Integrating Arduino-based Educational Mobile Robots in ROS

André Araújo, David Portugal, Micael S. Couceiro and Rui P. Rocha

***Abstract*— This article presents the full integration of compact educational mobile robotic platforms built around an Arduino controller board in the Robotic Operating System (ROS). To that end, a driver interface in ROS was created to drastically decrease the development time, providing hardware abstraction and intuitive operation mode, allowing researchers to focus essentially in their main research motivation. 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, sensors and teleoperation devices, thereby targeting engineering education. To validate the approach, diverse experimental field tests were conducted using different Arduino-based robotic platforms.**

I. Introduction

Mobile robotics is a technological field and a research area which has witnessed incredible advances for the last decades. It finds application in areas like automatic cleaning, agriculture, support to medical services, hazard environments, space exploration, military, intelligent transportation, social robotics, and entertainment [1]. In robotics research, the need for practical integration tools to implement valuable scientific contributions is felt frequently. However, roboticists end up spending excessive 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 and swarm robotics) and are also ideal for educational purposes. With such assumptions in mind, we have been doing engineering and research work with two Arduino-based mobile platforms [2]: the TraxBot [3] 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 [2].

Following the trend of research, in this work the focus is 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 Robotic Operating System (ROS) [4]. ROS is currently the most trending and popular robotic framework in the world, reaching critical mass and being the closest one to become the 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. The major goals of ROS are hardware abstraction, 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 [5], navigation capabilities [2], visual SLAM [6] and 3D point cloud based object recognition [7], among others. Regular updates enable the users to obtain, build, write and run ROS code across multiple computers.

In the next section, we review general purpose and educational mobile robots, focusing on those already integrated in ROS and briefly describe our Arduino-based robot platforms. In section III, the main contributions of this work are revealed and details on the development of the ROS driver and its features are presented. In the subsequent section, preliminary results with physical Arduino-based robots and a team of mixed real and virtual cooperating agents are presented. Finally, the article ends with conclusions and future work.

II. Related work

The following requirements, sorted by relevance, can be expected from robots to be used for educational purposes [8][9]:

- 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.*, cooperative systems);
- 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).

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).

A. Araújo, D. Portugal, M. Couceiro and R. P. Rocha are with the Institute of Systems and Robotics, University of Coimbra, Pólo II, 3030-290 Coimbra, Portugal, email: {aaraujo, davidbsp, micaelcouceiro, rprocha}@isr.uc.pt.

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

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

Fig. 1. 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.

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.

A. 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 Roomba Create [10] from iRobot 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, with extra space for larger sensors (*e.g.*, 2D laser sensor 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 in the *brown_drivers* stack), as well as the original vacuum cleaning Roomba (*roomba_robot* stack).

In fact, a popular off-the-shelf robot, developed at Willow Garage, has been built upon an iRobot Create: the TurtleBot[3]. 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), being fully open source and exploring all combined capabilities of its components.

The Mindstorms NXT [11] from Lego is an educational, academic robot kit, ideal for beginners. The robot is equipped with drive motors with 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.

The e-puck [12] is an educational swarm platform for beginners. It has tiny dimensions with only 80mm of diameter, equipped with a vast set of sensors like microphones, infrared sensors, 3D accelerometer and a VGA camera. Similarly, the MarXbot [13] 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 533MHz. 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 been released (*ethzasl_aseba* stack[4]).

Additionally, the SRV-1 Blackfin [14] from Surveyor is a small-sized robot equipped with tracks with differential configuration. This robot has a good processing power with a 1000MIPS at 500MHz CPU, capable of running Linux Kernel 2.6. It is equipped with two IR rangers or optional ultrasonic ranging and a 1.3MP camera. It also supports Wireless 802.11b/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* [5] stack, which offers a driver for the Surveyor Vision System in ROS.

Among the larger, more equipped and more powerful mobile robots, a reference platform for research and education is the Pioneer 3 DX from ActivMedia [15]. 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*[6] and *p2os*[7].

B. Arduino-Based Robotic Platforms

Even though most of referred platforms provide open source software, they usually require a slow learning curve and the hardware has limited expandability. Arduino solutions have recently appeared in the market to work around such issues. For this reason, our platforms were built around an Arduino control board, 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* [8], the *Bot'n Roll OMNI* [8] and many others (*e.g.*, [1]).

The Arduino-based platforms under consideration, namely the TraxBot *v1* and *v2* and the Stingbot [3], are depicted in Fig. 2. 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[8].

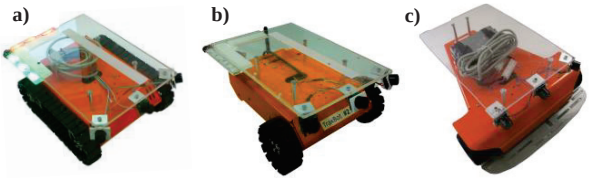

Fig. 2. Arduino-based robotic platforms,
a) TraxBot v1; **b)** TraxBot v2; **c)** StingBot.

As for power source, two packs of 12V 2300mAh Ni-MH batteries ensure good energy autonomy to the robots. For distance sensing, 3 Maxbotix Sonars MB1300 with a range of approximately 6 meters were used. However, and as experi-

[3] http://www.willowgarage.com/turtlebot
[4] http://www.ros.org/wiki/ethzasl_aseba

[5] https://github.com/rene0/ros-surveyor
[6] http://www.ros.org/wiki/ROSARIA
[7] http://www.ros.org/wiki/p2os
[8] http://botnroll.com/omni3md

mental results depict, the sensing capabilities of the platforms can be easily upgraded with other sensors, *e.g.*, laser range finders. Moreover, the platforms have the ability to also include a 10" netbook on top of an acrylic support, which extends the processing power and provides more flexibility. In our case, ASUS eeePC 1025C were used due to their reduced price and size. The notebook provides communication via Wireless Wi-Fi 802.11 b/g/n 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 2mW having a range between 40m to 120m, for indoor and outdoor operation, respectively.

C. Summary

Both Arduino-based platforms meet all the requirements previously pointed out, being ideal for multi-robot applications. In terms of cost, our platforms have a similar price to the Mindstorms NXT, being 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 hours, 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 is extended with WiFi when using a netbook. 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 26MhZ Atmega 328 microcontroller, but also the Intel Atom N2800 Dual Core at 1.86 GhZ processor of the netbook, similarly to the Turtlebot and outperforming the 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.

III. ROS Driver for Arduino-Based Robots

The key contributions of this work are the development and description of a driver that enables fast prototyping through the interface and control of custom educational platforms with ROS, which can be generalized to different Arduino-based platforms.

ROS provides tools to interface with the Arduino family of boards through the *rosserial* stack [9].However, it was verified that *rosserial* is not suitable for this work, due to the high overhead imposed by its data acquisition and commands, which result in an excessive workload to the Arduino microcontroller Atmel 328p SRAM. In fact, the microcontroller presents limited SRAM memory 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 PC / ROS and the Arduino, due to the structures used, for example, when publishing messages from the Arduino side. For this reason, a custom driver was created, being able to adopt a faster and more transparent communication between any Arduino board and ROS. We propose a solution based on the *serial_communication* stack [10], where the messages sent from the Arduino only consist of arrays of characters, which are parsed to integer variables on the PC / ROS side, hugely decreasing the communication load.

A. Driver Description

The *mrl_robots*[11] driver herein presented was developed for integration and control of the platform using ROS Fuerte version running on Ubuntu 11.10 "Oneiric Ocelot". The *serial_communication* stack [10], was used to establish point-to-point serial communication between the Arduino and the PC / ROS side, without the overhead of *rosserial.* This enables robust and fast communication in more complex applications, such as teleoperation, crossing of sensory information, the integration of the *navigation* stack, among others. It also has the versatility of creating 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. 3 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 communications. C/C++ language was used as the programming language for the ATmega328p microcontroller. Algorithm 1 illustrates the resident Robot/Arduino Firmware code.

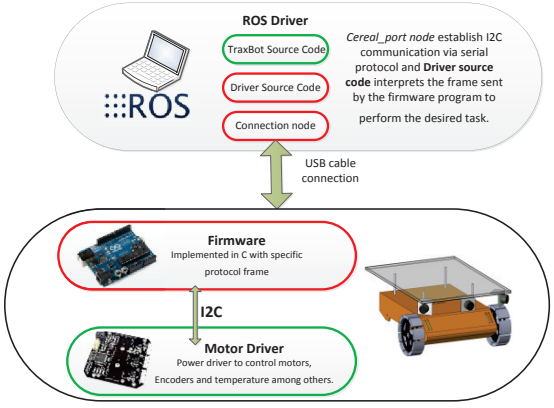

Fig. 3. ROS driver architecture diagram.

The protocol developed to interface ROS with the Arduino board consists on sending a frame with the configuration shown in Fig. 4. The character '@' is used at the beginning of every frame, and commas ',' separate the different parameters. Character 'e' identifies the end of the frame. Regarding the content 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.

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

[10] http://www.ros.org/wiki/cereal_port
[11] http://www.ros.org/wiki/mrl_robots

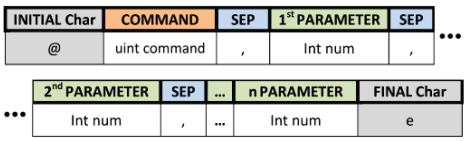

Fig. 4. Frame protocol to receive/send data from/to the Arduino Uno

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×10^3),(ang_vel×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 robot. Whenever a data frame has been received, 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. In Fig. 5, 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*[12], used for teleoperating the robot through a Nintendo's Wii remote controller (*WiiMote*), and the *hokuyo_node*[13] to add an Hokuyo laser range sensor to the robot. ROS 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 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.

B. 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 interface with ROS tools for data process and analysis of the platforms, like 3D visualization (*rviz)*, logging real-time robot experiments and playing them offline with (*rosbag/rxbag)*, plotting data (*rxplot)* and visualizing the entire ROS network structure (*rxgraph).*

Beyond the easiness of using the available tools, ROS also provides effortlessly integration of new sensors without needing hardware expertise, as it will be seen in section IV. 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*[14] stacks.

[12] http://www.ros.org/wiki/wiimote_node
[13] http://www.ros.org/wiki/hokuyo_node
[14] http://www.ros.org/wiki/slam_gmapping

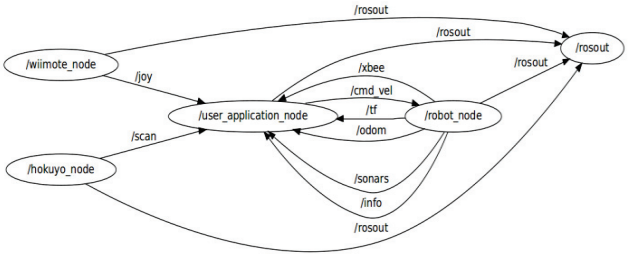

Fig. 5. Rxgraph topics and nodes provided by the *mrl_robots* driver.

Algorithm 1. Robot/Arduino Resident Firmware

```
#Omni3MD library  // main motor driver command functions
#EEPROM library    // storage robot particular specifications: robot ID,…
#Robot library   // range sonars acquisition, calibration, PID gains
#RobotSerialComm library   // protocol serial communication
#Standard libraries
Setup Functions(); // PID motor gains, using ports, encoders scale, set I2C connection,…
Streaming Functions():
    sendEncodersReads()
        Read encoder 1 and 2 pulses;
        Serial reply encoder data;
    sendEncodersSonarsReads()
        Read encoder 1 and 2 pulses;
        Read sonars 1, 2 and 3 ranges;
        Serial reply encoder and sonar data;
    sendRobotInfo()
        Read from EEPROM robot ID;
        Read internal board temperature;
        Read Omni-3MD driver firmware version;
        Read TraxBot battery voltage;
        Read firmware version;
        Serial reply info data;
Main loop():
    Switch (action):
        sendEncodersReads;
        sendEncodersSonarsReads;
        sendRobotInfo;
        Omni-3MD auto-calibration motors for controller purposes;
        Set PID gains;
        Receive Xbee message;
        Send Xbee message;
        Xbee node discovery;
        Set prescaler from encoders;
        Set desire encoders values;
        Robot info;
        Single encoders reading;
        Single sonars reading;
        Linear motors move with PID controller;
        Linear motors move;
        Stop motors;
        Reset encoders;
        Debug action;            // (Des)Activate debug option
        Start streaming data;   // Activate desire streaming data
        Stop streaming data;
```

Algorithm 2. PC/ROS Driver.

```
#ROS_msgs library    //  ROS type messages
#Cereal_port library //  protocol serial communication
Robot data callback():
    UpdateOdometry()
        Read encoder pulses;
        Pulses convert to cartesian pose (x, y, θ);
        Publish in ROS topic updated pose;
        Publish tf: odom → base_link
    DriveRobot()
        Subscribe ROS topic differential velocity commands;
        Send to robot angular and linear speeds;
    RangeUltrasonicSonars()
        Publish ROS topic range ultrasonic sonars;
    XbeeMsgs()
        Publish ROS topic with Xbee message received;
        Subscribe ROS topic with Xbee message to send from user node
    UpdateRobotInfo()
        Publish ROS topic robot information;
Main loop():
    Establish a serial connection;
    Receive data streaming from robot (activate callbacks);
```

As a result, the overall time spent in robotics research is greatly reduced due to code reuse and therefore the driver represents a valuable tool that enables fast prototyping and opens a gateway into the world of 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. 6, where an example of a ROS network is depicted.

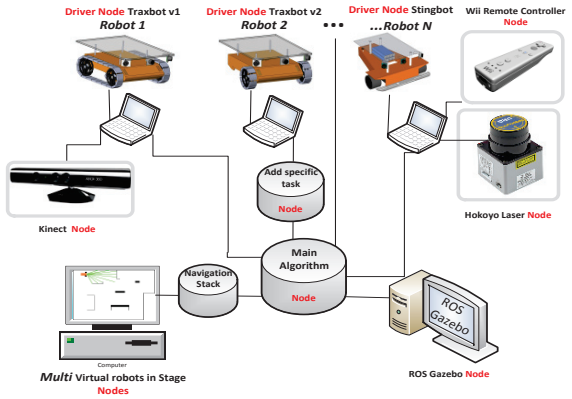

Fig. 6. Network topology example with multiple robots, sensors, tele-operation devices and applications.

ROS also has the potential to integrate mixed real and virtual robot teams. With the use of the driver herein presented, together with the Stage[15] multi-robot simulator, the same code can be used to drive either real robots or virtual agents running on Stage. Therefore, the developed ROS driver allows the integration of virtual robots with different sizes and driving characteristics, as seen later on. In addition, the communication between real and virtual agents is completely transparent since they are both operating in the same ROS network. 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.

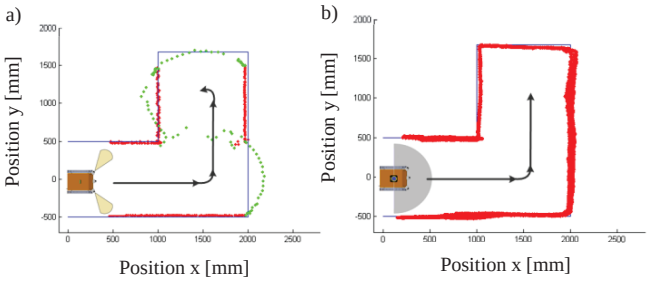

Fig. 7. Evaluation of the ROS driver in Traxbot *v1* with different sensors. **a)** Ultrassound Range Sensors integration; **b)** Hokuyo URG-04LX Laser Range Finder integration.

IV. Results and Discussion

In order to experimentally evaluate the ROS driver, some tests were conducted using physical Arduino-based robots and *stage*[15], which provides essential options like the information about the ground truth pose and odometry of virtual robots.

We present experimental tests that validate the aforementioned claims and we also show cooperative behaviors with real multi-robot systems, as well as mixed real and virtual robotic teams [16]. Therefore, the experiments will allow to evaluate the driver flexibility to different sensors, the driver portability to different robotic platforms and the driver expandability and integration with the existent ROS tools.

The first experiment aims to demonstrate the driver flexibility to integration of different sensors. The TraxBot *v1* platform was equipped with a laser range finder (LFR) and its performance was compared against the native ultrasonic range sonars on a simple mapping task. The Hokuyo URG-04LX is a LRF classified as an Amplitude Modulated Continuous Wave (AMCW) sensor. 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. 7). To perform this test, two lateral sonars placed at ±45 degrees were used.

In this test, the robot movement relies solely on odometry. In Fig. 7a 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 degrees rotation, since the sonar beam cone has an opening of approximately 36 degrees, thus presenting a much poorer resolution than the LRF, as illustrated in Fig. 7a. In the case of Fig. 7b, the Hokuyo LRF was used to perform the same test. The overture of the laser was set to 180 degrees with a rate of 512 samples per reading. It is possible to observe some discrepancy in some readings especially at the end of the movement due the odometry position error accumulated during motion.

In the second experiment, the main goal is to demonstrate the portability of the driver to different robots and sensors, which enables testing in our Arduino-based robots the existent algorithms in ROS. Hence, a mapping task with the incorporation and interaction of drivers for different sensors like LRF and a joystick controller was performed. Along these lines, this time the Traxbot *v2* platform was equipped with an Hokuyo LRF and teleoperated with a *WiiMote* for a mapping task using Hector Mapping [16], which is available in the *hector_slam* stack[17].

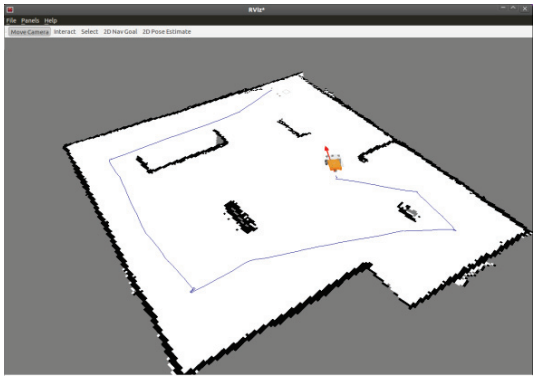

Fig. 8. Map generated by the Traxbot *v2* with hector mapping.

[15] http://www.ros.org/wiki/stage

[16] **A video of the experiments is available at:** http://www.isr.uc.pt/~aaraujo/videos/Robotica2013

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

The teleoperation node runs on the eeePC netbook connected over the ROS network. The node subscribes to a topic which has the information of the *Wiimote* state and assigns functions for each pressed button, publishing then velocity commands, which are interpreted by the ROS driver, resulting then on motor commands. 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. Fig. 8 presents the resulting map in *rviz*, which was obtained with *hector_mapping* on our experimental lab arena.

In the third and final 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 which communicates with the other two forming a mixed team of real and virtual robots, 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 our experimental lab arena (Fig.9).

The robots were commanded to navigate cyclically between a set of waypoints in the arena, as seen in the video of the experiment[16]. To further demonstrate their coordinating abilities, a common waypoint for all three robots was defined, and robots had to exchange messages through a shared ROS topic to avoid going to the common point at the same time. They would wait to go to the point at the center of the arena and priority was given to the robot who expressed firstly its intention to move to that point. All three robots were able to coordinate themselves in the environment without colliding to each other, due to the integration of the *navigation* stack[2]. Fig.10 presents a snapshot of rviz, illustrating the three robots moving in the arena.

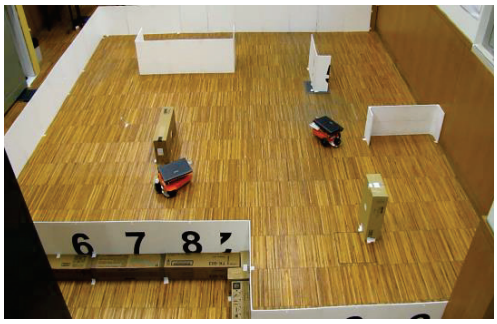

Fig. 9. Experimental arena with a Traxbot *v2* and a Stingbot cooperating with a virtual robot, running on stage.

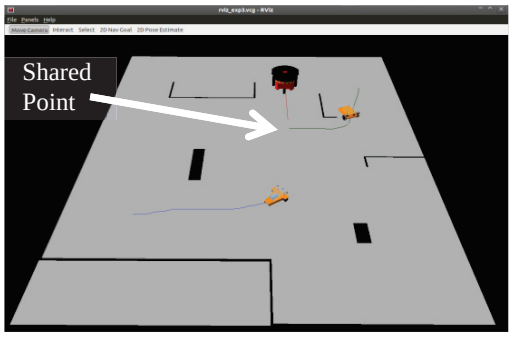

Fig. 10. The three robots coordinating their behaviors by exchanging ROS messages (rviz).

V. Conclusions and Future Work

In this paper a solution for integrating Arduino-based robotic platforms in ROS, through the development of a ROS driver, was presented. It was shown the great advantages of integrating the platform with ROS middleware, enabling the usage of a wide range of tools and reducing the development time through code reuse. 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 real and virtual agents, running the same code. Results from the experiments that were conducted demonstrate all these features and the insignificant overhead imposed by the driver was discussed.

References

[1] Brooks, Rodney A., “New Approaches to Robotics”, Science, vol. 253, pp. 1227- 1232, 13 September, 1991.
[2] John-David Warren, Josh Adams and Harald Molle “Arduino Robotics”, Springer Science and Business Media, 2011.
[3] A. Araújo, D. Portugal, M. Couceiro, C. Figueiredo and R. Rocha, "TraxBot: Assembling and Programming of a Mobile Robotic Platform". In Proc. of the 4th International Conference on Agents and Artificial Intelligence (ICAART 2012), Vilamoura, Portugal, Feb 6-8, 2012.
[4] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in Proc. Open-Source Software workshop of the International Conference on Robotics and Automation, Kobe, Japan, May, 2009.
[5] B. Gerkey, R. Vaughan and A. Howard, "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, Portugal, 2003.
[6] G. Grisetti, C. Stachniss and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters", In IEEE Transactions on Robotics, 2006.
[7] R. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)", In Proc. of International Conference on Robotics and Automation (ICRA 2011), Shanghai, China, May 2011.
[8] M. S. Couceiro, C. M. Figueiredo, J. M. Luz, N.M. F. Ferreira & R.P. Rocha. "A Low-Cost Educational Platform for Swarm Robotics", Int. Journal of Robots, Education and Art, Vol. 2(1), Feb., pp. 1-15, 2012.
[9] Park, I. W. and Kim, J. O., “Philosophy and Strategy of Minimalism-based User Created Robots (UCRs) for Educational Robotics - Education, Technology and Business Viewpoint”, International Journal of Robots, Education and Art, vol. 1, no. 1, 2011.
[10] M. Kuipers, “Localization with the iRobot Create”. In Proceedings of the 47th Annual Southeast Regional Conference ACM (ACM-SE 47), Clemson, South Carolina, USA, March 19-21, 2009.
[11] B. Bagnall, “Maximum LEGO NXT: Building Robots with Java Brains”. Variant Press, 2007.
[12] 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), pp. 59-65 2009.
[13] Bonani, et al, “The MarXbot, “a Miniature Mobile Robot Opening new Perspectives for the Collective-Robotic Research” In Int. Conf. on Intelligent Robots and Systems, Oct. 18-22, 2010, Taipei, Taiwan, 2010.
[14] Cummins J., Azhar, M.Q. and Sklar, E.: Using Surveyor SRV-1 Robots to Motivate CS1 Students. In: Proceedings of the AAAI 2008 Artificial Intelligence Education Colloquium, 2008.
[15] S. Zaman, W. Slany and G. Steinbauer, "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, Saudi-Arabia, 2011.
[16] S. Kohlbrecher, J. Meyer, O. von Stryk and U. Klingauf, "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), 50-55, Kyoto, Japan, Nov. 1-5, 2011.