Modular Development of Mobile Robots with Open Source Hardware and Software Components

Martino Migliavacca, Andrea Bonarini, and Matteo Matteucci

Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria,
Piazza Leonardo Da Vinci 32, 20133, Milano, Italy
{migliavacca,bonarini,matteucci}@polimi.it

Abstract. Prototyping and engineering robot hardware and low-level control often require time and efforts thus subtracted to core research activities, such as SLAM or planning algorithms development, which need a working, reliable, platform to be evaluated in a real world scenario. In this paper, we present Rapid Robot Prototyping (*R2P*), an open source, hardware and software architecture for the rapid prototyping of robotic applications, where off-the-shelf embedded modules (e.g., sensors, actuators, and controllers) are combined together in a plug-and-play fashion, enabling the implementation of a complex system in a simple and modular way. R2P makes people involved in robotics, from researchers and designers to students and hobbyists, dramatically reduce the time and efforts required to build a robot prototype.

1 Introduction

In recent years, several development frameworks [6, 4, 11, 8] have been proposed to assist researchers in the design of robotic applications. While these projects really boosted the development of high-level software, hardware design and low-level firmware development are still critical tasks. To develop a new mobile robot, designers always face the problem of selecting hardware devices, controlling them, and interfacing them with the high-level software. This slows down the progress of robotic research, as prototyping and engineering often requires more time and resources than tasks strictly related to the target application.

To simplify the development of new robotic applications, we developed Rapid Robot Prototyping (*R2P*) [2, 1], an open source hardware and software framework focused on speeding up the prototyping of robotic systems. R2P provides hardware modules that implement basic functionalities needed by common robotic applications, and a lightweight, real-time, middleware to easily write low-level control software. R2P targets span from mobile autonomous robots used for research purposes to entertainment and service applications, such as games, telepresence, and rescue. The limits of R2P, at the actual stage of development, are only imposed by the modules already available; moreover, as R2P is an open source, modular, framework, it can be extended by users with additional modules to cover other application fields.

2 Modular Hardware and Software Development

When a new robotic application is investigated, the first steps involve selecting the hardware devices, e.g., sensor and actuators, and building the platform needed to validate the overall idea. Looking at today's possibilities, we can pick devices either from the automation market or from the hobby market. Components from automation market are often expensive and offer overkilling performance with respect to the requirements of a robotic application prototype. Moreover, automation devices often require power supplies not suitable for battery powered systems like mobile robots. On the other hand, devices from hobby market are usually cheap, but they show poor performance, low reliability, and no real-time capabilities making impossible any distributed control loop. Having selected hardware devices, here it comes the problem of interfacing them with each other, and with the high-level control software. Different manufacturers generally use different data links and protocols, increasing wiring complexity and requiring specialized device drivers. As a consequence, resulting platforms are commonly based on custom setups, which are hardly reusable in different projects. Although mobile robots have been built for decades by integrating heterogeneous devices, or implementing custom solutions, we firmly believe that a modular approach based on off-the-shelf components would strongly help robot designers in developing new applications. To the best of our knowledge, the only available modular robotic platforms, such as the E-puck educational robot [10], the Kephera robot [7], and a few others, are aimed at developing small mobile robots for applications like swarm robotics and their usage is restricted to control the platform they are designed for.

With R2P, we aim at fulfilling the lack of hardware components focused on robot prototyping, pushing design strategies commonly exploited in software development – such as modular, component-based, software engineering – down to the hardware level. R2P relies on the principle that the requirements of a generic robot application can be implemented by modules not only at software level, as it is common in most frameworks, but also at hardware level. Basic functionalities such as motor control, distance measurement, inertial navigation are implemented by specific, standardized hardware modules, with corresponding firmware, that can be plugged on a common bus and can interact in real-time. Firmware development tools, and a middleware to foster distributed, reusable, software development, are provided, supporting users in writing code on resource-constrained devices. Using R2P, robot designers can build generic platforms by choosing the modules they need, configuring them, and easily developing the control software, implementing complex systems in a plug-and-play fashion. Integration with high-level robotics frameworks, such as ROS [11], is provided by a gateway module.

3 R2P: the Rapid Robot Prototyping Framework

In this section, we introduce R2P design choices and architecture. Then, a review of some of the already available hardware modules are presented.

3.1 Power and Data Link

R2P uses a single connector to transport both power and data. Power consumption is limited to $5V$, $200mA$, for each module, which suites the requirements of most electronic devices, while modules needing higher power, such as motor drivers, must rely on auxiliary connections. Modules exchange data using the CAN-Bus, which has been designed to work in harsh environments and is available on many microcontrollers. Its maximum data rate of $1Mbps$ is generally enough for a distributed system of smart devices, where only high level information needs to be sent over the network (i.e., no raw sensor data is exchanged), thus needing a relatively small bandwidth [3]. As part of R2P, we developed RTCAN [9], a CAN-Bus protocol targeted at robotic applications that supports both sporadic, event-triggered, and periodic, time-triggered communication, with soft and hard real-time constraints. To reduce wires, a daisy chain wiring schema is adopted: each module has two ports to connect to the previous and the next component, as shown in Figure 2(a). This also supports easy connection of new modules to an existing system.

3.2 Embedded firmware development

Writing code for resource-constrained devices, such as microcontrollers used to interface with sensors and actuators, requires specific knowledge and competence. Most robot designers are used to write software on desktop-level computer systems, and they have to spend time and efforts to start developing code targeted to embedded devices. To reduce this effort, the use of an operating system can significantly support software development even for small embedded systems as it features threads, memory management, message passing primitives, and other services programmers are commonly used to deal with. Moreover, an operating system with real-time capabilities is important to manage critical, high-priority tasks, which are often involved in robotic systems, e.g., for closed-loop control. For the mentioned reasons, R2P relies on ChibiOS/RT [12], a real-time operating system designed for deeply embedded real time applications. ChibiOS/RT has been preferred to other alternatives for its portability, ease of use, rich features set, and extremely high efficiency; anyway, a review of available embedded operating systems is out of the scope of this paper. ChibiOS/RT also includes a Hardware Abstraction Layer (*HAL*), which abstracts the hardware implementation of different low level peripherals, relieving the developer from acquiring specific competence on each specific platform and making easier the port of existing code to different targets.

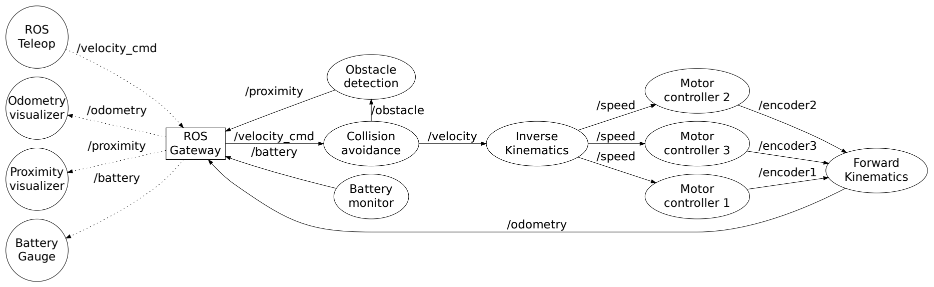


Fig. 1. The distributed architecture of the embedded software controlling Triskar2.

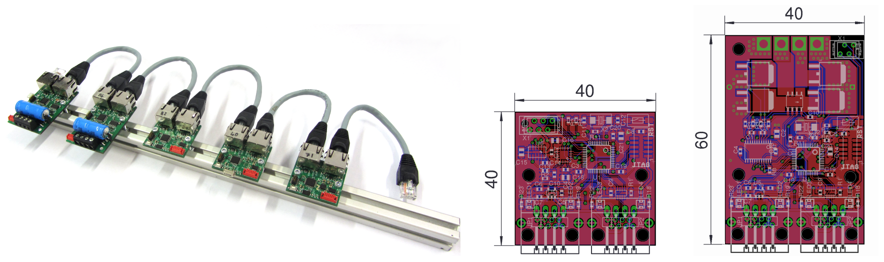
3.3 Publish/Subscribe Middleware

To support the development of modular software components on embedded target, R2P features a lightweight communication middleware. R2P middleware main goals are software reuse, real-time communication, efficient implementation, and ease of use. It follows the *publish/subscribe* paradigm [5]: data producers *publish* messages on a *topic*, i.e., a communication channel, while data consumers *subscribe* to the corresponding topic to receive messages. Identifying data by its content, i.e., the topic it is published on, instead of by its producer, also promotes loosely-coupled software design and, thus, code reuse. The middleware provides concepts common to most robotics frameworks used on computer systems, such as software nodes, topics, publishers, subscribers, and message queues.

R2P middleware is written in a subset of C++, to take advantage of some object-oriented programming features without compromising performance on embedded targets. Its implementation is focused on code efficiency and messaging performance. Software nodes can subscribe to both local and remote publishers, with no difference from the user point of view. The middleware supports both periodic and sporadic publishers, which can specify real-time communication constraints: update period for time-triggered messages, and delivery deadline for event-triggered ones. Finally, a simple API, which reminds the ROS syntax, enables developers to write embedded, distributed code as they are used to do on computer systems, fostering code reuse through different projects.

3.4 Integration with ROS

While R2P supports rapid development of robotic systems using off-the-shelf hardware and software components, applications involving computation-intensive tasks such as computer vision, localization, and complex planning, must also rely on a computer system and, eventually, a software framework. Among the many available development frameworks for robotics software, ROS [11] is currently the most widely adopted in academia and research laboratories, and, recently, it



(a) Daisy-chain wiring of hardware modules (b) R2P IMU (c) R2P DC motor controller

Fig. 2. R2P hardware modules.

has been considered also by industrial developers. To natively integrate resource-constrained devices within ROS, we developed *μROSnode*, a lightweight, open source, ANSI C ROS client library. R2P provides a gateway module (see Section 3.5), which acts as a proxy between the R2P middleware and ROS systems. Topics published on the R2P network can be accessed from ROS nodes, and, at the same way, R2P modules can subscribe data published by ROS software.

3.5 Off-the-Shelf Hardware Components

We have designed and built, as part of the R2P framework, a set of plug-and-play hardware modules that implement basic functionalities required by common robotics applications. Modules are based on STM32 Cortex-M3 microcontrollers with $20Kb$ of RAM and $128Kb$ of Flash memory, running the ChibiOS/RT and the R2P middleware. Each module has two RJ45 ports for daisy-chain connection to the bus, a serial port to download new firmware and for debugging purposes, and a JTAG header for advanced users who want to directly access the microcontroller. An overview of the currently available modules follows.

PSU Module. This is the power supply unit, which powers all the modules connected to the bus. Input voltage range is from $5.5V$ to $36V$ DC. A DC-DC converter produces a $5V$ regulated output with maximum current supply of $4A$ and short circuit protection. Both battery voltage and current drain can be published over the network to monitor power consumption and to estimate the residual battery life.

DC Motor Module. This high-power motor controller board can drive DC motors up to $36V$, delivering a continuous $20A$ current. It features closed loop control, with position feedback from a quadrature encoder and current measurement from the on-board Hall-effect sensor. The DC motor module accepts position, speed, and torque set points, and can publish position and speed messages, exploiting data from the encoder, and the measured current drawn.

IMU Module. A 10-DoF Inertial Measurement Unit featuring MEMS accelerometer, gyroscope, magnetometer and pressure sensor. An additional serial port to acquire GPS coordinates from an external GPS receiver is also provided on this module. The on-board sensor fusion algorithm produces heading, attitude, and position messages.

Proximity Module. A module to interface with proximity sensors such as the Sharp IR rangers or *MaxBotix* ultrasonic sensors. Each module connects to up to 4 sensors. Calibration and data filtering algorithms run on the microcontroller, which produces distance measurements.

Gateway Module. This is the gateway module mentioned in Section 3.4. It features an Ethernet port and a more powerful, Ethernet-enabled, microcontroller to handle the TCP/IP stack. R2P messages can be forwarded from the CAN-Bus to the IP network, and the other way around. The gateway module runs µROSnode, which enables a direct integration of R2P modules with ROS systems.

3.6 Open Source Development

R2P is fully open source, both hardware and software, to encourage its adoption and to take advantage of community-driven improvements to became a mature and widespread project. The design of the boards, the code they run and the middleware are available on the R2P repository: http://github.com/openrobots-dev. At the moment of writing, R2P has reached its maturity (see, e.g., the use case in the next section), but its development is still actively progressing, thus, the repository is frequently updated.

4 Use case: an Omnidirectional Robot

We used R2P to develop the omnidirectional wheeled robot *Triskar2*, shown in Figure 3(a). The robot sports 3 R2P DC modules, a PSU module, a Proximity module, and the Gateway module to interface with a computer running ROS. The low-level control software embedded on the modules, which exploits the R2P publish/subscriber middleware, is reported in Figure 3. Software components are enclosed in R2P nodes, which implement basic functionalities, performing a specific task. Then, nodes are composed as a distributed architecture, implementing a complex system from basic, reusable, components. This design strategy is not innovative, being commonly used in software development; the main contribution of R2P middleware is to bring the same approach, and, thus, the same advantages, to embedded firmware development, with the same programming interfaces known to most robot developers.

Software nodes have been deployed on the modules as shown in Figure 3(b). Some nodes have to run on specific boards (e.g., those that are directly connected to the hardware like motor controller nodes), while others can run on any connected module. For example, in our tests, the inverse kinematics model to compute wheel speeds was run on the *Motor 1* module, while the odometry node was deployed on *Motor 2*. In this way, we can balance processor load and reduce latency, easily moving nodes from an hardware module to another.

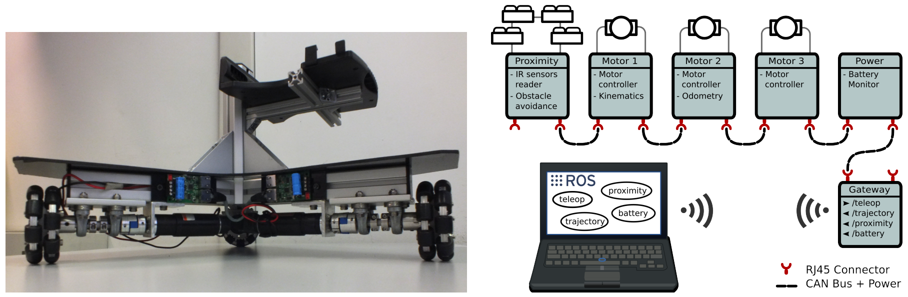


Fig. 3. The Triskar2 omnidirectional platform (a) and the R2P hardware modules controlling the robot (b).

Thanks to the R2P gateway, Triskar2 can be controlled by any ROS application publishing native ROS topics. We firstly teleoperated the robot by using standard ROS *teleop* messages, then we developed a robotic game, involving the Triskar2 robot and a quadricopter, both controlled by ROS software.

5 Conclusions

In this paper, we presented R2P, an open source hardware and software framework for the rapid prototyping of robots. Bringing design strategies such as modular development, and components reuse, down to hardware level, R2P enables robot designers to build and control a robotic platform using off-the-shelf modules. Exploiting the R2P framework, generic mobile robots can be built bottom-up in a distributed plug-and-play fashion by simply selecting the hardware modules to satisfy the needed functional requirements and easily programming their interaction. Integration with high-level software frameworks, e.g., ROS, allows to develop complex application, while low-level control is implemented by means of a modular distributed architecture, with real-time performance, without the need for advanced domain-specific knowledge. We are exploiting R2P to design new robots in our laboratory, as shown by the use case presented in Section 4, and to upgrade our previous platforms, the first being a balancing wheeled robot, a differential drive heavy-duty robot, and an autonomous wheelchair. The open source license encourages robot designers to adopt existing R2P modules to control their platforms, and to develop new hardware modules and software components that implement new functionalities.

Acknowledgements

This work has been partially supported by the research grant "Robotics for the Masses" from ST Microelectronics and Regione Lombardia, and by the Italian Ministry of University and Research (MIUR) through the PRIN 2009 grant "ROAMFREE: Robust Odometry Applying Multi-sensor Fusion to Reduce Estimation Errors".

Bibliography

[1] A. Bonarini, M. Matteucci, M. Migliavacca, and D. Rizzi. R2P: An open source hardware and software modular approach to robot prototyping. *Robotics and Autonomous Systems.*

[2] A. Bonarini, M. Matteucci, M. Migliavacca, and D. Rizzi. R2P: an open source modular architecture for rapid prototyping of robotics applications. In *Proceedings of 1st IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control (CESCIT'12)*, 2012.

[3] A. Bonarini, M. Matteucci, M. Migliavacca, R. Sannino, and D. Caltabiano. Modular low-cost robotics: What communication infrastructure? In *In proceedings of 18th World Congress of the International Federation of Automatic Control (IFAC)*, pages 917–922, 2011.

[4] H. Bruyninckx. Open robot control software: the OROCOS project. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, pages 2523–2528, 2001.

[5] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

[6] B. P. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *In Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, 2003.

[7] R. M. Harlan, D. B. Levine, and S. McClarigan. The khepera robot and the krobot class: a platform for introducing robotics in the undergraduate curriculum. In *ACM SIGCSE Bulletin*, volume 33, pages 105–109. ACM, 2001.

[8] A. Huang, E. Olson, and D. Moore. LCM: Lightweight communications and marshalling. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4057–4062, 2010.

[9] M. Migliavacca, A. Bonarini, and M. Matteucci. RTCAN: a real-time CAN-Bus protocol for robotic applications. In *Informatics in Control, Automation and Robotics (ICINCO), 2013 International Conference on*, 2013.

[10] 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*, volume 1, pages 59–65, 2009.

[11] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[12] G. D. Sirio. ChibiOS/RT real time operating system. http://www.chibios.org.