGPU. Although the Raspberry Pi GPU is not supported by OpenCL, other GPGPU programming environments are available. For example, QPULib [28] is a programming language and compiler for the Raspberry Pi QPUs that is implemented in C++, and PyVideoCore [29] is a Python library that attempts to achieve a similar goal.

This form of hardware acceleration could be used to efficiently implement on-board evolutionary algorithms [30] or artificial neural networks [31] in parallel. It would be particularly beneficial for accelerating the implementation of on-board simulations, which are computationally expensive, but have been shown to be useful for embodied evolution [32], prediction of robot behaviour [14], and fault detection [33].

Poulding [34] demonstrated that GPGPU could be used to execute several parallel instances of a low-fidelity on-board robot simulator, allowing many repeat runs to performed in real-time for such applications. Jones et al. [35] later

showed that GPGPU could also be used to accelerate high-fidelity simulations on low-power GPU chips for mobile robot platforms. The Raspberry Pi 3 GPU is clocked at a higher speed than that of the Raspberry Pi Zero W, so could potentially be used in conjunction with the Pi-puck extension board to produce similar results.

Hardware acceleration could also be achieved through the use of an FPGA, with hardware such as the ZynqBerry [21], as mentioned in Section IV. Custom cores for accelerating specific algorithms could be created in a hardware description language, or using technologies such as OpenCV through Xilinx Vivado High-Level Synthesis [36] for image processing tasks.

VII. Conclusions

The Pi-puck extension board presented in this paper can be used for research into single-robot, multi-robot, or swarm robotic systems. The board interfaces the e-puck platform with a Raspberry Pi single-board computer, which offers a modern and well-supported alternative to existing hardware designs that allow the robot to run Linux. Our open-source hardware and software infrastructure provide a cost-effective and convenient upgrade that breathes new life into the e-puck platform, transforming it into the Pi-puck, which has enhanced utility as a tool for experimental robotics research.

The prototype hardware presented in this paper will continue to be refined as it is used for education and research, in order to further the Pi-puck's usefulness as a robot platform. The latest version of our hardware design and supporting software infrastructure can be found on our website [20].

References

[1] F. Mondada, *et al.*, "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.

[2] S. Koos, *et al.*, "The transferability approach: crossing the reality gap in evolutionary robotics," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 1, pp. 122–145, 2013.

[3] M. D. Erbas, *et al.*, "Embodied imitation-enhanced reinforcement learning in multi-agent systems," *Adaptive Behavior*, vol. 22, no. 1, pp. 31–50, 2014.

[4] J. Chen, *et al.*, "Occlusion-based cooperative transport with a swarm of miniature mobile robots," *IEEE Transactions on Robotics*, vol. 31, no. 2, pp. 307–321, 2015.

[5] Enki, a fast 2D robot simulator. [Online]. Available: https://github.com/enki-community/enki

[6] O. Michel, "Cyberbotics Ltd. Webots™: professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.

[7] E. Rohmer, *et al.*, "V-REP: A versatile and scalable robot simulation framework," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1321–1326.

[8] C. Pinciroli, *et al.*, "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intelligence*, vol. 6, no. 4, pp. 271–295, 2012.

[9] Á. Gutiérrez, *et al.*, "Open e-puck range & bearing miniaturized board for local communication in swarm robotics," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2009, pp. 3111–3116.

[10] Raspberry Pi. [Online]. Available: https://www.raspberrypi.org

[11] W. Liu and A. F. T. Winfield, "Open-hardware e-puck Linux extension board for experimental swarm robotics research," *Microprocessors and Microsystems*, vol. 35, no. 1, pp. 60–67, 2011.

[12] Overo Extension – GCtronic Wiki. [Online]. Available: http://www.gctronic.com/doc/index.php/Overo_Extension

[13] N. Bredeche, *et al.*, "Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 18, no. 1, pp. 101–129, 2012.

[14] A. F. T. Winfield, *et al.*, "Towards an ethical robot: internal models, consequences and ethical action selection," in *Conference Towards Autonomous Robotic Systems*, 2014, pp. 85–96.

[15] A. G. Millard, *et al.*, "Run-time detection of faults in autonomous mobile robots based on the comparison of simulated and real robot behaviour," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 3720–3725.

[16] Gumstix Overo COM series. [Online]. Available: https://www.gumstix.com/support/hardware/overo

[17] G. Francesca, *et al.*, "AutoMoDe: A novel approach to the automatic design of control software for robot swarms," *Swarm Intelligence*, vol. 8, no. 2, pp. 89–112, 2014.

[18] A. Reina, *et al.*, "Augmented reality for robots: virtual sensing technology applied to a swarm of e-pucks," in *NASA/ESA Conference on Adaptive Hardware and Systems*, 2015, pp. 1–6.

[19] L. Garattoni, *et al.*, "Software infrastructure for e-puck (and TAM)," Université Libre de Bruxelles, Tech. Rep. TR/IRIDIA/2015-004, 2015.

[20] Pi-puck extension board. [Online]. Available: https://www.york.ac.uk/robot-lab/pi-puck

[21] ZynqBerry. [Online]. Available: https://shop.trenz-electronic.de/en/Products/Trenz-Electronic/TE07XX-Zynq-SoC/TE0726-Zynq-SoC/

[22] GCtronic – Shop: e-puck extension for Gumstix Overo COM. [Online]. Available: http://www.gctronic.com/shop.php

[23] Eurocircuits – Online PCB prototype and small series specialist. [Online]. Available: http://www.eurocircuits.com

[24] M. Quigley, *et al.*, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[25] GCtronic C++ ROS driver for e-puck robot. [Online]. Available: https://github.com/gctronic/epuck_driver_cpp

[26] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, pp. 120–123, 2000.

[27] S. Garrido-Jurado, *et al.*, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280 – 2292, 2014.

[28] QPULib – A language and compiler for the Raspberry Pi GPU. [Online]. Available: https://github.com/mn416/QPULib

[29] PyVideoCore – A Python library for GPGPU on Raspberry Pi boards. [Online]. Available: https://github.com/nineties/py-videocore

[30] O. Maitre, *et al.*, "Efficient parallel implementation of evolutionary algorithms on GPGPU cards," in *European Conference on Parallel Processing*, 2009, pp. 974–985.

[31] Z. Luo, *et al.*, "Artificial neural network computation on graphic process unit," in *IEEE International Joint Conference on Neural Networks*, vol. 1, 2005, pp. 622–626.

[32] P. J. O'Dowd, *et al.*, "Towards accelerated distributed evolution for adaptive behaviours in swarm robotics," in *Proceedings of Towards Autonomous Robotic Systems (TAROS)*. University of Plymouth, 2010, pp. 169–175.

[33] A. G. Millard, "Exogenous fault detection in swarm robotic systems," Ph.D. dissertation, University of York, 2016.

[34] S. M. Poulding, "The use of automated search in deriving software testing strategies," Ph.D. dissertation, University of York, 2013.

[35] S. Jones, *et al.*, "Mobile GPGPU acceleration of embodied robot simulation," in *Artificial Life and Intelligent Agents Symposium*, 2014, pp. 97–109.

[36] Xilinx – Vivado High-Level Synthesis. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html