The Pi-puck Ecosystem: Hardware and Software Support for the e-puck and e-puck2

Jacob M. Allen[1][0000−0001−9425−5917], Russell Joyce[1][0000−0002−6773−3837], Alan G. Millard[2][0000−0002−4424−5953], and Ian Gray[1][0000−0003−1150−9905]

[1] Department of Computer Science, University of York, York, UK
{jma542,russell.joyce,ian.gray}@york.ac.uk
[2] Lincoln Centre for Autonomous Systems, University of Lincoln, Lincoln, UK
amillard@lincoln.ac.uk

Abstract. This paper presents a hardware revision of the Pi-puck extension board that now includes support for the e-puck2. This Raspberry Pi interface for the e-puck robot provides a feature-rich experimentation platform suitable for multi-robot and swarm robotics research. We also present a new expansion board that features a 9-DOF IMU and XBee interface for increased functionality. We detail the revised Pi-puck hardware and software ecosystem, including ROS support that now allows mobile robotics algorithms and utilities developed by the ROS community to be leveraged by swarm robotics researchers. We also present the results of an illustrative multi-robot mapping experiment using new long-range Time-of-Flight distance sensor modules, to demonstrate the ease-of use and efficacy of this new Pi-puck ecosystem.

1 Introduction

The e-puck robot platform [15] is widely-used for mobile robotics research, and is a popular choice for swarm robotics due to its size and commercial availability. Three hardware revisions of the original e-puck (v1.1–1.3) have been released commercially by GCtronic and EPFL since it was first developed in 2004, followed by the release of the e-puck2 in 2018. Our Pi-puck[3] extension board allows a Raspberry Pi single-board computer to be interfaced with an e-puck or e-puck2 to enhance its capabilities. It features a range of augmentations over the base e-puck design, including greater computational power, and increased communication, sensing and interfacing abilities. The first prototype design of the Pi-puck extension board was created by the York Robotics Laboratory (YRL) at the University of York, and was published in 2017 [12], before the release of the e-puck2. The latest version of the hardware was developed as a collaboration between YRL and GCtronic to support both the e-puck and e-puck2, and is available to purchase from GCtronic and its distributors.

Nedjah and Junior [17] argue that there is an urgent need to standardise many aspects of swarm robotics research, so that faster progress can be made towards

[3] https://www.york.ac.uk/robot-lab/pi-puck

real-world applications. In particular, they call for standardisation of hardware and software – the Pi-puck aims to provide a common hardware and software ecosystem for researchers that wish to run embedded Linux and associated software on e-puck robots. The board was designed to replace the now deprecated Linux extension board developed by the Bristol Robotics Laboratory [9], and the Gumstix Overo COM turret [2]. The recently published Xpuck [7] is an e-puck extension that is similar in spirit to the Pi-puck – extending the e-puck with a powerful ODROID-XU4 single-board computer through custom hardware. This greatly enhances the robot's computational capabilities, but comes at the cost of size (the form factor of the XU4 is similar to that of the Raspberry Pi 3 or 4) and power consumption, necessitating the use of an auxiliary battery. The Xpuck was also developed prior to the release of the e-puck2, and its communication with the robot relies on an SPI bus that is not present on the e-puck2's expansion connector. In contrast, the Pi-puck has been designed around the Raspberry Pi Zero to provide a modest compute upgrade while minimising size and energy usage, and primarily uses I^2C for communication with the base robot, which is compatible with both the e-puck and e-puck2.

Nedjah and Junior [17] also encourage the use of Robot Operating System (ROS) [21] to facilitate standardisation. Although ROS has become the *de facto* standard for robotics middleware in single-robot and multi-robot studies, the swarm robotics research community has generally been reluctant to adopt it. This can partly be attributed to the fact that many swarm hardware platforms are microcontroller-based, so cannot run ROS on-board [24], however ROS integration can still be achieved via wireless communication and the `rosserial` interface – see the Mona [25] and HeRo [22] swarm platforms. Additionally, the ROS communication model is inherently centralised, which is antithetical to the philosophy of many swarm algorithms.

Rudimentary ROS support was implemented for the previous version of the Pi-puck [12], and the software infrastructure has now been updated to provide ROS Melodic support for the latest hardware revision, opening the door to a large body of existing work developed by the ROS community. This paper discusses the ROS drivers and ecosystem developed for the Pi-puck platform, and how they can be leveraged by other swarm robotics researchers. We recognise that the use of ROS may not be appropriate in some cases, and Pi-puck users may instead opt for lighter-weight software frameworks designed specifically for swarm robotics research such as Buzz [19], OpenSwarm [24], or SwarmTalk [26]. For resource-constrained experiments, the Pi-puck could be programmed to work with these frameworks instead of ROS.

2 Hardware Changes

There have been several major changes to the design of the Pi-puck hardware since it was first published in 2017 [12], which add new features, implement support for the e-puck2, and improve the stability of the platform for large-scale production. Many of these changes were made after consultation with members of

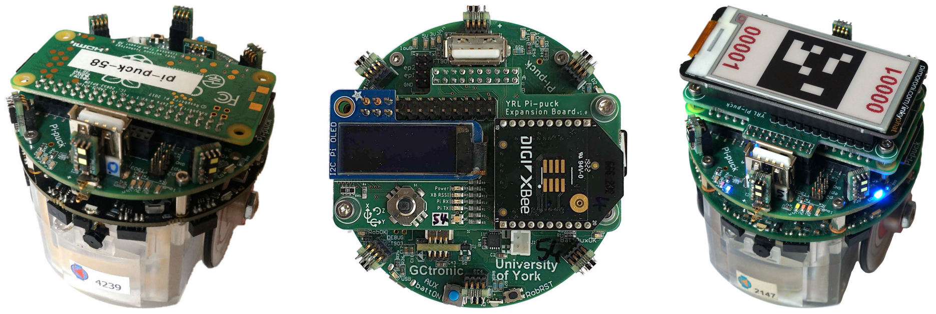

Fig. 1. *Left*: Pi-puck on e-puck2, with six Time-of-Flight distance sensor modules. *Centre:* Pi-puck board with YRL Expansion Board, XBee, and OLED display. *Right*: Pi-puck on e-puck1, with expansion board and e-ink pHAT showing an ArUco tag.

the swarm robotics community, and with GCtronic as the primary manufacturer of the e-puck robot. Figure 1 shows the latest version of the Pi-puck extension board connected to an e-puck robot, along with a further expansion board and attached hardware (detailed in Section 2.1), as well as six Time-of-Flight (ToF) distance sensor modules.

A full block diagram detailing the hardware of the Pi-puck platform is shown in Figure 2, which includes the components added to the robot on the extension board itself, as well as the major communication buses between the Raspberry Pi, extension board hardware, and the base e-puck robot. The Raspberry Pi Zero WH is specifically supported, due to its wide availability, low cost, minimal power consumption, integrated wireless capabilities, and small physical footprint. However, it is feasible that other Raspberry Pi and compatible boards could be used with the Pi-puck if additional mechanical support and wiring were added.

Raspberry Pi Support: The first fundamental change is in the mounting of the Raspberry Pi board, which is now face-down. This allows Raspberry Pi boards with pre-soldered headers to be used, which are easier to acquire in large quantities. A small micro-USB shim has been added to allow USB communication between the Raspberry Pi and Pi-puck extension board, and an integrated USB hub allows up to three devices to utilise this connection simultaneously. The Raspberry Pi UART is also accessible through a micro-USB port on the Pi-puck board, via a USB-UART converter, allowing a text console on the Raspberry Pi to be accessed without additional hardware.

Sensor Modules: Six 4-pin sockets are provided around the edge of the robot for connecting optional I^{2}C sensor modules, allowing for a range of flexible options for experimentation. The mapping application detailed in Section 5 uses custom-designed, open-source[4] distance sensor boards based around the

[4] https://github.com/yorkrobotlab/pi-puck-tof-sensor

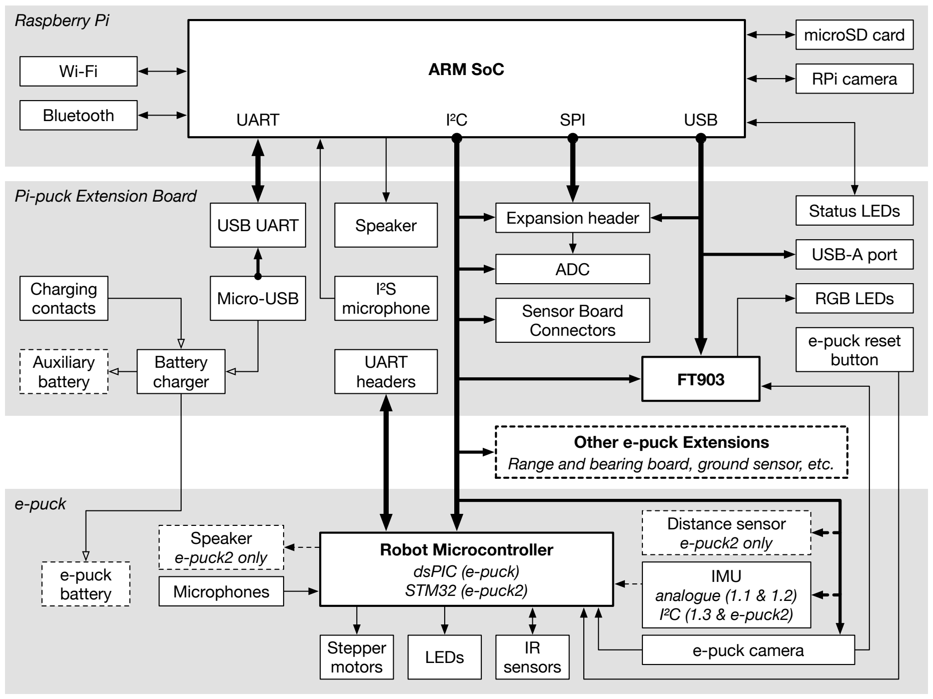

Fig. 2. Overview of interfaces between the Raspberry Pi, Pi-puck extension board, and e-puck hardware. Arrows show master to slave (I^2C/SPI), host to device (USB), or data/power direction. Dashed lines indicate hardware and connections that are optional or not available on all e-puck revisions.

VL53L1X ToF laser-ranging module (4 m range) – similar to the VL53L0X distance sensor on the front of the e-puck2 (with a 2 m range).

Battery Power: The Pi-puck extension board can be powered from either the standard e-puck battery, an auxiliary battery connected through a JST-PH socket, or from both batteries simultaneously. When powered from an 1800 mAh e-puck battery, an idling Pi-puck with no expansion hardware will drain its battery in approximately 5 hours. An active Pi-puck that is performing a simple obstacle avoidance algorithm controlled from the Raspberry Pi, while fitted with the extensions shown in Figure 1 (including an XBee, OLED display and six ToF sensors), has been measured to last around 1.5 hours. Both of these times could be increased significantly by attaching an auxiliary battery to augment the power provided by the e-puck battery.

The Pi-puck has two battery charging circuits on-board, to allow for charging each of the two batteries independently. Batteries can be charged either by connecting a 5 V power supply to the sprung charging contacts on the front of the robot (e.g. via an external charging wall), or through the integrated micro-USB

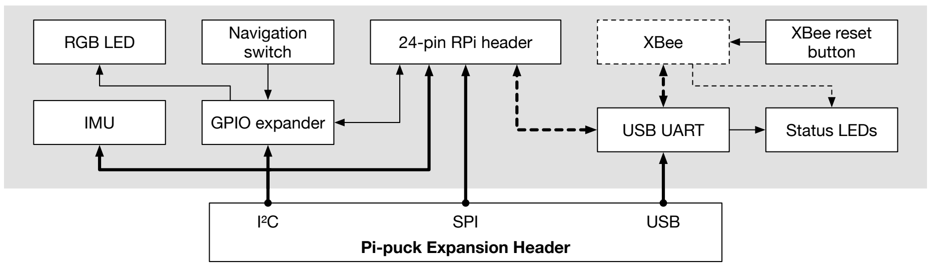

Fig. 3. Overview of interfaces between the Pi-puck and YRL Expansion Board. Arrows show master to slave (I^2C/SPI), host to device (USB), or data direction. Dashed lines indicate hardware and connections that are optional.

socket. Additionally, both battery voltages can be measured in real-time from the Raspberry Pi using an on-board analogue-to-digital converter (ADC), allowing automatic shutdown or recharging behaviours to be triggered in software.

Camera Interface: An FTDI FT903 MCU is used to convert the e-puck's parallel camera interface to a USB Video Device Class (UVC) peripheral to make it accessible from the standard Linux kernel and applications on the Raspberry Pi. Like the Xpuck [7], the Pi-puck can capture 640x480 resolution video at 15 frames per second, enabling improved on-board image processing over the resource-constrained e-puck microcontroller. The Raspberry Pi configures the camera sensor via I^2C, then the FT903 creates the UVC device and streams the image. The FT903's firmware can easily be updated over USB using the Device Firmware Upgrade (DFU) standard, and its UART interface is broken out to header pins, to allow for customisation of the microcontroller firmware and support for potential future e-puck camera sensor design changes.

Additional I/O: The Raspberry Pi is connected directly to an I^2S microphone on the Pi-puck extension board, as well as an audio amplifier and speaker. The Pi-puck board has three additional RGB LEDs that can be individually controlled over the I^2C interface, via the FT903 microcontroller. A vertical USB-A port allows the connection of an additional USB device to the Raspberry Pi, through the on-board USB hub, and a further USB channel is attached to a general-purpose expansion header, which also breaks out the I^2C and SPI buses along with power and other signals. Both e-puck1 and e-puck2 UART interfaces are broken-out to pins on the Pi-puck board, to allow for easier debugging of the robot's microcontroller firmware using modern 3.3 V signals.

2.1 YRL Expansion Board

To complement the base Pi-puck platform, we have developed an additional board that connects to the expansion header on the top of the robot. This board has a similar form-factor to the Raspberry Pi Zero, and is physically mounted to the Raspberry Pi while being electrically connected to the Pi-puck board (see Figure 1). An overview of the expansion board hardware is shown in Figure 3, including its I^2C, SPI and USB interfaces to the base Pi-puck, and additional I/O options. The hardware designs for the expansion board are fully open source[5], allowing it to be used as a basis for other custom expansion boards if desired.

The expansion board hosts an LSM9DS1 9-DOF IMU, which provides a 3-axis accelerometer, gyroscope, and magnetometer to the Raspberry Pi over an I^2C interface, and is an essential addition for certain robotics applications such as SLAM algorithms. This is useful primarily when using the e-puck1, which only has either a 3-DOF accelerometer connected to the dsPIC, or a 6-DOF I^2C accelerometer and gyroscope (depending on the specific hardware revision), compared to the e-puck2 which has a built-in MPU-9250 9-DOF IMU.

The expansion board also provides a socket for an XBee radio module, accessible through a USB-UART interface to the Raspberry Pi, and LEDs for showing the radio status. Using a generic set of headers for the XBee interface allows multiple generations and specifications of XBee modules to be used, as long as they comply to the standard pinout. The Pi-puck's XBee interface enables peer-to-peer, point-to-point, or mesh networking between robots, and can be used for transmitting data with higher bandwidth than infrared transceivers, as well as estimating the distance of neighbouring robots by measuring received signal strength (see Figure 4). In addition to robot-to-robot communication, the Pi-puck's XBee module could also be integrated with XBee-enabled experimental infrastructure like the IRIDIA Task Abstraction Module [1].

One benefit of the Pi-puck is the ability to leverage the Raspberry Pi ecosystem, which is taken further by the expansion board's 24-pin Raspberry Pi compatible header, allowing the robot to be extended with a large variety of existing hardware. Figure 1 shows the Pi-puck with the off-the-shelf Inky pHAT e-ink display from Pimoroni [18], which can be used in a swarm context for very low power dynamic agent identification through ArUco tags [3], and additional human-swarm interaction possibilities [14], as well as enabling simpler integration with tools like ARDebug [13]. This is achieved easily though using existing software packages with minimal modification, and without the need for custom Linux kernel drivers or firmware.

3 Software Ecosystem

This section describes the software ecosystem that supports the Pi-puck extension board hardware, including our customised Linux distribution, e-puck microcontroller firmware, and software interfaces to other e-puck extensions.

Linux Distribution: There are currently two main Linux distributions for the Pi-puck – one supported by GCtronic[6] (included on the Pi-puck's microSD card

[5] https://github.com/yorkrobotlab/pi-puck-expansion-board
[6] https://www.gctronic.com/doc/index.php?title=Pi-puck

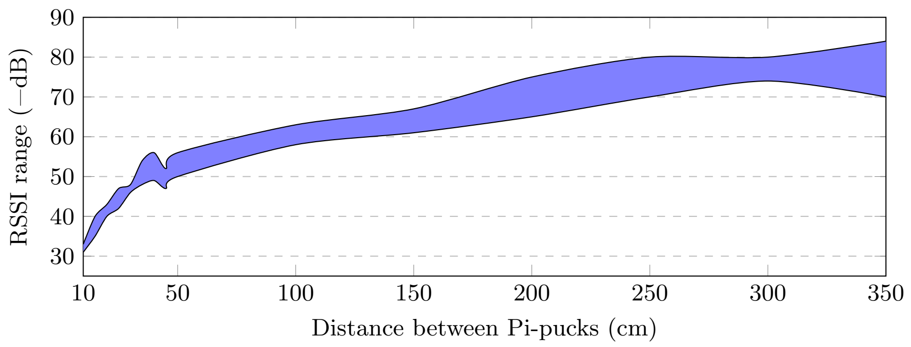

Fig. 4. Reported range of XBee packet RSSI (Received Signal Strength Indicator) values with varying transmission distance between a pair of Pi-pucks.

when purchased from them), and one created by YRL, which is detailed in this paper. This allow us to provide support for different features of the platform, while targeting different users and alternative approaches for packaging software.

The YRL Pi-puck software distribution offers a foundation for research and education, with a focus on open-source packages that are easy to modify, build and distribute. The core of the distribution is supplied as a set of Debian packages that are hosted in a package repository online[7], and are available in source format for modification if desired[8]. These packages cover the full Linux set-up of the Pi-puck hardware, as well as providing utilities for controlling and programming various devices on the robot. Distributing this software via Debian packages allows for easy installation on any Debian-based Linux distribution, automatic resolution of any dependencies, and straightforward updates.

To accelerate the initial configuration of each robot, the Pi-puck Debian packages are built into the Pi-puck Raspbian microSD card image[9], which is created using the `pi-gen` tool from the Raspberry Pi Foundation. This image is based on the standard Raspberry Pi Foundation Raspbian Buster image, but with additional build stages added to include the Pi-puck packages, and to modify the default Linux configuration to better support a swarm of robots. This image is supplied both in source form and as a file system image that can be directly copied onto a microSD card for use with the Raspberry Pi. Additional files for assisting with the deployment of a swarm of robots are included on the FAT32 `boot` partition of the SD card, allowing users to configure parameters such as Wi-Fi connection details and a unique robot hostname before the image is first booted. Users can also add additional packages and files into the `pi-gen` build system, in order to create a custom Raspbian distribution for a specific experiment or application.

[7] https://www.cs.york.ac.uk/pi-puck/
[8] https://github.com/yorkrobotlab/pi-puck-packages
[9] https://github.com/yorkrobotlab/pi-gen

e-puck Microcontroller Firmware: Alongside the Linux software for the Pi-puck, the e-puck1 dsPIC firmware has been re-written to support controlling the robot entirely over I^{2}C, and to update the code to work with modern Microchip XC16 compilers and the MPLAB X IDE. Firmware for the e-puck2 is currently provided and supported by GCtronic.

The Pi-puck hardware enables users to program the e-puck1 firmware HEX file directly from Linux running on the Raspberry Pi, allowing any changes to the firmware to be easily programmed onto a swarm of robots over an SSH connection. We provide a new dsPIC bootloader firmware that must be programmed to each e-puck once using a standard PIC in-circuit debugger to enable this feature, after which all subsequent programming can be done directly from the Raspberry Pi, using provided programming scripts. Due to hardware differences, the same method of firmware programming does not work with the e-puck2's microcontrollers, however they could be programmed from the Raspberry Pi using Bluetooth, Wi-Fi or USB (with the use of an additional cable) instead.

Other e-puck Extensions: The Pi-puck hardware allows the Raspberry Pi to communicate with any devices on the e-puck's I^{2}C bus, including other e-puck extension boards, such as the range and bearing turret [5] and ground sensors module. Python software has been written to demonstrate how to interface with the range and bearing turret directly from the Raspberry Pi (with both the standard and DEMIURGE firmware[10]), without requiring any input from the e-puck's microcontroller, and is provided as an example in the Pi-puck software repository. Python code for communicating with the e-puck ground sensors extension from the Raspberry Pi is also provided, allowing for this to be included in high-level robot control applications.

4 Robot Operating System (ROS) Support

The base e-puck1 is able to provide limited ROS support via Bluetooth and an external computer, and the e-puck2 extends this functionality with the option of using Wi-Fi instead of Bluetooth [4]. The Pi-puck allows ROS nodes to be executed directly on the robot instead, thanks to the Raspberry Pi's embedded Linux operating system. ROS integration for the Pi-puck is implemented by the `pi_puck_driver` package, which contains a series of Python nodes[11] that support the features listed in this section. The Pi-puck ROS repository has been created for ROS Melodic (supported until May 2023), and has been tested on Raspbian Buster on the Raspberry Pi.

Motors: The `motors` node communicates with a robot controller by subscribing to wheel speeds and publishing the step counts of the e-puck's stepper motors. This node interfaces with the e-puck's microcontroller via I^{2}C to request step

[10] https://github.com/demiurge-project/argos3-epuck/tree/master/erab_firmware
[11] https://github.com/yorkrobotlab/pi-puck-ros

counts and set the speed of each wheel independently, allowing for precise movement and turning. The `motors` node also publishes `nav_msgs/Odometry` messages, containing the robot's pose and linear/angular velocity, estimated from the motor step counts using dead reckoning.

A `base_controller` node is also provided to convert `geometry_msgs/Twist` messages (containing the desired linear and angular velocity of the robot) into control signals for the motors. This allows control of the robot to be abstracted away from specifying individual wheel speeds, and the implementation automatically scales the wheel speeds to account for combined linear and angular velocities exceeding physical limits.

Sensors: The `short_range_ir` node interfaces with the e-puck's analogue IR transceivers, and publishes their readings as eight separate `sensor_msgs/Range` messages (reflected IR, offset by ambient IR), so they can be used as proximity sensors. The raw IR readings are mapped to distances between 5 mm and 40 mm by applying a logarithmic least error fit to experimentally-measured sensor data. The `long_range_ir` node similarly interfaces with the Pi-puck's optional long-range ToF distance sensors (via STMicro's pre-built closed-source driver), and publishes their readings as up to six separate `sensor_msgs/Range` messages (depending on the number of sensor modules installed).

Many existing ROS SLAM packages (such as GMapping) require distance data to be presented as `sensor_msgs/LaserScan` messages rather than the point cloud that is obtained from the individual distance and ToF sensors. To solve this, a transform is provided that re-exposes a `sensor_msgs/Range` message as a `sensor_msgs/LaserScan` of three points, denoting the edges and centre of the field of view of the range sensor. Additionally, for sensor measurements to be mapped correctly onto the world, the reference frames for the sensors and the Pi-puck itself must be broadcast as a series of transforms, either statically or dynamically depending on whether the reference frame can move in relation to other reference frames. A Unified Robot Description Format (URDF) model is provided for this purpose, which contains a list of reference frames, and a list of static transforms between those reference frames. This also allows the Pi-puck sensor readings to be visualised on a 3D model in RViz.

Power, IMU, and OLED: The `power` node interfaces with the Pi-puck's ADC to obtain the voltages of the e-puck and auxiliary batteries, and publishes them, along with metadata such as whether the battery is currently charging, as `sensor_msgs/BatteryState` messages.

Additional support is provided for features of the YRL Expansion Board, such as the LSM9DS1 IMU and accessories using the 24-pin header. The `imu` node interfaces with the IMU on the expansion board, and publishes the accelerometer and gyroscope readings as `sensor_msgs/Imu` messages. Magnetometer readings are published as `sensor_msgs/MagneticField` messages, and the robot's pose is subsequently derived from these two message types using Madgwick's IMU and AHRS (Attitude and Heading Reference System) algorithm [10].

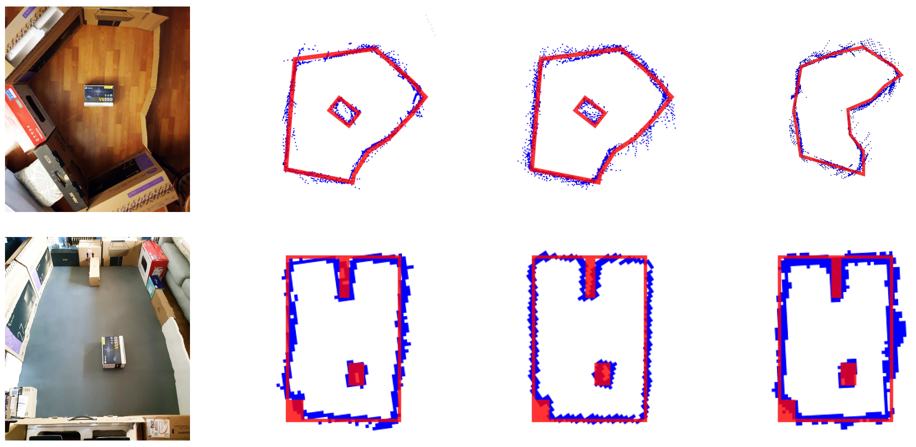

Fig. 5. Results of single-robot and multi-robot mapping experiments. *From left to right*: test arenas, single-robot mapping with obstacle avoidance, single-robot mapping with frontier exploration, and swarm mapping.

Calibration scripts are also provided in order to account for magnetic interference from the robot's motors and speaker. The `oled` node interfaces with the optional Adafruit PiOLED display, and subscribes to `std_msgs/String` and `sensor_msgs/Image` messages published by other nodes, allowing text and images to be displayed (e.g. robot ID, or status).

5 Environment Mapping

As an initial application case study, we present experimental results from an environment mapping task to demonstrate the applicability of the Pi-puck ROS platform for swarm experimentation. The focus here is on the integration of the Pi-puck drivers with existing third-party ROS packages, to highlight the compatibility and ease of use of the platform. The currently available experimental environments are quite small, limiting the maximum swarm size due to practical working limitations, but experimentation with larger environments and more robots is planned once possible.

Single-robot Mapping: Single-robot SLAM has previously been implemented by GCtronic, both using a real e-puck and an e-puck model simulated in Webots [11]. This was achieved via the e-puck's Bluetooth ROS driver, the Webots ROS interface, and the OpenSLAM GMapping package. However, this approach used the e-puck's analogue IR sensors, so the mapping range was limited to around 40 mm. Longer-range mapping has since been achieved through the use of ToF sensor modules via an Arduino interface and custom hardware [16].

To test the compatibility of the Pi-puck ROS drivers with other ROS packages, we integrated them with the GMapping package [23] (running on a separate computer) and the default ROS navigation stack. We used a single Pi-puck to map the two arenas shown in Figure 5, controlled both via an obstacle avoidance controller, and with the `explore_lite` frontier exploration package [6]. Both of these environments were successfully mapped, as shown in Figure 5.

Multi-robot Mapping: Kegeleirs et al. [8] recently investigated the effect of different random walk behaviours on an e-puck swarm's ability to map simple environments. Their work was implemented using ROS Indigo support for the e-puck's Gumstix Overo COM turret and the GMapping package, along with the `multirobot_map_merge` package [6] for combining the maps produced by each robot. Results from their initial experiments in the ARGoS robot simulator [20] were quite promising, but unfortunately the maps produced by the real e-puck robots were far less faithful to the true environment. This can be attributed to the limited range and high noise of the base e-puck's IR sensors, as well as compound odometry errors.

To test the ability of the Pi-puck ROS drivers to work in an environment where multiple robots are operating together on the same ROS network, a small swarm of four robots was used to perform mapping (without localisation) using GMapping while avoiding obstacles. The four individual maps were then combined in real-time using the `multirobot_map_merge` package. The Pi-puck swarm was able to successfully map the environments (as shown in Figure 5), thanks to the improved IMU on the expansion board and the longer-range, higher-accuracy ToF sensor modules. Initial robot positions were randomised, not known to the swarm, and were not coordinated in any way.

6 Conclusion

The Pi-puck is an open-source extension for the e-puck and e-puck2 robot platforms that expands their capabilities by interfacing with the Raspberry Pi – a popular single-board computer. This paper has detailed the latest hardware revision of the Pi-puck, as well as the software infrastructure developed to support it, including Raspbian and ROS integration. This affords access to the Debian and ROS ecosystems, allowing for easy use of standard algorithms for tasks such as navigation and SLAM.

We hope that the hardware presented in this paper will facilitate experimentation with swarm algorithms that were previously either not possible or inconvenient to implement, and that the evolving software infrastructure continues to support the efforts of other researchers. Full documentation and source code for the Pi-puck platform and associated extensions is available online[12], in addition to the resources on the GCtronic Wiki [4].

[12] https://pi-puck.readthedocs.io

References

1. Brutschy, A., Garattoni, L., Brambilla, M., Francesca, G., Pini, G., Dorigo, M., Birattari, M.: The TAM: abstracting complex tasks in swarm robotics research. Swarm Intelligence **9**(1), 1–22 (2015)
2. Garattoni, L., Francesca, G., Brutschy, A., Pinciroli, C., Birattari, M.: Software infrastructure for e-puck (and TAM). Université Libre de Bruxelles, Tech. Rep. TR/IRIDIA/2015-004 (2015)
3. Garrido-Jurado, S., Muñoz Salinas, R., Madrid-Cuevas, F., Medina-Carnicer, R.: Generation of fiducial marker dictionaries using mixed integer linear programming. Pattern Recognition **51** (10 2015)
4. GCtronic: GCtronic Wiki, https://www.gctronic.com/doc/index.php?title=GCtronic_Wiki
5. Gutiérrez, Á., Campo, A., Dorigo, M., Donate, J., Monasterio-Huelin, F., Magdalena, L.: Open e-puck range & bearing miniaturized board for local communication in swarm robotics. In: International Conference on Robotics and Automation. pp. 3111–3116. IEEE (2009)
6. Hörner, J.: Map-merging for multi-robot system. Bachelor's thesis, Charles University in Prague, Faculty of Mathematics and Physics, Prague (2016)
7. Jones, S., Studley, M., Hauert, S., Winfield, A.F.T.: A Two Teraflop Swarm. Frontiers in Robotics and AI: Multi-Robot Systems **5**(11), 1–19 (2018)
8. Kegeleirs, M., Ramos, D.G., Birattari, M.: Random walk exploration for swarm mapping. In: Annual Conference Towards Autonomous Robotic Systems. pp. 211–222. Springer (2019)
9. Liu, W., Winfield, A.F.T.: Open-hardware e-puck Linux extension board for experimental swarm robotics research. Microprocessors and Microsystems **35**(1), 60–67 (2011)
10. Madgwick, S.O.H., Harrison, A.J.L., Vaidyanathan, R.: Estimation of IMU and MARG orientation using a gradient descent algorithm. In: IEEE International Conference on Rehabilitation Robotics (2011)
11. Michel, O.: Cyberbotics Ltd. Webots: professional mobile robot simulation. International Journal of Advanced Robotic Systems **1**(1), 5 (2004)
12. Millard, A.G., Joyce, R., Hilder, J.A., Fleşeriu, C., Newbrook, L., Li, W., McDaid, L.J., Halliday, D.M.: The Pi-puck extension board: a Raspberry Pi interface for the e-puck robot platform. In: International Conference on Intelligent Robots and Systems (IROS). pp. 741–748. IEEE (2017)
13. Millard, A.G., Redpath, R., Jewers, A.M., Arndt, C., Joyce, R., Hilder, J.A., McDaid, L.J., Halliday, D.M.: ARDebug: an augmented reality tool for analysing and debugging swarm robotic systems. Frontiers in Robotics and AI: Multi-Robot Systems **5**(87), 1–6 (2018)
14. Millard, A.G., Joyce, R., Gray, I.: Human-swarm interaction via e-ink displays. In: ICRA Human-Swarm Interaction Workshop (2020)
15. Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.C., Floreano, D., Martinoli, A.: The e-puck, a robot designed for education in engineering. In: Conference on Autonomous Robot Systems and Competitions. vol. 1, pp. 59–65 (2009)
16. Moriarty, D.: Swarm Robotics — Mapping Using E-Pucks: Part II, https://medium.com/@DanielMoriarty/swarm-robotics-mapping-using-e-pucks-part-ii-ac15c5d62e3

17. Nedjah, N., Junior, L.S.: Review of methodologies and tasks in swarm robotics towards standardization. Swarm and Evolutionary Computation **50**, 100565 (2019)
18. Pimoroni: Inky pHAT EPD Display for Raspberry Pi. https://shop.pimoroni.com/products/inky-phat
19. Pinciroli, C., Beltrame, G.: Buzz: An extensible programming language for heterogeneous swarm robotics. In: International Conference on Intelligent Robots and Systems (IROS). pp. 3794–3800. IEEE (2016)
20. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., et al.: ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm intelligence **6**(4), 271–295 (2012)
21. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. In: ICRA workshop on open source software (2009)
22. Rezeck, P.A., Azpurua, H., Chaimowicz, L.: HeRo: An open platform for robotics research and education. In: Latin American Robotics Symposium (LARS) and Brazilian Symposium on Robotics (SBR). pp. 1–6. IEEE (2017)
23. ROS Contributors: gmapping - ROS Wiki, http://wiki.ros.org/gmapping
24. Trenkwalder, S.M., Lopes, Y.K., Kolling, A., Christensen, A.L., Prodan, R., Groß, R.: OpenSwarm: An event-driven embedded operating system for miniature robots. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 4483–4490. IEEE (2016)
25. West, A., Arvin, F., Martin, H., Watson, S., Lennox, B.: ROS integration for miniature mobile robots. In: Annual Conference Towards Autonomous Robotic Systems. pp. 345–356. Springer (2018)
26. Zhang, Y., Zhang, L., Wang, H., Bustamante, F.E., Rubenstein, M.: SwarmTalk – Towards Benchmark Software Suites for Swarm Robotics Platforms. In: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems. pp. 1638–1646 (2020)