# ros4mat: A Matlab Programming Interface for Remote Operations of ROS-based Robotic Devices in an Educational Context

Yannick Hold-Geoffroy, Marc-André Gardner, and Christian Gagné
*Laboratoire de vision et systèmes numériques*
*Département de génie électrique et de génie informatique*
*Université Laval, Québec (Québec), Canada G1V 0A6*
{*yannick.hold-geoffroy.1, marc-andre.gardner.1*}*@ulaval.ca*
*christian.gagne@gel.ulaval.ca*

Maxime Latulippe and Philippe Giguère
*Département d'informatique et de génie logiciel*
*Université Laval, Québec (Québec), Canada G1V 0A6*
*maxime.latulippe.1@ulaval.ca*
*philippe.giguere@ift.ulaval.ca*

***Abstract*—More and more, robotics is perceived in education as being an excellent way to promote higher quality learning among students, by grounding theoretical concepts into reality. In order to maximize the learning throughput, the focus of any robotics software platform should be on ease of use, with little time spent integrating the components together. To this effect, we introduce ros4mat, an open source library which provides a simple and flexible interface between ROS (Robot Operating System) and Matlab®. The conception is focused on academic use, and allows a very simple integration of sensors and actuators to existing Matlab code. The library is designed to provide an easy, platform-independent, and fast connection between a robot (running ROS) and multiple clients (running only Matlab). Moreover, it is very versatile and can be used with many common types of sensors in robotics, including low-cost ones. We report the results of ros4mat use in a robotics course to provide more real world experimentation, along with some code samples illustrating the simplicity of our approach.**

***Keywords*-Robot programming; Robot control; Mobile robots; Educational robots; Engineering education; Matlab.**

## I. Introduction

Using robots in engineering and computer science curricula is recognized as an excellent way to stimulate curiosity and motivation among students [1], to provide an unique learning experience [2], while linking theory to practice. It has even been proposed by some that robotics may lead to a new revolution in the way computer science will be taught [3]. To achieve this, a simple and intuitive tools for controlling robotic devices is desirable, hence reducing the learning curve required to achieve practical validation of theoretical educational content.

In the current paper, we a presenting ros4mat, an open source robotic library that allows the integration of ROS (Robot Operating System) [4] into the Matlab [5] environment. This library has been designed to allow students of robotics courses at an undergraduate (and graduate) level to realize practical experiments with real robotic platforms very easily, requiring no more than basic Matlab knowledge.

The structure of the paper goes as follows. First, the motivations of this project are explained in Sec. II. Sec. III then makes a brief review of similar software and integrated robotic platforms currently available. A summary of design objectives is presented in Sec. IV, followed by explanations on the implementation and its uses (Sec. V) and some experimentations (Sec. VI).

## II. Motivations

The main motivation of the project is to propose an easy way to integrate inexpensive robotic platforms into tools commonly used in academia. Consumer robotics are now more and more widespread thanks to the sustained progresses of consumer electronics. Cheaper and better components are accessible at a continuing pace including sensors, processors, and mechanical devices [3]. However, there is still a lack of inexpensive and flexible mobile robotic platforms [6]. In fact, the robotic platforms currently available that aimed mostly at research or education are either complicated, expensive, or poorly supported by the tools commonly available in academia. We are stressing here the need for inexpensive alternatives, as it allows the purchase of many units for teaching purposes while decreasing the cost of research units.

Developing an extensible and generic library improves the usability of the system. Using the same software to control similar sensors saves integration time. For example, the use of inexpensive hardware for teaching, while using more accurate sensors for research purposes, should not require software rewrite. Moreover, reusable software lessen the need for re-engineering when newer versions come along or a new generation of sensors is used. Furthermore, being compatible and using as much off-the-shelf components as possible ensures easy ordering, replacement, and repair parts in the event of failures or in the need to build more platforms.

Manipulating data from multiple heterogeneous sensors is a complex task often left to the user. Keeping data coherent, ordered, and synchronised between sensors is delicate to achieve and prone to error. Therefore, we are proposing that it is highly desirable to follow an approach that keeps data retrieval and manipulation as simple as possible. This can be achieved by making sensor acquisition, timestamps, and communication transparent to the user, while supplying data in an intuitively shaped and ordered way.

Moreover, autonomous robots have a limited energy budget. Running on strict power consumption rules keeps computing power fairly low. Developing lightweight software is an important matter for onboard robot execution.

In an educational context, the ease of installation is also crucial: the students should be able to quickly perform the installation steps, no matter which OS or which software they use. Besides, as some experiments might have to be performed on public workstations, the installation should not require administrator or root access to the computer.

The last aspect of our considerations is licensing. Access to the source code and the documentation of the project is important for learning and improvement purposes.

## III. Related work

### A. Generic ROS integration in Matlab

Linking ROS and Matlab generated keen interest over the last year. A project has been developed at Penn Robotics using an Inter-Process Communication (IPC) bridge [9]. This solution has some rigid requirements restraining its use to local libraries or programs built for the same architecture. Moreover, the Matlab interpreter is limited to Linux as it should be run on the same machine as ROS. However, its integration with Matlab adheres to the ROS philosophy by implementing the wrapper as a generic communication node. This method optimizes interface coherence and minimizes the amount of protocol and code duplication. On the other hand, data in the Matlab interpreter is not supplied in an efficient way considering a matrix oriented language. Data is read one sample at a time requiring the user to build the matrices on his own.

Other projects are proposed in ROS discussion boards to implement Matlab communication. There is a Special Interest Groups in the ROS community that currently works on the integration of Matlab in ROS [10]. Most of the proposed solutions are based on IPC communication, which bears the same limitations as the previously mentioned project.

Another project similar to ROS called Player provides Matlab integration of robotic devices. However, its Matlab client library is unmaintained since 2008. Furthermore, while it allows remote connection to robotic devices, it requires the installation of the playerc library on the client computer which is more intrusive than our approach and requires administrative rights, not always available in laboratories.

### B. Specific robotic platform integration in Matlab

Kits available for entertainment, such as the Lego's Mindstorms or Robix's Rascal, are interesting for specific actuator or sensor uses. Advanced usage of the Mindstorms NXT allows Matlab integration using a third party toolbox [11]. This toolbox is limited to the Mindstorms NXT and is not working with other robotic devices.

Multiple robot kits such as the TurtleBot [12], the Bilibot [13], the Eddiebot [14], and the likes are built using off-the-shelf components and are designed to accept a small form factor computer. These inexpensive kits are ideal for educational purposes and are supported by ROS. However, no Matlab integration is currently available for these kind of platforms.

Most high-end robotic platforms provide some tool integration to different extent. For instance, the e-puck [15] is a miniature robot aimed at education at university level developed with an open hardware platform and costs around 1000$ CAD per unit. Aside from its relatively high price, it will not be physically able to support large sensors such as the Kinect. It comes with a Matlab framework to control and receive feedback from the unit using a graphical interface or functions usable from the Matlab command line. Another example is the Corobot [16] (priced at between 4000$ and 11 000$) which is compatible with the RoboRealm API [17], usable with most languages including Matlab.

## IV. Specifications

We consider Matlab as the lowest common denominator in science and engineering academia, mainly due to its easiness of data representation, programming capabilities, and simplicity of its syntax based on linear algebra. The use of the Matlab environment is widespread in various disciplines such as computer science and engineering, but also in electrical and mechanical engineering, which also have direct connections to robotics. This language is based on the matrix notation, and readily applies to robotics concepts and algorithms. Indeed, most fundamental operations computed by robots involve matrix equations to be solved. Writing matrix equations directly in the development language keeps the code clean and similar to the scientific literature, thereby easing the task of the practitioner.

ROS is now the *de facto* open source platform used in robotics [4]. Initially developed in 2007 at the Stanford Artificial Intelligence Laboratory, this framework provides libraries and tools under the BSD license that allows communication schemes and modular integration of hardware. In the ROS nomenclature, every process, such as a device driver producing data or gathering it, is called a *node*. Each node can communicate with other nodes using a specific data structure called a *message*. Messages are distributed using the Publisher/Subscriber pattern [7] where a process can subscribe to one or many publishers to receive the published data. Nodes emitting data are called publishers while those gathering this data are subscribers. This pattern offers several advantages over standard point-to-point connections for data distribution, such as loose coupling and high scalability.

By linking these two software libraries together, namely the power of the Matlab programming environment and the hardware abstraction packages of ROS, it allows for easier and faster deployment of code over a working unit. The Linux kernel used by ROS grants the project a vast driver availability, and allows the flexibility of deploying live bootable USB flash drives to users.

A major limitation of personal computers being used as robot controllers is their lack of I/O interfaces [8]. This is an issue addressed by software platforms such as ROS and Microsoft Robotic Studio. Given a small low-cost appliance with enough I/O running ROS or alike, one can use the Matlab interpreter on its own computer for monitoring and issuing commands to the wireless robot. This also eliminates the ROS requirements on the client Matlab system, allowing the use of unsupported operating systems or configurations.

## V. Implementation

### A. General architecture

The proposed ros4mat library is based on a client/server model. It is freely available[1] and licensed under the LGPL. The server part (running on the robot) is built on ROS, while the client part is a standalone Matlab extension (using the MEX file interface). The communication between servers and clients is based on standard TCP sockets.

This architecture choice enforces the principles outlined in the motivations section. First, it makes ros4mat completely network transparent: the Matlab client can run on the embedded computer itself, or on a remote control station. This is especially interesting in a mobile robot context, since the embedded intelligence is often limited by the power requirements or available memory on the robot's computer. Moreover, by running the intelligent part on a remote workstation, the user can comfortably debug the system as it runs, having access to the full visualization power of Matlab. Besides its remote capabilities, ros4mat is also suitable for applications with low latency or high bandwidth requirements (like a stereo camera system), since the Matlab client can also be run locally.

Second, this architecture does not require the client part to have a functional ROS installation. This is an important feature, since it may sometimes be hard to install ROS on non-Linux platforms (e.g. Windows). As ros4mat only requires the compilation of a standalone MEX file, it can be used on virtually any platform supported by Matlab.

Finally, this approach allows multiple simultaneous client connections on a single robot. Of course, a given sensor can only have one configuration at the time, but the gathered data may be forwarded to more than a single Matlab client. For instance, this can be used to monitor a robotic system or to split the workload between two remote stations. In a teaching context, it may also be useful to allow all students to connect to the same robot, thus ensuring that they all get the same live data of a given experiment.

[1]https://ros4mat.googlecode.com

### B. Server part

The server part (which runs on the robot) is a standard ROS node, apart from the fact that it is able to send information remotely. It automatically manages topic subscriptions, sensors configuration, synchronization, and data packing for the remote client. For most applications, this node does not actually need any specific configuration or hint from the user: it starts and waits for a connection from a Matlab client.

This node is written in C++ and is fully compliant with ROS specifications. It can be compiled using the *rosmake* utility available in ROS and, whenever possible, uses standard ROS messages. This adds versatility and stability to the library, since most of the acquisition work is done by other well-tested ROS nodes. For instance, the camera capture makes use of the *uvc_camera* node[2], thus making it compatible with all UVC cameras.

The ros4mat node converts the information from the ROS topics to our transmission protocol. This protocol is platform-independent (both for OS and CPU architecture) and was designed to minimize the overhead on the transmission latency and time. Depending on the application, the user can also activate data compression, performed by a lossless compression algorithm. This leads to an higher CPU workload, but it may greatly minimize the transmission time and bandwidth consumed.

Currently, the ros4mat node integrates eight sensor types among the most common in robotics. These sensor categories are:

- Inertial Measurement Unit (IMU);
- Global Positioning System (GPS);
- Single and stereo-paired RGB camera;
- Laser rangefinder (Hokuyo);
- Kinect and Kinect-like device;
- Analog input on multiple channels;
- Battery information;
- System information (CPU load, memory consumption, I/O load, computer temperature, etc.).

In addition to that, ros4mat also allows the remote control of a serial port (read and write). This may be used to interface other non-standard sensors which use RS-232 as communication protocol or to connect to some external devices. For instance, we used it to control an iRobot Create robotic base: we only needed to adapt the communication function of a given toolkit to ensure that it uses the ros4mat virtual serial port instead of an actual serial port interface.

### C. Matlab client

The Matlab client is implemented in a single C file using the standard MEX file format to efficiently interface with Matlab. It is written in ANSI C (a.k.a. C89) and uses standard BSD socket functions when available to ensure portability. Winsock 2 is used on Windows platforms. It

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

has been successfully compiled and tested on Linux using GCC, and on Windows (both 32 and 64 bits) using MinGW, VCC (the Microsoft Visual Studio Compiler), and LCC (the embedded Matlab compiler).

This client is mostly a wrapper around the communication structures used in ros4mat. It handles the data to ensure that it complies with Matlab data formats, for example by transposing the matrix structures, as Matlab uses a column-major order while ROS uses a row-major order.

### D. Communication protocol

The communication protocol is based on a simple pull approach. The main objective is to clearly separate the fast, real-time processing needed for data acquisition on the robot and the processing part on Matlab. The server part only sends data to a Matlab client on request, thus ensuring that a client will never get filled with unrequested data and minimizing network congestion. On the server part, a circular buffer ensures that the transmitted data always contain the most recently obtained. The size of the circular buffer defaults to 1 s of data for the relevant sensor, but can be easily modified by the user. This approach is consistent with practical considerations in mobile robotics, where it is often more important to get the most recent data quickly rather than being forced to process the older data first. For instance, if a sensor is used for collision avoidance, one may use a small buffer size, as older data is then not very interesting. The use of this circular buffer is also important in decoupling the fast-polling required for data acquisition on the robot from the slower “sense-think-act” loop running on the Matlab client.

A typical ros4mat session begins by a client connection to an already started server:

```
ros4mat('connect', 'IP_ADDRESS'); % For instance 127.0.0.1
```

Thereafter, one may *subscribe* to a given sensor and set its configuration parameters:

```
ros4mat('subscribe', 'sensor_name', params ...);
```

At this point, the server part will configure the appropriate node and begin to listen to the data feed from the sensor. The data will accumulate in a circular buffer and be paired with their acquisition timestamps. Subsequently, the data can be retrieved into Matlab via the client:

```
[data, timestamps] = ros4mat('sensor_name');
```

where the data format is dependent on the sensor used, and the second return value contains a list of global timestamps corresponding to the acquisition time of each measurement. There is no guarantee on the absolute accuracy of timestamps – they are based upon the internal clock of the ROS system, which may not be accurate – but the relative precision between two timestamps of two different sensors has been verified empirically to be less than 10 ms.

The number of returned samples is guaranteed to be no more than the size of the circular buffer, but can be smaller (or even zero) if there is not enough data. The returned samples are removed from the circular buffer by the server, so there are no duplicates between subsequent calls. However, the circular buffers are specific to each client, so that the read operation does not modify the data available for another.

When the data produced by a sensor is not needed anymore, an optional *unsubscription* method can be called to reduce the resources used on the server part:

```
ros4mat('unsubscribe', 'sensor_name');
```

It is possible to subscribe to more than one sensor at a time. In this case, the returned timestamps are also synchronized between the different sensors which use independent circular buffers. This way, the user never has to worry about synchronisation, when performing sensor fusion. The ros4mat function accepts some other parameters, notably in the specific cases of bidirectional communication like the use of the ROS system serial port.

### E. Extending ros4mat

The ros4mat library was designed for straightforward sensor addition. In order to do so, one must first define a C structure which will be used to wrap up the sensor data so it can be sent on the network. For instance, the GPS data structure is defined as:

```
struct msgGps{
    char state;
     float latitude;
     float longitude;
     float altitude;
     float speedModule;
     float speedAngle;
     float verticalSpeed;
    double timestamp;
};
```

Thereafter, a function must be added to the ros4mat ROS node in order to handle this new data type – that is, read the corresponding topic and insert it into the circular buffer. Similarly, a handle function must be added in the Matlab client to convert back this data structure to a Matlab-compatible representation.

## VI. Experimentation

### A. Code examples

The following Matlab code excerpt exhibits sensor subscription and data retrieval using the Analog to Digital Converter (ADC):

```
% Add ros4mat to Matlab path.
addpath('./ros4mat/')

% Robot parameters
IP = '10.10.10.2'; % Robot IP
adc_freq = 100; % ADC Acquisition frequency
adc_channels = 1; % Subscribe to channel 1
 polling_freq = 10; % Communicate data at 10 Hz
 buffer_length = 0.5; % 0.5 second buffer

% Connect to the robot
```

```
ros4mat('connect', IP);
ros4mat('subscribe', 'adc', adc_freq, adc_channels, ...
        buffer_length * adc_freq, polling_freq );
pause(1)

while (true)
    % Retrieve ADC data and timestamps
    [adc, ts] = ros4mat('adc');
    % Extract the first channel of the ADC
    DataCH1 = adc(1,:);

    % Perform processing, wait for data and loop
    pause(1)
end
```

This example shows that with about 25 lines of code (including variables definitions and comments) one can make a program which:

- connects itself to a remote ros4mat server;
- configures an ADC with specific parameters; and
- retrieves the data into native Matlab structures.

Another interesting example is the combined use of two sensors. In this case, we have a standard Sharp infrared proximity sensor GP2Y0A02YK0F and a LPR510AL gyroscope, connected to two different analog inputs. Those sensors are placed onto a robot base which rotates at a constant rate. The objective is to create a map of the environment, by synchronizing the data from both sensors. This leads to the following code:

```
% Connect to the server
ros4mat('connect', '10.0.0.2');

% Select the ADC, 100 Hz acquisition rate, channels 1 and 6
% and 2000 samples buffer (i.e. 20 seconds of data)
ros4mat('subscribe', 'adc', 100, 33, 2000);

% We suppose here that the robot is turning around
pause(20);

% Get the data from the ADC
[data, ts] = ros4mat('adc');

% Compute the map
radSensor = 12; % Distance from the rotation center to the sensor
dt = 0.01; % 100 Hz sampling
ZeroLevel = 1.2120;  % zero level of the gyro
DegRotByVolt = 1/0.00944;
Map{1}(1,:) = cumsum((data(6,:)- ZeroLevel).*DegRotByVolt)*dt;
% We use precalibrated Volt - distance paired values
Map{1}(2,:) = interp1( calibInfraVolt , calibInfraDist , data (1,:));

polar(Map{1}(1,:) ./ 57.3, Map{1}(2,:));
% We could also convert the polar coordinates with pol2cart
% and display them in a cartesian plot
```

The output figure produced is presented in Fig. 1. There is inevitably some noise on the produced map, due to the imprecise nature of the sensor and the analog-to-digital converter, but it shows that significant information about the environment can be gathered easily with our platform and these low-cost sensors.

As a final example showing the promising possibilities of ros4mat, Fig 2 shows the graphical interface of a complete implementation of an ICP mapper in Matlab. It used a Hokuyo laser range finder, and the displays and buttons (not shown on screen) allowed for an interactive creation of the map on a remote client by the students. Here, ros4mat allowed the use of sophisticated algorithms on real-time data, even if less than 10 lines were actually used in Matlab to obtain these data.

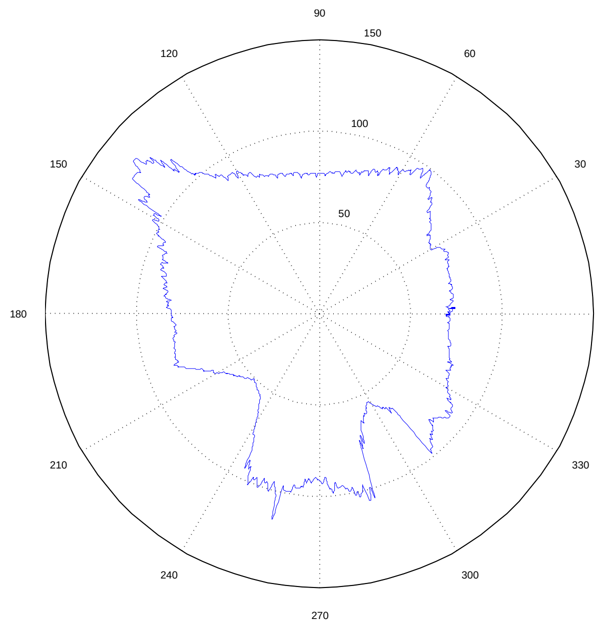

Figure 1. Sample output of the polar map creator, using a Sharp GP2Y0A02YK0F infrared sensor and a LPR510AL analog gyroscope.

## *B. Course usage*

The proposed platform was used as a teaching aid in an introduction to mobile robotics course at Laval University. Ten robots were quickly assembled using an iRobot Create as mobile base and a custom designed Lexan$^{TM}$ structure for placing sensors as well as a laptop running the server interface of ros4mat. Each robot was equipped with a camera, two infrared sensors, and a two-axis gyroscope. With such a configuration, the total cost of an individual robot, excluding the laptop which was supplied by the students, was of about 500 dollars. Some Hokuyo laser range finders were also available as extra sensors, but are not included in the cost figure. Figure 3 shows one of these robots. Note that the design of those robots is not linked in any way to the ros4mat framework: it is simply a cheap, simple design used in a particular course. In fact, any kind of robot ROS-compatible may be used with the ros4mat framework (including various widely available robots like the Turtlebot).

The robots were used in a series of practical laboratories, where the students could experiment with concepts such as:

- infrared sensor measurement (non-linear model) and behaviour;
- gyroscope calibration and use;
- camera-based localization;

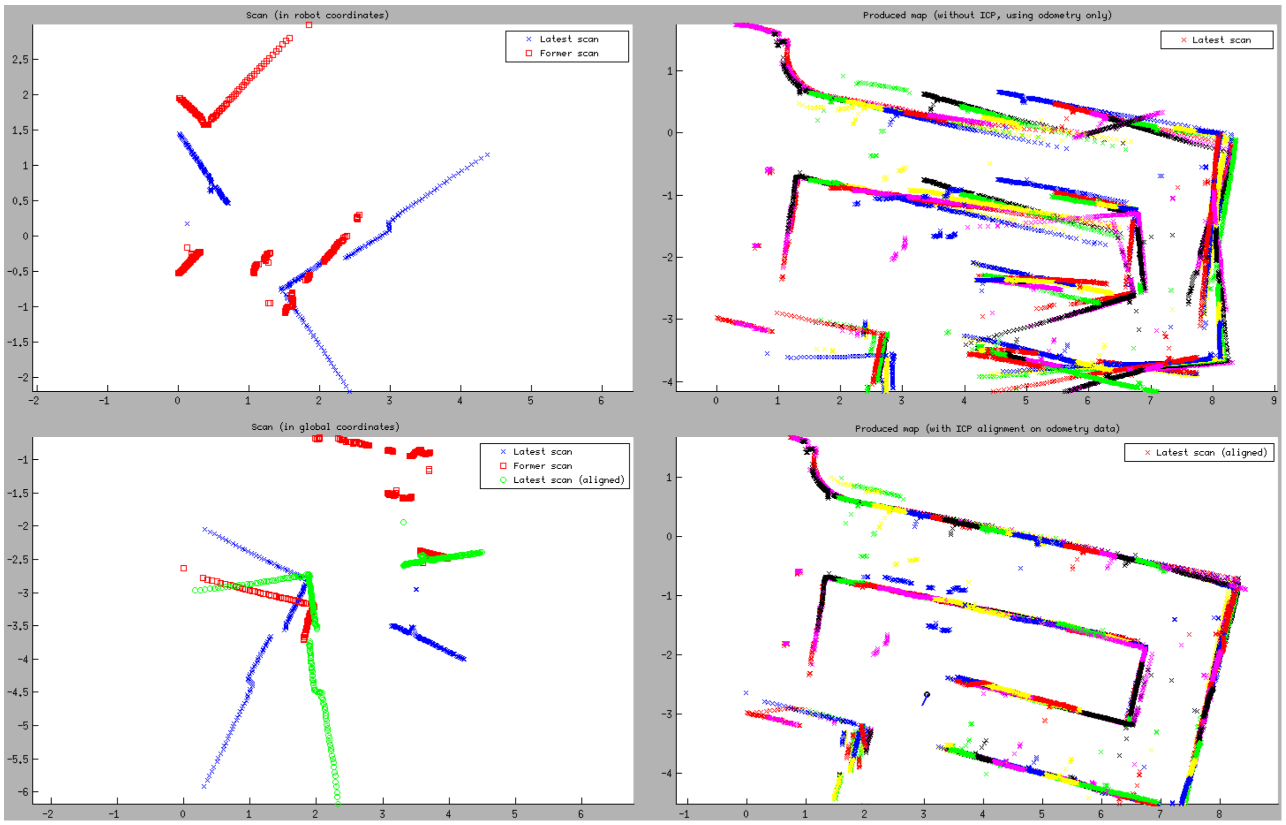

Figure 2. Sample output of an ICP interactive program using a iRobot Create base and a Hokuyo laser rangefinder. The lower right figure is the ICP output, while the upper right figure is the map created without ICP, using only the robot position sensors. Students could control the motion of the robot (straight, turn, scan and align) via buttons on the GUI (not shown).

- robot's motion model estimation;
- feedback control loops;
- Kalman filters; and
- point clouds registration using Iterative Closest Point.

Furthermore, as a final evaluation, students were asked to elaborate a project of their own using these robots and the ros4mat library.

To accomplish the experiments, the students were given a bootable image of the ros4mat server, which they could easily run from a USB stick on their personal laptops. Any computer with Matlab installed could then be used to connect to the server, control the robot, and process the sensor data. Each student team having its own material, about 20 different setups were consequently tested. The ros4mat interface has shown to work well on all of them.

The ease of use of the proposed robotic platform allowed the students to link theory to practice, while avoiding technical implementation and integration difficulties, therefore getting them to focus more on concepts than technical intricacies. As an evaluation metric of this platform's utility for teaching, the general appreciation of the students, in terms of learning quality, increased from a steady 90% in the past years to 94% using the robots, as measured by their course evaluation responses. Moreover, course registration doubled, going from an average of 20 students in previous years to 40. Finally, the demographic of enrolled students (Table I), in terms of their study programs, justified ros4mat's requirement for limited programming needs. Indeed, not all of them were familiar with languages such as Python or C++.

Table I
PROGRAM OF ENROLLED STUDENTS.

| Program | number of students |
|---|---|
| Computer Science | 23 |
| Software Engineering | 6 |
| Computer Engineering | 6 |
| Electrical Engineering | 2 |
| Mechanical Engineering | 3 |

## VII. Conclusion

In this paper, we proposed ros4mat, a generic, easy-to-use library that aims at facilitating the integration of robotic platforms into classical robotic courses. By handling the complex integration and synchronization of real-world sensors into a well-known mathematical programming environment, it allows the user to focus on his algorithms and their mathematical background, making it a valuable tool for education and training. Moreover, the client part can be run remotely, and does not require any third-party library.

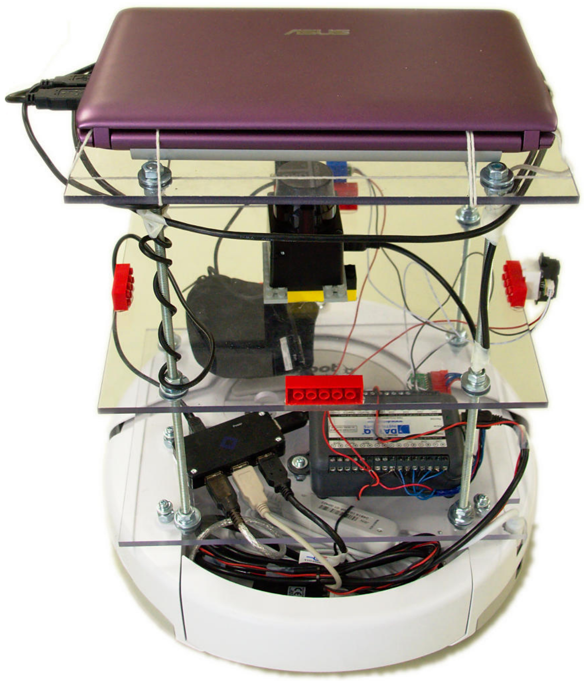

Figure 3. The robot produced in ten copies for the course usage, featuring an 8-channel ADC, a laser rangefinder, two infrared sensors, and a motorized Roomba base. The upper plate was left clear such that students could strap in their own laptop.

Multiple clients can also be connected to a single server, effectively fetching the same data without the need of manual synchronization. The hardware architectures supported correspond to those of ROS, which covers a wide range of components, sufficient for most current uses in mobile robotics, and can be extended to fit specific needs with little work. The ros4mat library has already been tested with great results and positive feedback in a computer science course, for both laboratory assignments and student projects.

Overall, ros4mat may be viewed as an assembly of good building blocks. Nevertheless, it is much more than a simple packaging of different tools: this assembly is done so it highlights the good sides of each block, but hides the complex and undesirable parts, effectively resulting in a library as simple as versatile, which may be of great resource into various domains.

## Acknowledgement

The authors would like to thank David Dastous-St-Hilaire and Youssef Mabrouk for their help and advices in the library preliminary conception.

## References

[1] R. V. Aroca, R. B. Gomes, D. M. Tavares, A. A. S. Souza, A. M. F. Burlamaqui, G. A. P. Caurin, and L. M. G. Goncalves, "Increasing students' interest with low-cost cellbots," *IEEE trans. on Education*, vol. 56, no. 1, pp. 3–8, feb. 2013.

[2] J. Weinberg and X. Yu, "Robotics in education: Low-cost platforms for teaching integrated systems," *IEEE Robotics Automation Magazine*, vol. 10, no. 2, pp. 4–6, june 2003.

[3] D. S. Touretzky, "Preparing computer science students for the robotics revolution," *Communications of the ACM*, vol. 53, no. 8, pp. 27–29, 2010.

[4] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[5] The Mathworks Inc., "MATLAB: The language of technical computing," 2013. [Online]. Available: http://www.mathworks.com/products/matlab/

[6] J. Lumsden and C. Ortega-Sanchez, "Modular autonomous robotics platform for educational use," in *IEEE Region 10 Conference (TENCON)*, nov. 2010, pp. 1577 –1582.

[7] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, Jun. 2003. [Online]. Available: http://doi.acm.org/10.1145/857076.857078

[8] J. Hamblen and T. Hall, "Engaging undergraduate students with robotic design projects," in *Proc. of the IEEE International Conference on Field-Programmable Technology*, jan. 2004, pp. 140 – 145.

[9] B. Cohen and P. R. Nathan Michael, "Matlab & ros," Jul. 2011. [Online]. Available: https://alliance.seas.upenn.edu/~meam620/wiki/index.php?n=Roslab.IpcBridge

[10] G. C. Martin Riedel and A. Franchi, "Ros on matlab," Jun. 2012. [Online]. Available: http://www.ros.org/wiki/groovy/Planning/Matlab

[11] A. Behrens, L. Atorf, R. Schwann, J. Ballé, T. Herold, and A. Telle, "Practicing engineering in a first-year student project: Matlab meets lego mindstorms," in *Proc. of the 12th International Student Conference on Electrical Engineering*, Prague, 2008.

[12] "The turtlebot website." [Online]. Available: http://turtlebot.com

[13] "Bilibot project." [Online]. Available: http://www.bilibot.com

[14] "The parallax eddiebot website." [Online]. Available: http://www.parallax.com/eddie

[15] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a robot designed for education in engineering," in *Proc. of the 9th Conference on Autonomous Robot Systems and Competitions*, 2009, pp. 59–65.

[16] "The corobot website." [Online]. Available: http://robotics.coroware.com

[17] "Roborealm website." [Online]. Available: http://www.roborealm.com