Integrating Arduino-based Educational Mobile Robots in ROS

Conference Paper · April 2013

DOI: 10.1109/Robotica.2013.6623520

4 authors, including:

André Araújo
University of Coimbra
10 PUBLICATIONS **111** CITATIONS
SEE PROFILE

Micael S. Couceiro
University of Coimbra
232 PUBLICATIONS **1,716** CITATIONS
SEE PROFILE

Rui P. Rocha
University of Coimbra
91 PUBLICATIONS **1,135** CITATIONS
SEE PROFILE

Integrating Arduino-Based Educational Mobile Robots in ROS

André Araújo · David Portugal · Micael S. Couceiro · Rui P. Rocha

Received: 5 July 2013 / Accepted: 3 December 2013

Abstract This article presents the full integration of compact educational mobile robotic platforms built around an Arduino controller board in the Robot Operating System (ROS). To decrease the development time, a driver interface in ROS was created so as to provide hardware abstraction and intuitive operation mode, thus allowing researchers to focus essentially in their main research motivation, e.g., search and rescue, multi-robot surveillance or swarm robotics. Moreover, the full integration in ROS provided by the driver enables the use of several tools for data analysis, easiness of interaction between multiple robots, use of different sensors and teleoperation devices, thereby targeting engineering education. To validate the approach, diverse experimental tests were conducted using different Arduino-based robotic platforms.

Keywords Arduino-based robots · ROS · Educational robotics · Sensing

Electronic supplementary material The online version of this article (doi:10.1007/s10846-013-0007-4) contains supplementary material, which is available to authorized users.

A. Araújo · D. Portugal (✉) · M. S. Couceiro · R. P. Rocha
Institute of Systems and Robotics, University of Coimbra, Pólo II, 3030-290 Coimbra, Portugal
e-mail: davidbsp@isr.uc.pt

A. Araújo
e-mail: aaraujo@isr.uc.pt

M. S. Couceiro
e-mail: micaelcouceiro@isr.uc.pt

R. P. Rocha
e-mail: rprocha@isr.uc.pt

1 Introduction

Mobile robotics is a technological field and a research area which has witnessed incredible advances for the last decades [1]. It finds application in areas like automatic cleaning, agriculture, support to medical services, hazard environments, space exploration, military, intelligent transportation, social robotics, and entertainment [2]. In robotics research, the need for practical integration tools to implement valuable scientific contributions is felt frequently. However, roboticists end up spending an excessive amount of time with engineering solutions for their particular hardware setup, often "reinventing the wheel". For that purpose, several different mobile robotic platforms have emerged with the ability to support research work focusing on applications like search and rescue, security applications, human interaction or robotics soccer and, nowadays, almost every major engineering institute has one or more laboratories focusing on mobile robotics research.

Earlier, the focus of research was especially on large and medium systems. However, with recent advances in sensor miniaturization and the increasing computational power and capability of microcontrollers in the past years, the emphasis has been put on the development of smaller and lower cost robots. Such low-cost platforms make affordable the experimentation with a larger number of robots (e.g., in cooperative robotics [3] and swarm robotics [4] tasks) and are also ideal for educational purposes. With such assumptions in mind, our group has been doing engineering and research work with two Arduino-based mobile platforms [5]: the TraxBot [6] and the StingBot.[1] The choice fell upon Arduino solutions, since it presents an easy-to-learn programming language (derived from C++) that incorporates various complex programming functions into simple commands that are much easier for students to learn. Moreover, the simplicity of the Arduino to create, modify and improve projects, as well as its open-source and reduced cost makes it among the most used microcontroller solutions in the educational context [5].

Following this trend of research, this article focuses on educational, open-source platforms that enable researchers, students and robot enthusiasts to quickly perform real world experimentation, having access to the tools provided by the Robot Operating System (ROS) [7]. ROS is currently the most trending and popular robotic framework in the world, reaching critical mass and being the closest one to become the *de facto* standard that the robotics community urgently needed.

With the exponential growth of robotics, some difficulties have been found in terms of writing software for robots. Different types of robots can have wildly varying hardware, making code reuse nontrivial. Opposing this tendency, ROS provides libraries and tools to help software developers to create robot applications. ROS features hardware abstraction in low-level device control, implementation of commonly-used functionally, message-passing between processes and package management. One of its gold marks is the amount of tools available for the community like the Stage simulator [8], navigation capabilities,[2] visual SLAM [9] and 3D point cloud based object recognition [10], among others. Regular updates enable the users to obtain, build, write and run ROS code across multiple computers.

In the next section, general purpose and educational mobile robots are reviewed, with a focus on those already integrated in ROS and on our Arduino-based robot platforms. In Section 3, the main contributions of this article are revealed and details on the development of the ROS driver and its features are presented. In the subsequent section, results with physical Arduino-based robots and a team of mixed real and virtual cooperating robots are presented and discussed, to demonstrate the integration of Arduino-based robots in ROS, as well as the easiness of extending the platforms with new sensors due to the ROS driver. Finally, the article ends with conclusions and future work.

2 Related Work

The following requirements, sorted by relevance, can be expected from robots to be used for educational purposes [11, 12]:

- Cost—Robots should be as cheap as possible to overcome budget limitations and evaluate multi-robot applications (e.g., swarm robotics);
- Energy Autonomy—Robots should have a long battery life since they may have to operate long enough during development and experimentation;
- Communication—Robots need to support wireless communication to increase the range of applications (e.g., multi-robot systems and networked robotics);
- Sensory System—Robots should be equipped with some form of sensing capability to allow interaction between them and with their environment;
- Processing—Robots need to be able to process information about other robots and the environment (e.g., sensing data).

The following subsection reviews popular educational and research platforms available in the market, after which we present the Arduino-based educational platforms developed and evaluate them according to the requirements presented above.

[1] http://www.isr.uc.pt/~aaraujo/doc

[2] http://www.ros.org/wiki/navigation

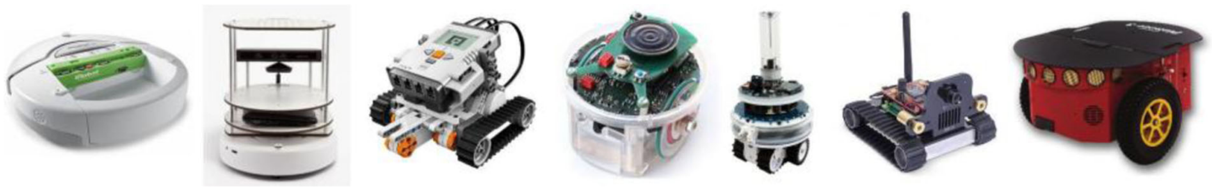


Fig. 1 Some well-known educational and research mobile robotic platforms: from left to right, iRobot Create, Turtlebot, Mindstorm NXT, e-puck, MarXbot, SRV-1 Blackfin and Pioneer 3-DX, respectively

2.1 Educational Robotic Platforms

Several off-the-shelf mobile robots with various sensors and diverse capabilities are illustrated in Fig. 1. We address their mobility within different ground environments, capabilities, size, sensing/perception, processing power, autonomous navigation and integration in ROS.

The iRobot Create [13] was designed for students and researchers, being very popular in the robotics community due to its small size and low cost. It is a circular platform based on the well-known vacuum cleaner Roomba, with extra space for larger sensors (e.g., 2D laser range finder or Kinect). Many choose to utilize an external computer that supports serial communication to control the Create robot, due to troublesome limitations in storage space and processing power. A ROS driver for the Roomba iCreate has already been developed (*irobot_create_2_1* package[3] in the *brown_drivers* stack),[4] as well as the original vacuum cleaning Roomba (*roomba_robot* stack).[5] Among the low cost platforms, the iRobot Create is one of the most typical robots used in laboratory context. As a consequence, there is a large number of known frameworks supporting it, such as ROS and Player/Stage [8]. Besides those, the emss iRobot Create Framework[6] not only provides an interface with the iRobot Create hardware, but also a completely emulated interface which mimics the hardware in every type of sensing information.

In fact, a popular off-the-shelf robot, developed at Willow Garage, has been built upon an iRobot Create: the TurtleBot.[7] This is a modular development platform incorporating an Xbox Kinect and an ASUS eeePC 1215N netbook. TurtleBot provides 3D functionalities and ROS out-of-the-box (through the *turtlebot* stack),[8] being fully open source and exploring all combined capabilities of its components.

The Mindstorms NXT [14] from Lego is an educational, academic robot kit, ideal for beginners. The robot is equipped with drive motors, encoders and a good variety of cheap sensors, like an accelerometer, light, sound, ultrasound and touch sensors. Support for interfacing and controlling this robot with ROS is also available, through the *nxt* stack.[9] Besides ROS, and being a commercial platform, there is a wide range of frameworks that either fully or partially support the Lego NXT, namely NXT-G,[10] Microsoft Robotics Developer Studio,[11] and Carnegie Mellon Robotic's Academy ROBOTC.[12]

The e-puck [15] is an educational swarm platform for beginners. It has tiny dimensions with only 80 mm of diameter, equipped with a vast set of sensors, like microphones, infrared sensors, 3D accelerometer and a VGA camera. Similarly, the MarXbot [16] platform has 170 mm of diameter, being fully equipped with infrared range sensors, 3D accelerometer, gyroscope, and an omnidirectional camera. It has a good processing power with an ARM 11 processor at 533 MHz. Both the e-puck and the MarXbot are programmed in a user-friendly scripting language, which uses ASEBA, an event-based low level control architecture. In order to interface it with ROS, a ROS/ASEBA Bridge has

[3]http://www.ros.org/wiki/irobot_create_2_1

[4]http://www.ros.org/wiki/brown_drivers

[5]http://www.ros.org/wiki/roomba_robot

[6]http://emssframework.sourceforge.net

[7]http://www.willowgarage.com/turtlebot

[8]http://www.ros.org/wiki/Robots/TurtleBot

[9]http://www.ros.org/wiki/Robots/NXT

[10]http://www.legoengineering.com/program/nxt-g

[11]http://www.microsoft.com/robotics

[12]http://www.robotc.net

been released (*ethzasl_aseba* stack).[13] Despite being supported in ROS, and more recently in the Virtual Experimentation Platform Coppelia Robotics (V-REP),[14] e-puck's most compatible framework is still Cyberbotics Webots.[15]

Additionally, the SRV-1 Blackfin [17] from Surveyor is a small-sized robot equipped with tracks. This robot has a good processing power with a 1000 MIPS at 500 MHz CPU, capable of running Linux Kernel 2.6. It is equipped with two IR rangers or optional ultrasonic ranging and a 1.3 MP camera. It also supports Wireless 802.11 b/g communication and various I2C sensors. Unlike the previous platforms, SRV-1 Blackfin can be driven in rough terrains due to its tracking system. At the time of writing, only partial support for ROS is available through the *ros-surveyor*[16] stack, which offers a driver for the Surveyor Vision System in ROS. According to the manufacturer, the SRV-1 can also run onboard interpreted C programs or be remotely managed from a base station with Python or Java-based console software. Additional support is also available for RoboRealm machine vision software,[17] Microsoft Robotics Developer Studio, and Cyberbotics Webots.

Another educational robot is the SAR's Bot'n Roll ONE C.[18] It has a differential configuration, supported in a black acrylic base with 22 cm of length and a weight of 1300 g. It provides two infrared obstacle detection sensors with possibility to add extra modules as a line follower component, a LCD for interface with the user (e.g., print program variables) and a RGB color sensor. A USB-Serial (RS232) converter allows the programming of the robot using an external computer. In general, it is an excellent starting kit for beginners. The ROS driver for this platform is currently under development. Basic programming is available using PICAXE editor,[19] as well as C programming through MICROCHIP's MPLAB.[20]

The Hemisson from K-Team [18] is an aluminium platform with 120 mm of diameter. The basic kit only provides limited computational power and a few sensors like 8 infrared light sensors, 6 of them for obstacle detection and 2 facing the ground. It also provides four SMD LEDs and a buzzer that emits sounds at a unique frequency, used to assign the robot different internal behaviors. This platform is extensible providing a front connector for I2C bus communication, a left-side connector that allows flash-memory programming and a right side connector that provides serial port communication. It also supports cameras and ultrasonic sensors beyond others. These extensions may increase both computational power and sensing capabilities. Despite not being integrated in ROS, Hemisson is supported by Cyberbotics Webots. Also, K-Team provides a PIC compiler that allows users to program the robot in C, as well as software to upload cross-compiled files.[21]

Among the larger, more equipped and more powerful mobile robots, a reference platform for research and education is the Pioneer 3-DX from Adept MobileRobots [19]. This is a robust differential drive platform with 8 sonars in a ring disposition, a high-performance onboard microcontroller based on a 32-bit Renesas SH2-7144 RISC microprocessor, offering great reliability and easiness of use. Compared to the previously referred robots, this robot has greater weight and less affordability. Two different drivers are available to interface the Pioneer 3-DX with ROS: *ROSARIA*[22] and *p2os*.[23] Furthermore, MobileRobots provides the Pioneer SDK,[24] a set of robotics applications and libraries for the development of robotics projects. The mobile robot base platform can also be used through interfaces with several other third party software systems, including Player,[25] Matlab[26] and the Python and Java programming languages.

2.2 Arduino-Based Robotic Platforms

Even though most platforms referred in section II-A provide open source software, they usually require a slow learning curve and the hardware has limited expandability. Arduino solutions have recently

[13]http://www.ros.org/wiki/ethzasl_aseba
[14]http://www.coppeliarobotics.com
[15]http://www.cyberbotics.com
[16]http://github.com/rene0/ros-surveyor
[17]http://www.roborealm.com
[18]http://botnroll.com/onec/
[19]http://www.picaxe.com/Software
[20]http://www.microchip.com/mplab
[21]http://www.k-team.com/mobile-robotics-products/hemisson
[22]http://www.ros.org/wiki/ROSARIA
[23]http://www.ros.org/wiki/p2os
[24]http://www.mobilerobots.com/Software.aspx
[25]http://playerstage.sourceforge.net
[26]http://www.mathworks.com

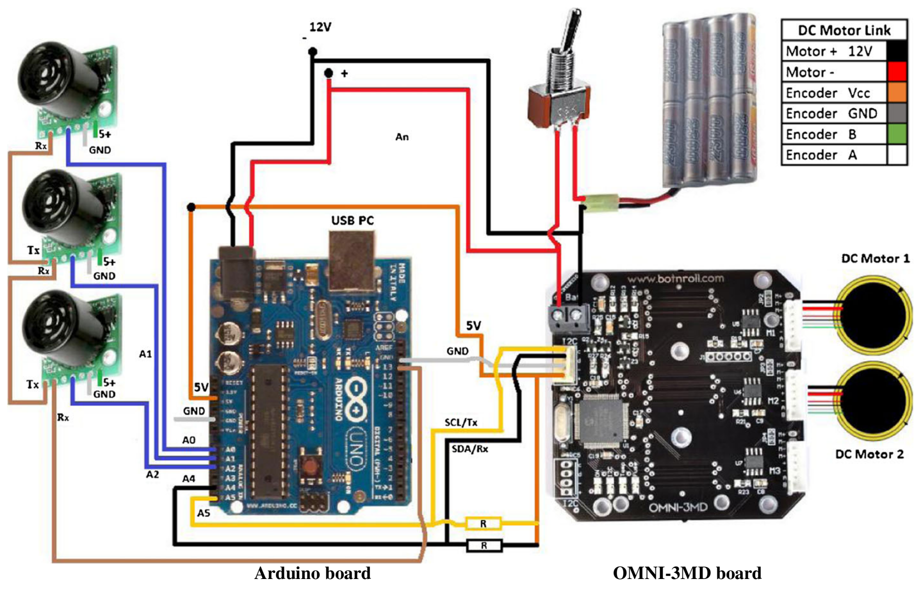


Fig. 2 Main hardware parts of TraxBot

appeared in the market to work around such issues. For this reason, our platforms were built around an Arduino control board [6] (Fig. 2), which accesses the motor encoders and other information from the power motor driver, like temperature and battery state, being also able to send commands to the motors, read sonar information and exchange messages natively through Zigbee. Although this section briefly describes the platforms assembled in our research laboratory, the proposed driver could be applied to any other Arduino-based platform such as the *eSwarBot* [11], the *Bot'n Roll OMNI*[27] and several others (*cf.*, [2]).

The Arduino-based platforms under consideration, namely the TraxBot *v1* and *v2* and the Stingbot are depicted in Fig. 3. All these platforms' processing units consist of Arduino Uno boards, which include a microcontroller ATmega 328p that controls the platforms motion through the use of the Bot'n Roll OMNI-3MD motor driver.[27]

As for power source, two packs of 12 V 2300 mAh Ni-MH batteries ensure good energy autonomy to the robots (around 2–3 h with a netbook atop). For distance sensing, three Maxbotix Sonars MB1300 with a range of approximately 6 m were used. However, and as experimental results depict in Section 4, the sensing capabilities of the platforms can be easily upgraded with other sensors, e.g., laser range finders, RGB depth sensors (e.g., Kinect), temperature, dust and alcohol sensors, etc. Moreover, the platforms have the ability to also include a netbook on top of an acrylic support, which extends the processing power and provides more flexibility. In our case, the ASUS eeePC 1025C has been used due to its reduced price and size.

The netbook provides Wireless Wi-Fi 802.11 b/g/n communication to the robot and is dedicated to run ROS onboard, providing the tools and means for enhanced control of the robot. Additionally, the platforms are also equipped with an Xbee Shield from Maxstream, consisting on a ZigBee communication module with an antenna attached on top of the Arduino Uno board as an expansion module. This Xbee Series 2 module is powered at 2 mW having a range between 40 m and 120 m, for indoor and outdoor operation, respectively.

Having specified the hardware and the platform electronics, the modular control architecture of these robots is summarized in Fig. 4. As Fig. 4 depicts, the Arduino Uno board is used as the central component of the system. The sonars range finders connect to the Arduino board using its analog inputs. As for the connection to the Bot'n Roll OMNI-3MD motor driver, the pins A4 and A5 are used as I2C peripheral. The Bot'n Roll motor driver has the ability to control the motion of the platform by benefiting from a

[27]http://botnroll.com/omni3md

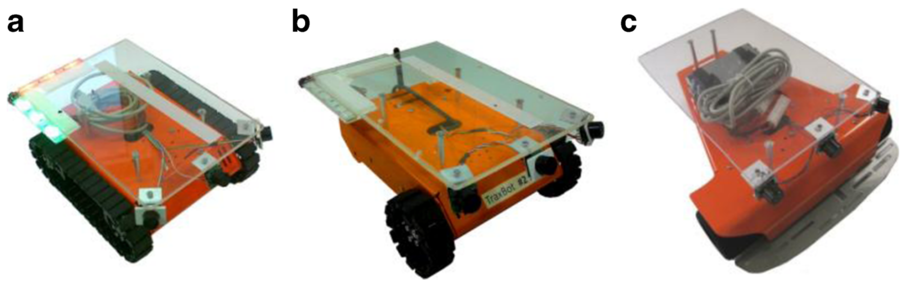


Fig. 3 Arduino-based robotic platforms, **a** TraxBot v1; **b** TraxBot v2; **c** StingBot

PID controller for both velocity (linear and angular) and position of the motors. The feedback is given by the motors integrated encoders. The other analog and digital ports available can be used to integrate more sensors. The USB port of the Arduino board connects to the netbook, receiving (RX) and transmiting (TX) TTL serial data, which is decoded using a USB-to-TTL serial chip. It is also possible to send and receive information wirelessly, by benefiting from a ZigBee shield, or other shields that can be mounted on top of the Arduino board, providing a wireless interface.

2.3 Summary

Both Arduino-based platforms, the TraxBot and StingBot, meet all the requirements previously pointed out, being ideal for multi-robot applications. Our platforms have a similar price to the Mindstorms NXT, being

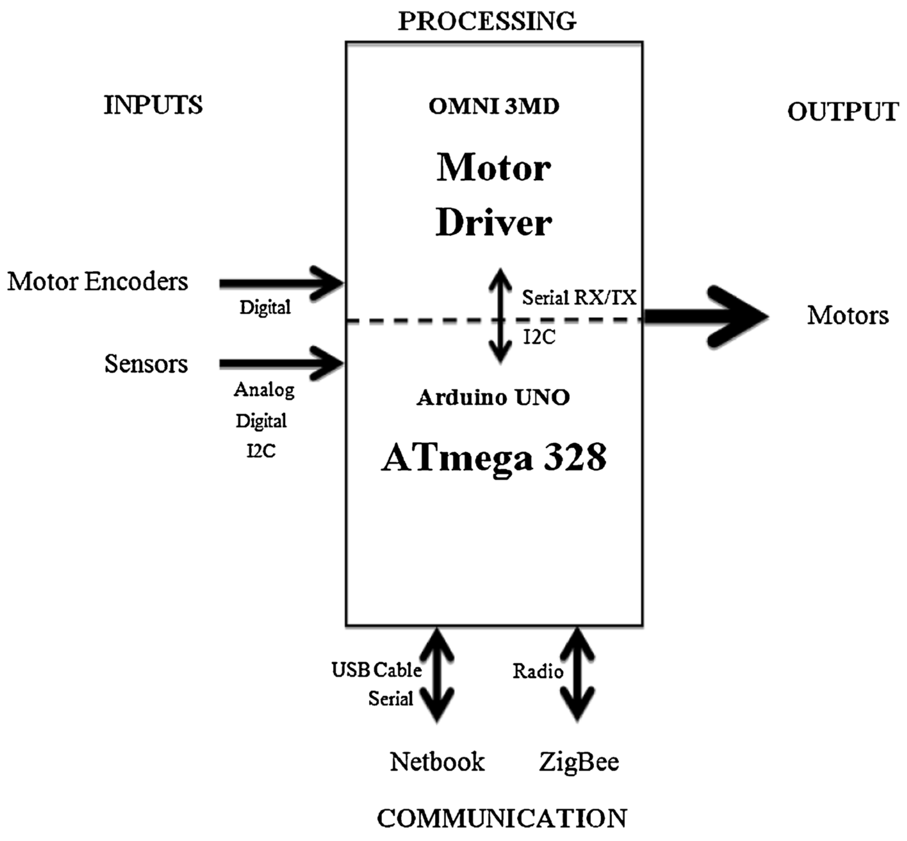


Fig. 4 Control architecture of the robotic platform

more affordable than the Turtlebot, e-puck, MarXbot or the Pioneer. In terms of energy autonomy, both the TraxBot and the Stingbot can operate continuously around 3 h, which is a common operation time for compact platforms. As for communication, unlike the iRobot Create and the Pioneer, which do not offer multi-point communication out-of-the-box, our platforms support Zigbee communication, which can be extended with WiFi when using a netbook atop. Having distance sensors and wheel encoders with high resolution, these platforms have the flexibility to incorporate even more custom sensors, as opposed to the SRV-1 Blackfin or the Mindstorms NXT. Furthermore, its hybrid design enables not only to make use of the 24 MIPS at 26 MhZ Atmega 328 microcontroller, but also the Intel Atom N2800 Dual Core at 1.86 GhZ processor of a typical netbook, similarly to the Turtlebot and outperforming smaller platforms.

Additionally, when developing our educational robots, other requirements were taken into account: all hardware is either made of aluminium or stainless steel, being extremely robust; their dimensions are adequate for both indoor and outdoor experiments; and they have the ability to run ROS, thanks to the driver described in the next section.

3 ROS Driver for Arduino-Based Robots

The key contribution of this work is the development of an open source driver that enables the integration of Arduino-based custom educational robotic platforms in ROS, which can be used as a starting point and easily generalized to a wide variety of other robots also based on Arduino controller boards.

3.1 Driver Description

The *mrl_robots*[28] driver herein presented was developed for integration and control of the platform using ROS Fuerte version running on any Operating System that supports ROS. ROS provides tools to interface with the Arduino family of boards through the *rosserial* stack.[29] However, *rosserial* does not suit the requirements of this work due to the high overhead imposed by its data acquisition and commands, which results in an excessive workload to the Arduino microcontroller Atmel 328p SRAM. In fact, the microcontroller presents limited SRAM memory (only 2 Kbytes) and for standard ROS topics (float32 messages + message headers). Stress tests have shown that only a maximum of 15 ROS topics can be used in parallel and the message buffer is limited to 70 standard messages.

The most important feature in *rosserial* is to add libraries to the Arduino source code, in order to emulate ROS language directly in Arduino code. This results in high overhead in communication between the PC running ROS and the Arduino board, due to the structures used, for example when publishing messages from the Arduino side. For this reason, an alternative way of interfacing with the Arduino board was adopted, and a custom driver was created to communicate faster and transparently with any Arduino board. We propose a solution based on the *serial_communication* stack,[30] wherein messages sent from the Arduino only consist of a simple data packet sent through a standard serial communication protocol. The data packet comprises an array of characters that are parsed to integer variables on the PC/ROS side, decreasing significantly the communication load, thus overcoming the *rosserial* drawback. This enables robust and fast communication in more complex applications, such as teleoperation, crossing of sensory information and the integration of the *navigation* stack, among others. It also has the versatility of using different protocols to exchange data between the Arduino and the PC/ROS side, which enables the creation of a customized and transparent serial communication.

The Arduino firmware code was developed taking into account all components and their features, which are required for the robots' operation. In Fig. 5 the architecture of the ROS Driver is illustrated. The power motor driver OMNI-3MD provides libraries to control the motors (*i.e.*, velocity or position control), read encoders and temperature, as well as setting the parameters for the initial configurations of the PID controller, among others. The motor driver is connected to the Arduino Uno through I2C

[28]http://www.ros.org/wiki/mrl_robots

[29]http://www.ros.org/wiki/rosserial

[30]http://www.ros.org/wiki/serial_communication

communication. C/C++ language was used as the programming language for the ATmega328p microcontroller. Algorithm 1 illustrates the resident Robot/Arduino Firmware code.

Algorithm 1. Robot/Arduino Resident Firmware

```
 1: #Omni3MD library  // main motor driver command functions
 2: #EEPROM library   // storage robot particular specifications: robot ID,...
 3: #Robot library   // range sonars acquisition, calibration, PID gains
 4: #RobotSerialComm library   // protocol serial communication
 5: #Standard libraries
 6: Setup Functions(); // PID motor gains, used ports, encoders scale, set I2C
 7: connection,...
 8: Streaming Functions():
 9:     sendEncodersReads()
10:         Read encoder 1 and 2 pulses;
11:         Serial reply encoder data;
12:     sendEncodersSonarsReads()
13:         Read encoder 1 and 2 pulses;
15:         Read sonars 1, 2 and 3 ranges;
16:         Serial reply encoder and sonar data;
17:     sendRobotInfo()
18:         Read from EEPROM robot ID;
19:         Read internal board temperature;
20:         Read Omni-3MD driver firmware version;
21:         Read TraxBot battery voltage;
22:         Read firmware version;
23:         Serial reply info data;
24: Main loop():
25:     Switch (action):
26:         sendEncodersReads;
27:         sendEncodersSonarsReads;
28:         sendRobotInfo;
29:         Omni-3MD auto-calibration motors for controller purposes;
30:         Set PID gains;
31:         Receive Xbee message;
32:         Send Xbee message;
33:         Xbee node discovery;
34:         Set prescaler from encoders;
35:         Set desire encoders values;
36:         Robot info;
37:         Single encoders reading;
38:         Single sonars reading;
39:         Linear motors move with PID controller;
40:         Linear motors move;
41:         Stop motors;
42:         Reset encoders;
43:         Debug action;            // (Des)Activate debug option
44:         Start streaming data;    // Activate data streaming
45:         Stop streaming data;
```

The protocol developed to interface ROS with the Arduino board consists on sending a frame with the configuration shown in Fig. 6. The character '@' is used at the beginning of every frame, and commas are used to separate the different parameters. Character 'e identifies the end of the frame. Regarding the contents of the protocol, the first parameter corresponds to the action command; like *move motors*, and others (Algorithm 1). Following the action command, commas separate the arguments of the designated commands, which have been defined as signed integers.

Let us suppose, for instance, that we want the platform to move with a linear velocity of 0.5 m/s and an angular velocity of -0.8 rad/s, the frame would be, "@11,500,-800e" representing "@command,(lin_vel $\times$ 10^3),(ang_vel $\times$ 10^3)e".

In the ROS side, a computation process (*robot_node*) has been programmed, which starts the serial connection using the *cereal_port* library of *serial_communication* stack and receives streams of information from the Arduino board, in the robot. Whenever a data frame is received from the serial channel (line 21 of algorithm 2), a callback is triggered, publishing the corresponding message into appropriate ROS topics, thus providing updated information to the rest of ROS ecosystem. Algorithm 2 shows how the driver works.

Algorithm 2. PC/ROS Driver.

```
 1: #ROS_msgs library    //  ROS type messages
 2: #Cereal_port library //  protocol serial communication
 3: Robot data callback():
 4:     UpdateOdometry()
 5:         Read encoder pulses;
 6:         Pulses converted to cartesian pose $(x, y, \theta)$;
 7:         Publish updated pose in a ROS topic;
 8:         Publish tf: odom $\rightarrow$ base_link
 9:     DriveRobot()
10:         Callback to subscribed ROS topic with differential velocity commands ;
11:         Send to robot angular and linear speed;
12:     RangeUltrasonicSonars()
13:         Publish the range of the ultrasonic sonars in ROS topics;
14:     XbeeMsgs()
15:         Publish ROS topic with Xbee message received by the robot;
16:         Callback to subscribed ROS topic with message to send to other robots;
17:     UpdateRobotInfo()
18:         Publish ROS topic with robot information;
19: Main loop():
20:     Establish a serial connection;
21:     Receive serial data from robot (activating ROS callbacks)
```

In Fig. 7, it is shown how a ROS user application node (e.g., a mapping algorithm) can interact with *robot_node* by sending velocity commands to the base and receiving information like sonar range, odometry, transforms, etc. One of many ROS tools, *rxgraph*, has been used to allow real time monitoring of the available nodes, as well as topics exchanged by each node. Note also the interaction with other existing nodes in ROS like the *wiimote_node,*[31] used for teleoperating the robot through a Nintendo's Wii remote controller (*WiiMote*), and the *hokuyo_node,*[32] which provides sensor readings from an Hokuyo laser range finder to the robot. ROS also provides many different built-in sensor message types, which are appropriately assigned to the topics of each component of the driver.

The ability to stream data from the Arduino board is an interesting feature of the driver because it does

[31] http://www.ros.org/wiki/wiimote_node

[32] http://www.ros.org/wiki/hokuyo_node

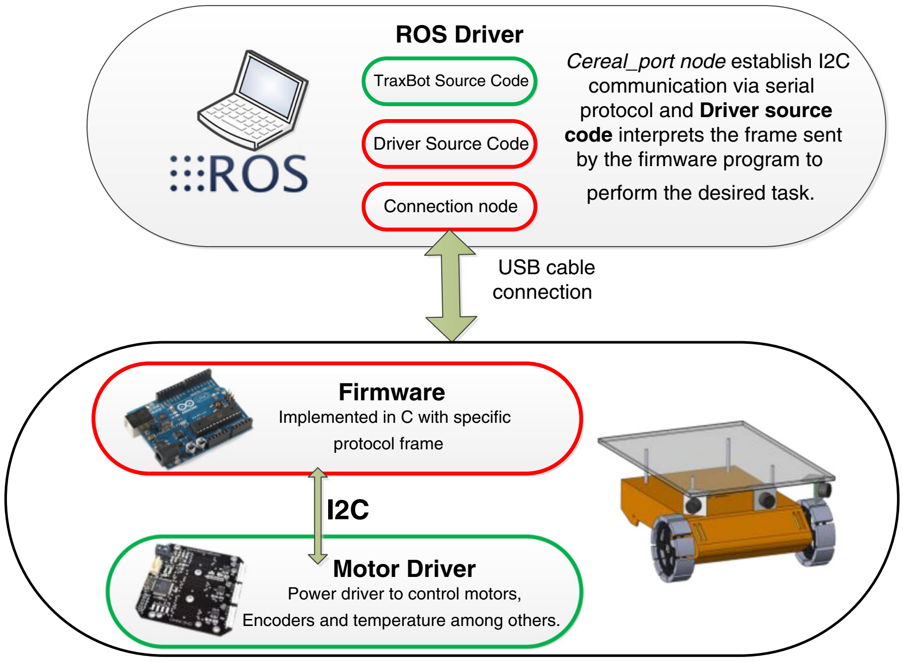
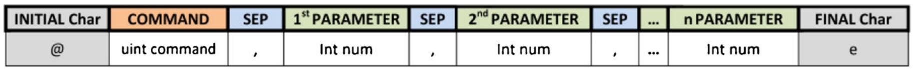

ROS Driver
TraxBot Source Code
Driver Source Code
Connection node
ROS
Cereal_port node establish I2C communication via serial protocol and Driver source code interprets the frame sent by the firmware program to perform the desired task.
USB cable connection
Firmware
Implemented in C with specific protocol frame
I2C
Motor Driver
Power driver to control motors, Encoders and temperature among others.

Fig. 5 Diagram of the ROS driver architecture



INITIAL Char	COMMAND	SEP	1st PARAMETER	SEP	2nd PARAMETER	SEP	...	n PARAMETER	FINAL Char
@	uint command	,	Int num	,	Int num	,	...	Int num	e

Fig. 6 Frame protocol to receive/send data from/to the Arduino Uno

not require a synchronous communication involving requests and responses between ROS and the Arduino board. Hence, it frees the serial communication channel since it only needs a starting request and can be stopped at any time. Furthermore the *mrl_robots* driver has the ability to enable and disable debugging options to track eventual errors.

3.2 Driver Features and Potential

The driver presented in the last subsection offers several features, many of which are inherited by the direct integration with the ROS middleware. The driver enables the robot interface with ROS tools for data processing and analysis, like 3D visualization (*rviz)*, logging real-time robot experiments and playing them

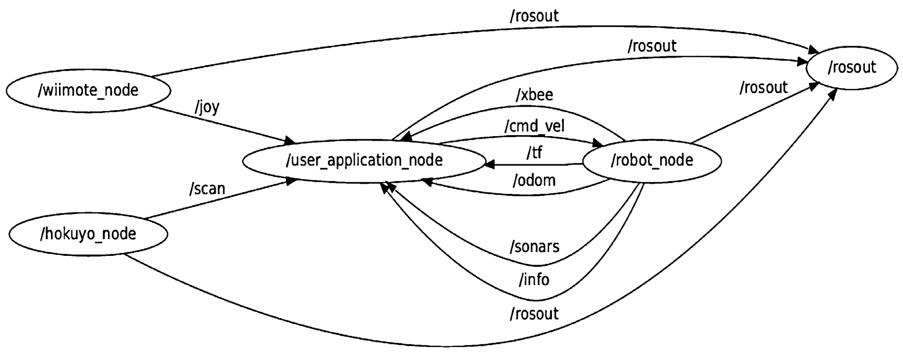


Fig. 7 *rxgraph* nodes and topics provided by the *mrl_robots* driver

offline with (*rosbag/rxbag)*, plotting data (*rxplot)* and visualizing the entire ROS network structure (*rxgraph*, e.g. Fig. 7*)*.

Beyond the easiness of using the available tools, ROS also provides effortlessly integration of new sensors in the robot without needing high hardware expertise, as will be demonstrated in Section 4. This opens a new range of possibilities since several well-known stacks from the ROS community comprise algorithms for robotics development such as the *navigation*[2] and *slam_gmapping*[33] stacks. These are useful since they free users from programming low-level and basic behaviors for their robots by reusing the code that is available in ROS, thus staying more focused in the main aspects of their scientific experimentation thanks to fast prototyping. Moreover, they enable users who are beginners in robotics to easily deploy a robot running with basic low-level and navigation behaviors, thus being an excellent tool for educational robotics. The interaction between high-level programs and the available resources in Arduino-based platforms enables hardware abstraction. This brings the possibility of using standard interfaces of any other mobile robotic platforms also integrated in ROS.

Another interesting feature of the driver is the simplicity for enabling heterogeneous multi-robot coordination and cooperation. Running the same hardware abstraction layer in the team of robots, ROS takes care of the communication messaging system using a publish/subscribe method, which enables all kinds of interaction between members of the same team, as seen in Fig. 8 where an example of a ROS network is depicted. It is also shown how the network has the flexibility to work with a large variety of integrated solutions, which enables the assignment of particular tasks for specific members of the team. Robots of the same team may have different purposes and perform different tasks, meaning that heterogeneous teams with different capabilities can coexist. For example, in a search and rescue scenario, the cooperation of a multi-robot team may arise from performing different tasks with hundreds or thousands of heterogeneous robots. In a scenario where a swarm of flying scouts are deployed [4], these scouts may search and mark objective points to pass them on, in real time, to ground robot surveillance teams [3].

[33]http://www.ros.org/wiki/slam_gmapping

Although ROS is not the only framework that has the potential to integrate homogenous and heterogeneous robotic teams, it also allows integrating mixed real and virtual robot teams sharing the same scenario and mission. Using the driver herein presented together with Stage [8] or Gazebo [20], the same code can be used in both physical robots and virtual simulated robots coexisting in the same network. In addition, the communication between real and virtual robots is completely transparent since they are both running ROS. This major feature is ideal to perform multi-robot tasks, allowing the use of a large population of robots, when no extra physical robots are available, being cheaper and promoting safer test scenarios, by making interactions between physical and virtual world objects [21].

Multi-robot simulation is available for ROS through wrappers of Stage and Gazebo, which support a high number of interacting robots depending on the computational complexity of the algorithm. However, the opportunity to execute nodes on multiple CPUs in the same network turns the upper bound of the team size arbitrarily high, removing the overloading processing of a single machine.

Finally, another particularity of using the ROS driver developed is a means to improve security in multi-robot tasks, for example in a military operation, where it is possible to create a specific encrypted node for a robot of the team [22], making the system more robust to attacks. Additionally, it is always possible to monitor, log and debug at any time all exchanged information and specific nodes running in the network.

4 Results and Discussion

In order to experimentally evaluate the ROS driver developed to Arduino-based mobile robots, some tests were conducted using physical robots and simulated robots in Stage. Besides running virtual agents, the simulator allows to compare physical robots' odometry and pose estimation with the ground truth information. We present experimental tests that validate the aforementioned claims and we also show cooperative behaviors with real multi-robot systems, as well as mixed teams of real and virtual robots.[34] The aim is

[34]**A video of the experiments is available at:** http://www.isr.uc.pt/~aaraujo/videos/JINT2013

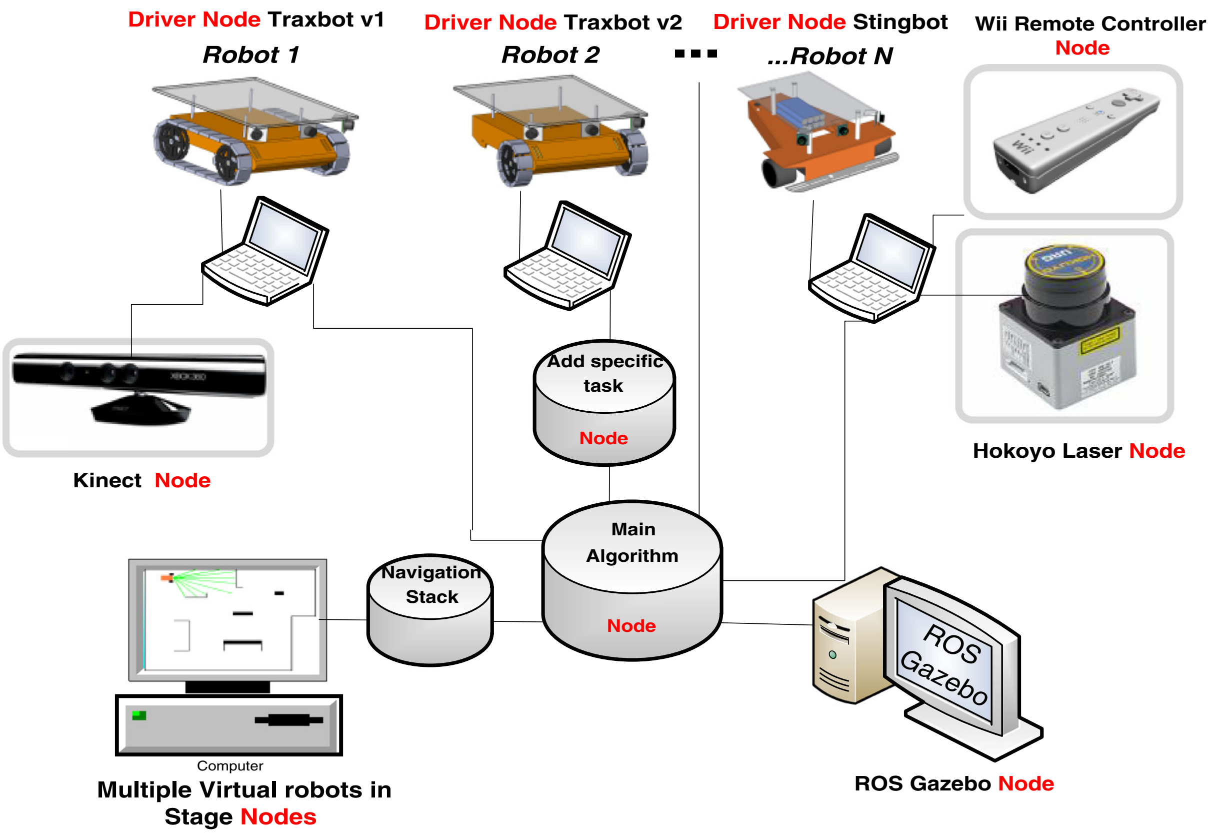


Fig. 8 Network topology example with multiple robots, sensors, tele-operation devices and applications

to evaluate the driver flexibility when interacting with different sensors, the ability to easily integrate new sensors in the Arduino-based robot, update the ROS driver, the driver portability to different robotic platforms, and the driver integration with the existent ROS tools.

4.1 Driver Flexibility to Interact with other Nodes in ROS Network

The first experiment aims to demonstrate the driver flexibility to interact with different sensors (Fig. 8). The TraxBot *v1* platform was equipped with a laser range finder (LRF) and its performance was compared against the native ultrasonic range sonars on a simple mapping task [23]. The Hokuyo URG-04LX is a LRF classified as an Amplitude Modulated Continuous Wave (AMCW) sensor [24].

In order to test the sonars performance, an L-shaped scenario of 2 m by 1.6 m was set up, with a 1 m width (Fig. 9). To perform this test, two lateral sonars placed at ±45° were used (see Fig. 10a). The front sonar was used for reactive obstacle avoidance, given that in mapping situations, due to the low resolution of the sonar sensor, the projection of walls and obstacles detected by the cone of the front sonar is placed in front of the robot, which is highly unreliable in most situations as shown in Fig. 10b.

In this test, the robot movement relies solely on odometry. In Fig. 9a, it can be seen in the first rectilinear motion that the sonars readings are stable (red dots) and coincident with the ground truth scenario limits (blue line). Green dots represent the midpoint of sonars acoustic beam while turning. Some issues arise during the 90° rotation, since the sonar beam cone has an opening of approximately 36°, thus presenting a loss of resolution (see Fig. 9a). In the case of Fig. 9b, the Hokuyo LRF was used to perform the same test. The laser field of view was set to 180° with a rate of 512 samples per reading. It is possible to observe some discrepancy in some readings especially at the

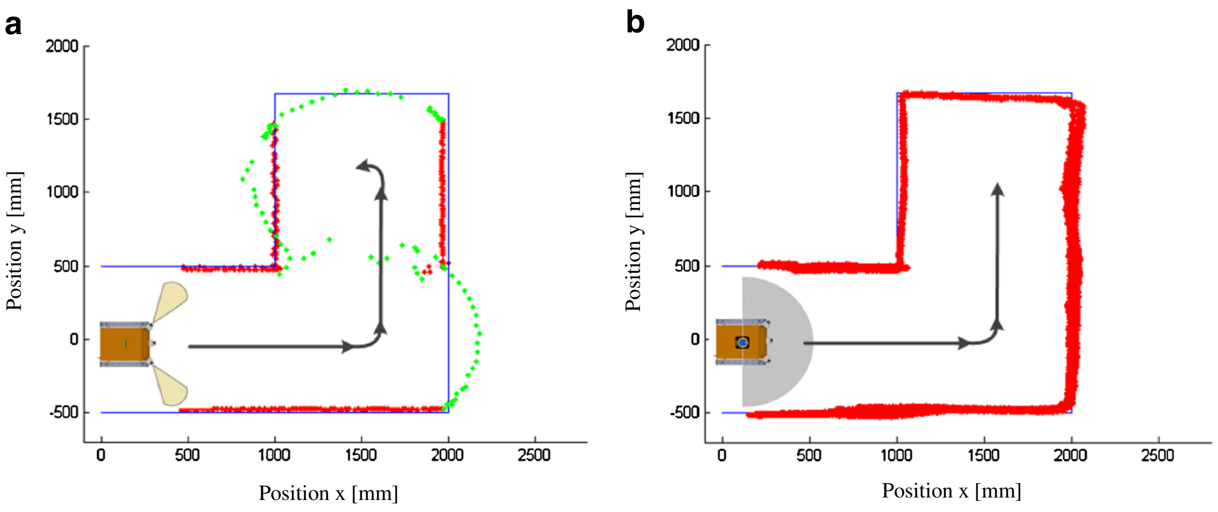


Fig. 9 Evaluation of the ROS driver in Traxbot *v1* with different sensors. **a** Ultrassound Range Sensors integration; **b** Hokuyo URG-04LX Laser Range Finder integration

end of the movement, due the odometry position error accumulated during motion.

As can be observed, the ROS driver presents a high flexibility on interacting with other nodes in the ROS network. In this particular case, a simple mapping task was considered by comparing the performance of the TraxBot when equipped either with sonars or LRF, showing that both sensors communicate seamlessly with the ROS driver. The trajectory of the robot was also followed successfully due to the motor control component of the driver developed.

4.2 Driver Portability to Different Robots and Integration in ROS

In the second experiment, the goal was to demonstrate the portability of the driver to different Arduino-based robots using other sensors, which *i*) enables testing the driver in different variants of Arduino-based robots; and *ii)* illustrates the integration of these robots and the corresponding ROS driver with hardware devices and diverse algorithms that are already available in ROS. Hence, a mapping task, which includes the interaction with a laser range finder (LRF) and a joystick

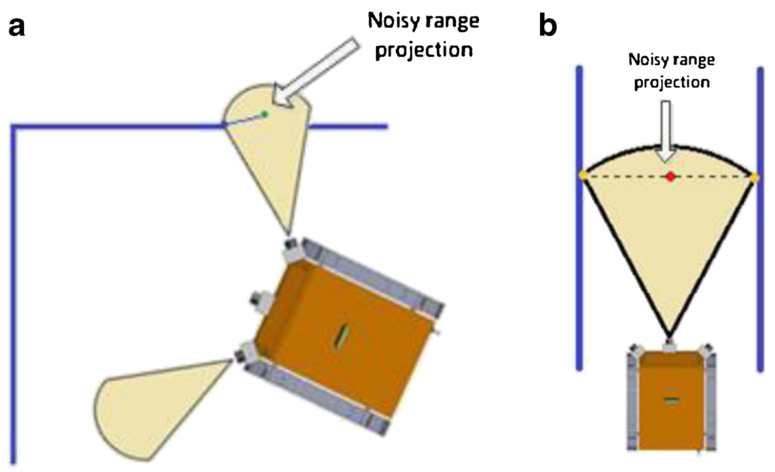


Fig. 10 Noisy range situations: **a** issue during rotations using lateral sonars; **b** the front sonar was not used for mapping

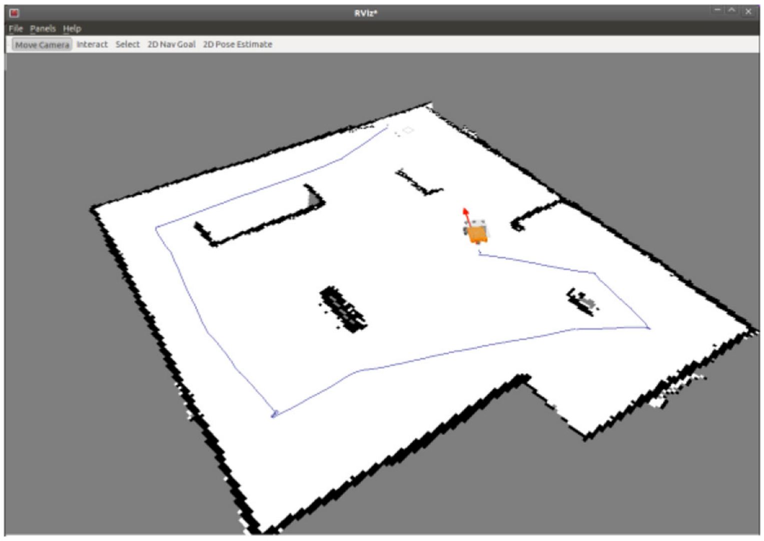


Fig. 11 Map generated by the Traxbot *v2* with hector mapping in *rviz*

controller was performed. This time, the Traxbot *v2* platform was equipped with an Hokuyo LRF and teleoperated with a *WiiMote* for a mapping task using Hector Mapping [25], which is available in the *hector_slam* stack.[35]

The teleoperation node runs on the eeePC netbook, which connects to the ROS network. The node subscribes a ROS topic with the information about the *Wiimote* state, and assigns functions for each pressed button, publishing then velocity commands that are interpreted by the ROS driver, which in turn generates motor commands sent to the robot. The wiimote driver uses Bluetooth to pair the joystick with the netbook.

Additionally, the *hector_mapping* node subscribes to the scans provided by the *hokuyo_node* and publishes the estimate of the robot's pose within the map, while generating the map. Figure 11 presents the resulting map in *rviz*, and corresponding robot trajectory, which was obtained with *hector_mapping* on our experimental lab arena.

In a related experiment, we show not only the possibility to have coordinated behaviors with two physical robots, but also the possibility to include a third simulated robot running in *stage*, forming a mixed team of two real robots and one virtual robot, as described in section III-B. In addition, we also integrate navigation capabilities in our robots, by running the *navigation* stack[2] with a known map, in this case, the map of an experimental area in our lab (Fig. 12).

[35]http://www.ros.org/wiki/hector_slam

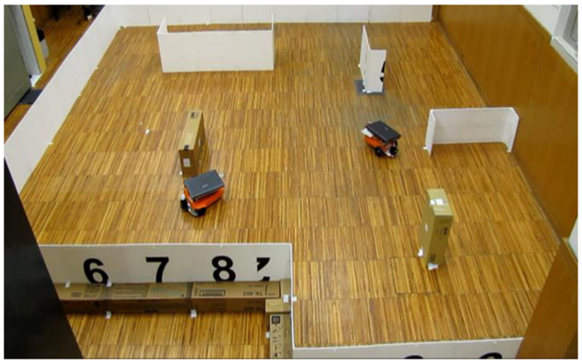


Fig. 12 Experimental arena with a Traxbot *v2* and a Stingbot cooperating with a virtual robot, running on *stage*

The robots were commanded to navigate cyclically between a set of waypoints in the arena, as shown in the video of the experiment[34]. To further demonstrate their coordination, a new trial was considered, wherein each robot ran an independent *roscore* and a common waypoint for all three robots was defined. The robots had to exchange messages through a shared ROS topic using the *wifi_comm*[36] package, in order to avoid going to a common point at the same time: robots waited before starting to move to the point at the center of the arena; and priority was given to the robot that firstly expressed intention to move to it. All three robots were able to coordinate themselves in the environment without colliding to each other, due

[36]http://wiki.ros.org/wifi_comm

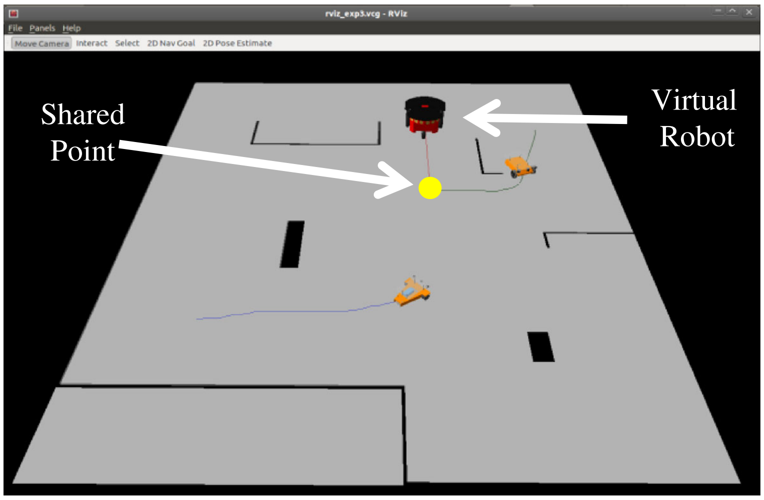


Fig. 13 The three robots coordinating their behaviors by exchanging ROS messages (rviz)

to the integration of the *navigation* stack[2]. Figure 13 presents a snapshot of rviz, illustrating the three robots moving in the arena.

In this section, it was possible to verify the full integration of the ROS driver developed, by running a generic *SLAM* approach and the navigation stack available in the ROS community in our robots and apply them to cooperative algorithms. Moreover, it was possible to evaluate the portability of the driver to other Arduino-based robotic platforms. With only minor adjustments in the kinematics parameterization caused by the different physical properties of the robots, it was possible to use the same ROS driver in similar robotic platforms: *TraxBot v2* and *StingBot*.

4.3 Adding other Sensors to the Arduino-Based Robots

In this experiment, we demonstrate the simplicity of extending the driver of the Arduino-based robots. This is done through the physical connection of sensors to the Arduino and their integration in the ROS driver. Three new sensors were assembled in the TraxBot *v2* plataform. These sensors have been used in the scope of the CHOPIN (Cooperation between Human and rObotic teams in catastroPhic INcidents) R&D Project[37] to measure variables related with the occurrence of hazards situations in urban search and rescue (*USAR*) missions, more specifically fire outbreaks, leakage of toxic gases and detection of victims [26].

To add this three new sensors, one only needs to add the corresponding sensor data acquisition in the firmware illustrated in Algorithm 1 (section III.A), more precisely in the *Streaming Functions*. Consequently, in the ROS side (Algorithm 2), only the corresponding callback functions are necessary to add in order to interpret the new sensor data.

The dust sensor (model *PPD42NS*) is illustrated in Fig. 14a. Being manufactured by Grove, this sensor returns a modulated digital output based on the

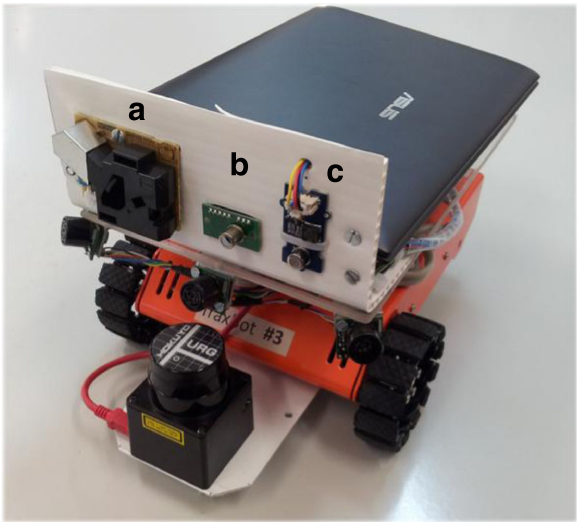


Fig. 14 Sensors setup in the TraxBot v2. **a** Dust sensor; **b** Thermopile array sensor; **c** Gas sensor

[37]http://chopin.isr.uc.pt/

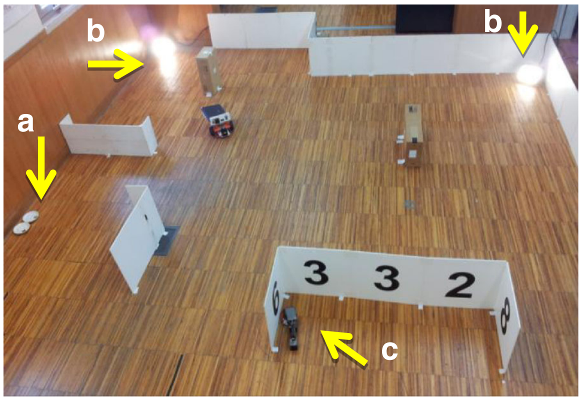


Fig. 15 Points of interest in testing scenario: **a** alcohol containers; **b** spotlights with 500 W which produce high temperature; **c** dusty air flow area

detected Particulate Matters (*PM*), which can be used to measure smoke density near a fire outbreak. The output responds to *PM* whose size is around 1 micro meter or larger. Considering D as the number of particles with at least 1μm diameter, the output of the dust sensor is defined as:

$$0 \le D \le 40000[\text{pcs/liter}]. \quad (1)$$

The sensor depicted in Fig. 14b, is the thermopile Array sensor (model TPA81). This sensor is characterized by its ability to output an array of 8 elements of 8 bits each. The analog value corresponds directly to the temperature. Hence, one may define the thermopile output as:

$$10 \le T_i \le 100\ [°\text{C}], \quad (2)$$
$$(T_i, i = 1, \ldots, 8.\ T_i\ 8\ bits\ entry\ T = \max_i v_i).$$

The last sensor (Fig. 14c), is the Alcohol Sensor (model MQ303A). This sensor has the feature to output (A) a voltage inversely proportional to the alcohol concentration in the air:

$$0 \le A \le 700\ \ mV. \quad (3)$$

The three sensors were assembled in a plastic support and mounted in the front part of the robot as depicted in Fig. 14. This configuration was chosen for a better analysis of the natural air flow generated by the robot movement during the scenario exploration. Moreover, this configuration took into consideration a better positioning of the field of view for the thermopile array sensor, due to its horizontal array configuration. The dust sensor was connected to a digital port, the alcohol sensor to an analogic port and the thermopile array sensor via I2C Arduino ports.

As previously stated, the choice of these sensors took into account the requirements of the CHOPIN project. Therefore, the dust sensor was chosen to work with the thermopile to detect fire or victims, and with the alcohol sensor to detect air contamination, like gas leaks and different sources of fire.

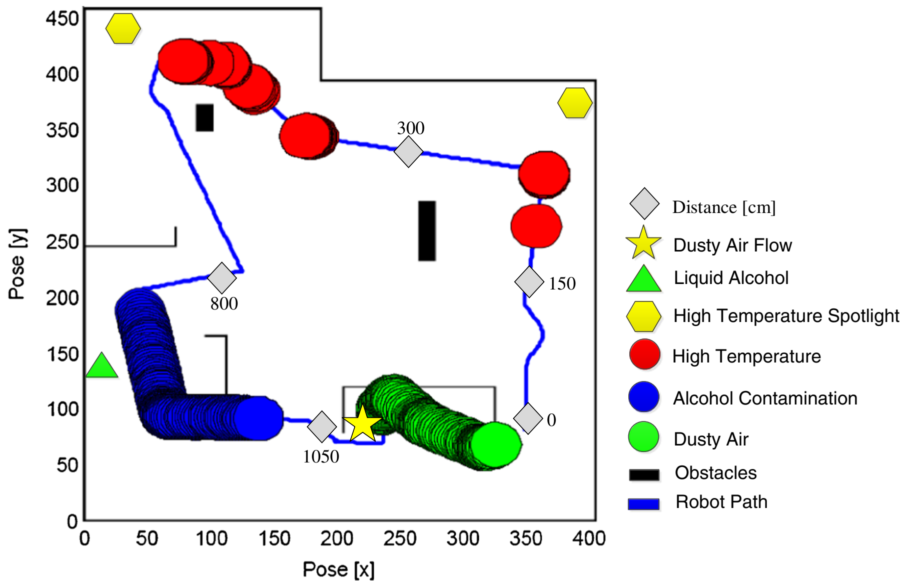


Fig. 16 Map of environmental variables during the experimental test

The experiment with the TraxBot v2 (Fig. 14) was conducted in the same area of the laboratory from section V-B, with a size of 4.0 × 4.6 m and several obstacles placed on it (see Fig. 15). Four points of interest were introduced to simulate critical conditions. More specifically, gas air contamination was simulated with liquid alcohol in containers (Fig. 15a); a fire outbreak was emulated using a 500 *watts* spotlight able to produce a hot spot of high temperature (Fig. 15b); and a small dusty area was simulated with a 120 mm fan creating disturbance in the air, lifting up dust particles on the floor and thus increasing the concentration of dust particles in the air (Fig. 15c).

In the experiment carried out in this scenario, it was possible to map the 3 environmental variables measured by the aforementioned small sensors. The data provided by the three sensors is represented on top of the map with circles of different colors: red circles for

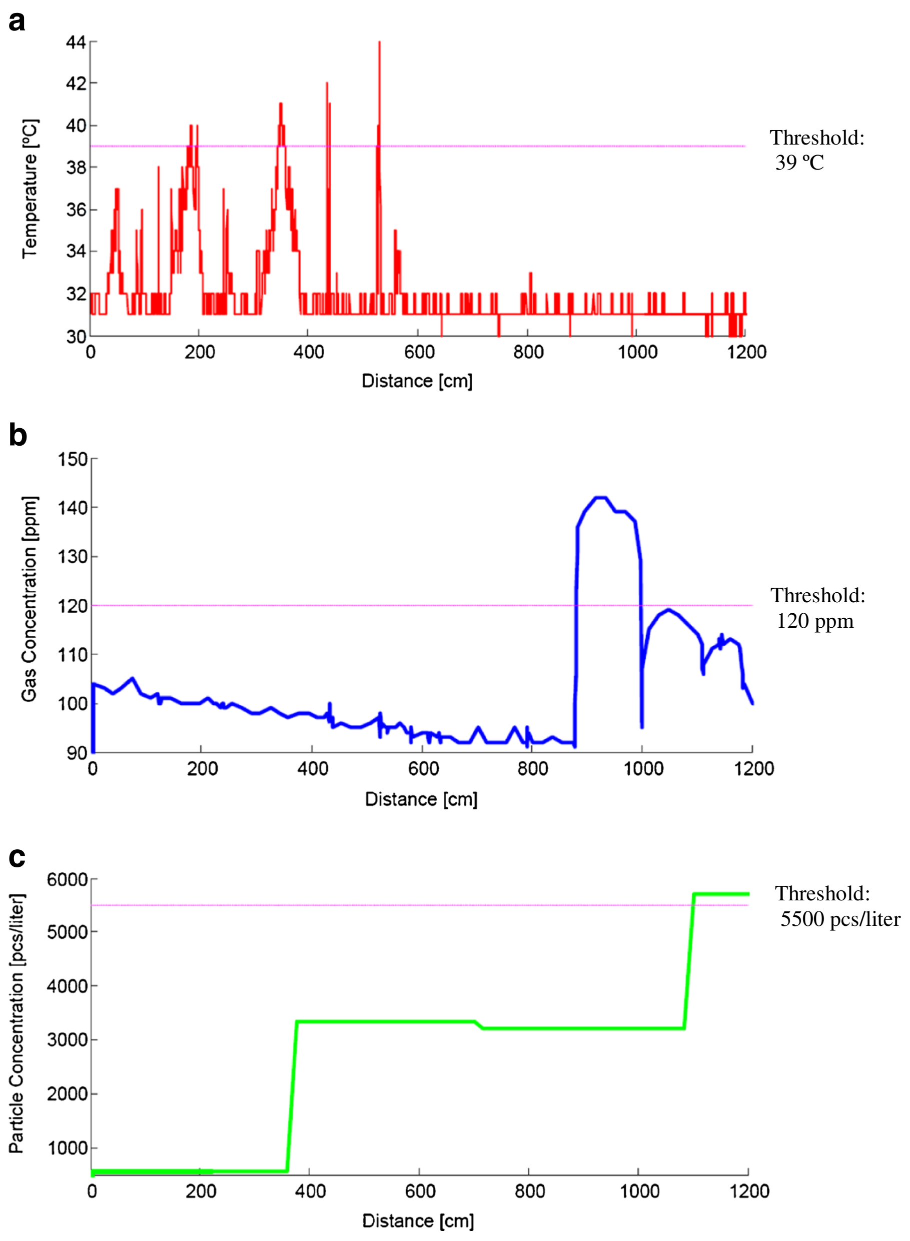


Fig. 17 Sensors output during the experiment. **a** Temperature; **b** Alcohol; **c** Dust concentration

temperature, green circles for dust and blue circles for air contamination (see Fig. 16).

Comparing Fig. 15 and the output map based on sensor readings in Fig. 16, we can observe that this map matches with the real scenario. The red circles represent points in the environment where the thermopile array measures values above 39 °C (near the spotlights). This was predictable due to the high sensing capability of the thermopile sensor for distances of up to one meter. The blue circles represent the air contamination variable. This allows retrieving the spread of the contamination caused by the natural air flow along the arena. The green circles, represent points (Fig. 15c) where dust particles density are higher than 5500 *pcs/liter*.

A netbook using Ubuntu 11.10 and *ROS* [7] *Fuerte*[38] was placed on top of the TraxBot v2. To explore the scenario, the robot was teleoperated using the Wiimote controller. The ROS driver was updated to support the new sensors, thus making their information available on the ROS network. TraxBot odometry data provided by the ROS driver was used in this experiment, resulting in the path depicted in Fig. 16.

Figure 17 shows the data acquired from the three small sensors by the TraxBot *v*2 robot during the experimental test.

In this last result section, it was possible to verify the possibility to easily add new sensors in the Arduino-based robot using ROS, which can be used in a wide range of applications. In this case, it was used to perform a trial to validate an approach for environmental monitoring based on thresholds, e.g., for a search and rescue application.

5 Conclusions and Future Work

In this article, a solution for integrating Arduino-based robotic platforms in ROS through the development of a ROS driver was presented. The advantages of integrating these platforms with ROS middleware, enabling the usage of a wide range of tools and reducing the development time through code reuse, were shown and discussed. The robots, alongside with Arduino and ROS open-source development tools, present themselves as ideal platforms for educational robotics. Beyond providing access to all ROS tools, the driver also simplifies the robotic development by: *i*) supporting hardware abstraction to easily control the platform; *ii*) allowing for the extension and integration of all kinds of sensors; and *iii*) enabling multi-robot cooperation and coordination through the operation in a ROS network, both for real teams of homogeneous and heterogeneous robots, as well as hybrid teams of physical robots and simulated robots running the same code.

Results from experiments that were conducted demonstrate all these features. Moreover, the negligible overhead imposed by the driver did not restrict any of the experiments conducted. Finally, as open-source development tools were used throughout the work presented in this article, all the source code developed is available online[39] in the hope of being useful for robotic developers worldwide.

These results open new perspectives into applying the proposed solutions at the hardware, firmware and software levels, as an innovative education tool for engineering in a broad sense. The low cost, open-source, and highly flexible nature of the proposed approach make these mobile robotic platforms and corresponding algorithms accessible for both educational and research purposes, which one can exploit to teach a wide range of topics in Mobile Robotics, and also to pursue research on relevant topics, such as autonomous robots, cooperative robotics, artificial perception, and robots interaction with human, leveraging from hardware abstraction.

Acknowledgments This work was supported by the CHOPIN research project (PTDC/EEA-CRO/119000/2010), by PhD scholarships SFRH/BD/64426/2009 and SFRH/BD/73382/2010, and by ISR-Institute of Systems and Robotics (project PEst-C/EEI/UI0048/2011), all of them funded by the Portuguese Science Agency "Fundação para a Ciência e a Tecnologia" (FCT).

References

1. Petrina, A.M.: Advances in robotics. In: Automatic Documentation and Mathematical Linguistics, vol. 45, No. 2, pp. 43–57. Allerton Press, Inc. (2011)
2. Brooks, R.A.: New approaches to robotics. Science **253**, 1227–1232 (1991)
3. Portugal, D., Rocha, R.P.: Distributed multi-robot patrol: a scalable and fault-tolerant framework. Robot. Auton. Syst. **61**(12), 1572–1587, Elsevier (2013)

[38] http://ros.org/wiki/fuerte

[39] http://www.ros.org/wiki/mrl_robots

4. Couceiro, M.S., Rocha, R.P., Ferreira, N.M.F.: A PSO multi-robot exploration approach over unreliable MANETs. Adv. Robot. **27**(16), 1221–1234, Robotics Society of Japan (2013)
5. Warren, J.-D., Adams, J., Molle, H.: Arduino robotics. Springer Science and Business Media (2011)
6. Araújo, A., Portugal, D., Couceiro, M., Figueiredo, C., Rocha, R.: TraxBot: assembling and programming of a mobile robotic platform. In: Proc. of the 4th International Conference on Agents and Artificial Intelligence (ICAART 2012). Vilamoura (2012)
7. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.Y.: ROS: an open-source robot operating system. In: Proc. Open-Source Software workshop of the International Conference on Robotics and Automation. Kobe (2009)
8. Gerkey, B., Vaughan, R., Howard, A.: The player/stage project: tools for multi-robot and distributed sensor systems. In: Proc. of the Intl. Conf. on Advanced Robotics, pp. 317–323. Coimbra (2003)
9. Grisetti, G., Stachniss, C., Burgard, W.: Improved techniques for grid mapping with rao-blackwellized particle filters. In: IEEE Transactions on Robotics (2006)
10. Rusu, R., Cousins, S.: 3D is here: Point Cloud Library (PCL). In: Proc. of International Conference on Robotics and Automation (ICRA 2011). Shanghai (2011)
11. Couceiro, M.S., Figueiredo, C.M., Luz, J.M., Ferreira, N.M.F., Rocha, R.P.: A low-cost educational platform for swarm robotics. Int. J. Robot. Educ. Art **2**(1), 1–15 (2012)
12. Park, I.W., Kim, J.O.: Philosophy and strategy of minimalism-based user created robots (UCRs) for educational robotics - education, technology and business viewpoint. Int. J. Robot. Educ. Art **1**(1), 26–38 (2011)
13. Kuipers, M.: Localization with the iRobot create. In: Proceedings of the 47th Annual Southeast Regional Conference ACM (ACM-SE 47). Clemson (2009)
14. Bagnall, B.: Maximum LEGO NXT: Building Robots with Java Brains. Variant Press, Winnipeg, Manitoba (2007)
15. Mondada, F., et al.: The e-puck, a robot designed for education in engineering. In: Proc. of the 9th Conf. on Autonomous Robot Systems and Competitions **1**(1):59–65 (2009)
16. Bonani, M., Longchamp, V., Magnenat, S., Rétornaz, P., Burnier, D., Roulet, G., Mondada, F.: The MarXbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In: Int. Conf. on Intelligent Robots and Systems. Taipei, 18–22 Oct (2010)
17. Cummins, J., Azhar, M.Q., Sklar, E.: Using surveyor SRV-1 robots to motivate CS1 students. In: Proceedings of the AAAI 2008 Artificial Intelligence Education Colloquium (2008)
18. Mitrović Srđan, T.: Design of fuzzy logic controller for autonomous garaging of mobile robot. J. Autom. Control **16.1**, 13–16 (2006)
19. Zaman, S., Slany, W., Steinbauer, G.: ROS-based mapping, localization and autonomous navigation using a pioneer 3-DX robot and their relevant issues. In: Proc. of the IEEE Saudi International Electronics, Communications and Photonics Conference. Riad (2011)
20. Linner, T., Shrikathiresan, A., Vetrenko, M., Ellmann, B.: Modeling and operating robotic environent using Gazebo/ROS. In: Proceedings of the 28th International Symposium on Automation and Robotics in Construction (ISARC2011), pp. 957–962. Seoul (2011)
21. Wunsche, B., Chen, I., MacDonald, B.: Mixed reality simulation for mobile robots. In: Proc. of International Conference on Robotics and Automation (ICRA 2009). Kobe (2009)
22. Wagner, A., Arkin, R.: Robot deception: recognizing when a robot should deceive. In: Proc. of Computational Intelligence in Robotics and Automation (CIRA). Daejeon (2009)
23. Rocha, R., Dias, J., Carvalho, A.: Cooperative multi-robot systems: a study of vision-based 3-D mapping using information theory. In: Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA'2005), pp. 386–391. Barcelona (2005)
24. Kneip, L., Tâche, F., Caprari, G., Siegwart, R.: Characterization of the compact Hokuyo URG-04LX 2D laser range scanner. In: Proceedings of the IEEE International Conference on Robotics and Automation. Kobe (2009)
25. Kohlbrecher, S., Meyer, J., von Stryk, O., Klingauf, U.: A flexible and scalable SLAM system with full 3D motion estimation. In: Proc. of the IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR'2011), pp. 50–55. Kyoto (2011)
26. Couceiro, M.S., Portugal, D., Rocha, R.P.: A collective robotic architecture in search and rescue scenarios. In: Proc. of 28th Symposium on Applied Computing (SAC 2013), pp. 64–69. Coimbra (2013)