U.P.B. Sci. Bull., Series C, Vol. 80, Iss. 3, 2018 ISSN 2286-3540

INTEGRATING A V-REP SIMULATED MOBILE ROBOT INTO ROS

Andrei George Florea[1]

Robotics courses usually require students to implement and test various algorithms on real mobile robots in order to observe minute technical details that influence the behavior of the robot. However, most robots have a high price tag that prevents students from owning a robot and testing algorithms at home. An extension for the V-REP robotics simulator is presented in this paper. This extension is used to integrate multiple simulated educational robots (e-puck) into the Robot Operating System (ROS) with the aim of offering students a free experimentation platform that uses standard models and conventions. The accuracy and flexibility of the extension are discussed in this paper, along with two use case scenarios.

Keywords: simulation, V-REP, ROS, e-puck

1. Introduction

During robotics courses, there are several aspects that are best understood when studied on real robots. For example, understanding the need for a proportional controller for controlling the robot speed requires that the students note movement errors that are accumulated over time.

As even educational robots have a high price tag, robot experiments can only be done in class. Computer simulations are a solution, provided that the simulator has a low price and reasonable accuracy.

The differences in architecture between robots and computers are usually noticeable also at a software level. This results in less code portability between real robots and computer simulations and is error prone. A robotics development framework, such as the Robot Operating System (ROS), can provide a solution. ROS consists of a set of high level software packages that allow for the development of general purpose behaviours that are portable from one platform to another [10, p. 3]

These aspects present different sides of a more complex problem, that of presenting robotics course material, without loosing contact with the physical details. This paper attempts to address all issues by making a compromise between cost effectiveness, utility, portability and accessibility.

[1] Teaching Assistant, Faculty of Automatics and Computer Science, University "Politehnica" of Bucharest, Romania, `andrei.florea@acse.pub.ro`

In essence, the proposed solution is an extension for a free robotics simulator (V-REP) that allows one or more educational robots (e-puck) to be interfaced with a standardized robot software framework (ROS), thus allowing the use of high level code for the control of both simulated and real robots, without modification. The extension is free and open-source (available at [3]) as is the robot software framework while the robotics simulator is free for academic/hobbyist use.

The paper continues with a short overview of the hardware and software resources that were used, followed by a short presentation of the design aspects that were considered for the extension. In the fourth section there is a discussion regarding general robot simulation characteristics as well as a particular analysis that is focused on the simulation accuracy of the proposed extension. The paper ends with two case studies and the conclusions.

2. Background

2.1. ROS

Robots generally employ complex electronics, mechanics and software. The software side has the task of coordinating all of the sensing and actuation components while working on or more general objectives of the robotic system. A common approach in order to handle these distinct tasks is to employ a layered architecture based on complexity/abstraction level. Different layers can be physically separated on distinct computing boards.

At each abstraction level, there are well-studied algorithms that ensure optimal performance. Integrating each algorithm into the control architecture can be difficult and error prone due to the use of different programming languages, communication protocols and numeric scales.

Robotics software middleware are software systems that are designed to interconnect specialized software components using common communication channels. The *Robot Operating System* (*ROS*) is an open-source software framework that offers a "structured communication layer" [9], a packaging system, for distributing ready-made algorithm implementations and a multitude of special-purpose applications.

Interfacing a robot with ROS consists of exposing the robot's sensors and actuators to the ROS communication network. After this step, the robot can be controlled using various control algorithms that need not be originally designed for that particular robot. As an example, consider a two wheel mobile robot and a drone. Although the movement mechanisms differ greatly, within ROS there is a single generic movement message that stores linear and angular velocities for all three axes. Using this communication mechanism allows algorithms to be deployed on an extended range of devices without modifications.

The four basic items that compose a ROS system are: (1) nodes, (2) messages, (3) topics and (4) services. Nodes can be compared to operating system processes where both have a unique ID in the system, receive parameters and executed one or more sub-tasks in order to run an application.

Messages represent a previously agreed-upon data format that is used by nodes when exchanging information. A message can be as simple as a single integer value or as complex as camera images and 3D point clouds. The ROS framework provides a set of standard messages, but any application can define custom messages if necessary, using a language-neutral paradigm [9].

Topics are a means of identifying information flows in the publisher-subscriber model that is used in ROS. Each topic is described by its name and the message type that it allows. This identification becomes necessary when dealing with complex systems that transfer various types of information, as there can be multiple publishers and multiple subscribers [9]. This communication method is asynchronous.

Services are used for exchanging data in a synchronous manner, using a request-reply model. A client requests a server to do a certain task and then waits for a response. As with topics, services are also identified uniquely in the system and are composed of two message types, one for the request and one for the response [9].

The ROS system allows a developer to construct a distributed robotic system by connecting multiple nodes, either locally or through a network. The free and open-source license of ROS has allowed it to receive contributions from many sources. One notable example of a ROS package is `tf`. This package offers both a library and inspection/debug applications that are used mainly for geometric tracking and transformations, at the present time and also in the past [4].

All of these components and features have determined researchers to use ROS for a wide range of applications that target both beginner and advanced users. One example for the former user category is "rosbridge", a secondary middleware on top of ROS, that allows remote users to interact with ROS using web technologies such as browsers, Javascript and webGL, in an attempt to lower the robotics knowledge entry barrier for general application developers [2]. At the other end of the scale are robotics applications with tight requirements such as long term autonomy and reliability, operating without human intervention. An example in this sense is the "STRANDS" project which aims at providing a robotic platform that is capable of long term deployment in indoor environments with the goal of monitoring and modeling the environment while also generating alerts for unusual behavior [5].

2.2. V-REP

V-REP is a scalable, general purpose, 3D robot simulation framework [12]. Among the main features of the simulator are: (1) portability, (2) versatility, and (3) a comprehensive set of programming interfaces

The first feature, portability, refers to both operating system portability as well as robotic application portability. V-REP can run on Windows, Linux and Mac and uses internal scripts that are attached to each simulated object and specify its behavior [12]. These scripts, known as *child* scripts are written in Lua and are portable across all supported operating system platforms. These scripts can be attached to complete robots or to specific components such as a custom-designed sensor/effector.

The second feature, versatility, stems from the distributed architecture used for connecting internal calculation modules [12]. V-REP uses both free and proprietary modules for the following tasks [12]: kinematics, dynamics, collision detection, mesh-mesh distance calculation, path/motion planning.

The third feature, the programming interfaces, are important because they allow external applications to interact with the simulation. V-REP offers external programming interfaces for the following languages [12]: C/C++, Python, Java, Matlab, Urbi and a ROS interface.

2.3. e-puck

The e-puck is a small two-wheel robot that was designed for educational use [7]. The robot, presented in Fig. 1, measures 75 mm in diameter and is powered by a dsPIC microcontroller that runs at 64 Mhz and provides 8 KB of RAM memory and 144 KB of flash memory. The most important sensors and effectors are presented in Fig. 1.

A distinct feature of the e-puck is that it uses a free license for both the hardware and the software components [7]. This has ensured the wide spread adoption of the robot in educational institutions and in the development of various hardware and software extensions and tools. On the hardware side, there are several boards that can be connected to the robot's main board in order to add functionalities such as [7]: Zigbee radio communication, omnidirectional vision, line followers and others. On the software side, there are APIs for programming languages such as C/C++, Python, Matlab and others, as well as two well established simulation platforms: Webots and Enki [7].

The two available simulation platforms differ in both licensing and features. Webots is a commercial 3D simulator that uses physics modules and can transfer controllers to the real robot [6], while Enki is a free 2D simulator that features a less accurate but fast physics module [7]. What is missing in this context is a free, accurate, 3D simulator.

V-REP can fill this gap as it has a free license for educational entities and hobbyists and also includes a model of the e-puck robot. The downsides, when compared with Webots, are twofold.

Firstly, the simulated e-puck, in the provided form, can only be controlled using V-REP scripts written in Lua. Although the simulator offers external APIs in various languages, these have to be interfaced with the robot control code written in Lua.

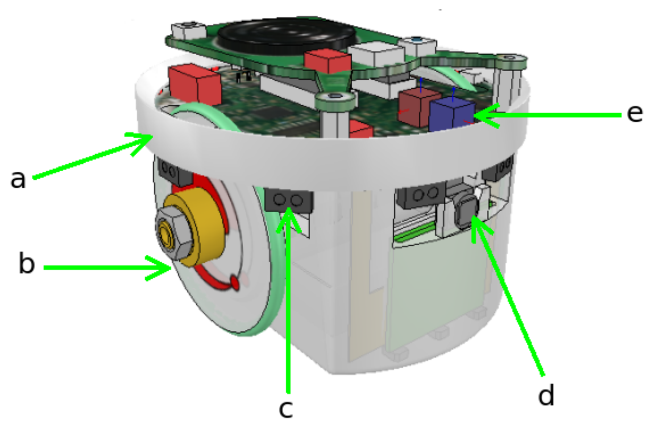


FIG. 1. e-puck simulated in V-REP. The main components, that have been interfaced with ROS are: (a) ring of 8 red LEDs, (b) stepper motors, (c) ring of 8 infrared range sensors, (d) RGB camera and (e) accelerometer and gyroscope

Secondly, there is no method to automatically transfer a controller from the simulated robot to a real robot. This implies that a considerable amount of time will have to be invested in adapting the Lua source code used in the simulation to the C source code that is used on the e-puck, without considering API differences.

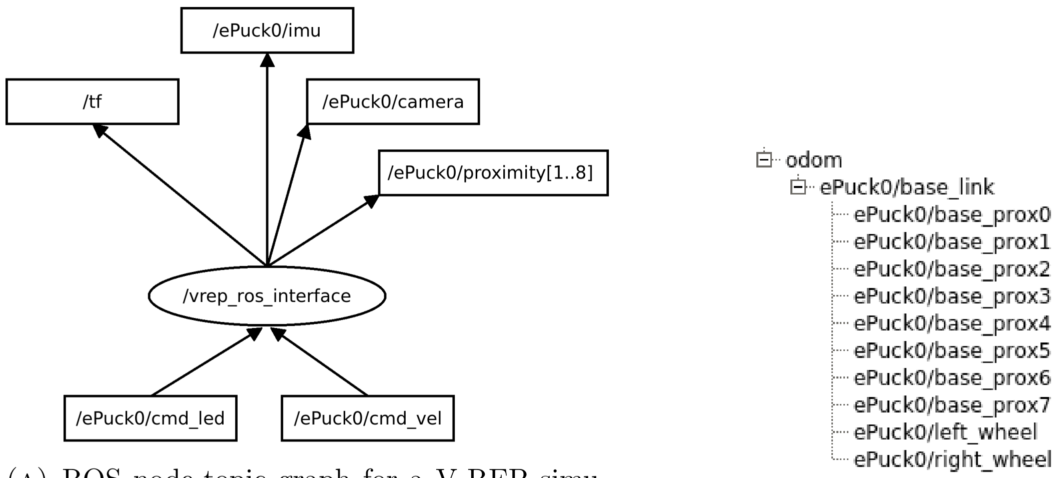
The contribution of this paper, described in detail in the next section, is that of exposing each V-REP simulated e-puck robot to the ROS system. This allows on one hand to control an e-puck robot using generic ROS nodes that are written in either C++ or Python and on the other hand, the same nodes can be used to control a real robot, without any source code modifications. The latter objective was achieved by making the simulated robot use the same communication topics and message types as the real robot.

3. **Design aspects**

The ROS integration of each simulated e-puck is performed through the use of the ROSInterface plugin of V-REP, that comes shipped with the simulator. This plugin offers an almost exact replica of the ROS API within the internal scripts (written in Lua) that control each individual robot.

Simulated robots are usually designed to be a drop-in replacement of the real robot driver [10, p. 93] and this aspect has been considered for the e-puck simulations. The topics that the simulator node will publish and subscribe are presented in Fig. 2a, above and bellow the `vrep_ros_interface` node respectively. This arrangement will be repeated for each robot, under a different namespace.

The Transform Library topic `/tf` is shared by all simulated robots. This is a core library of the ROS system and was designed to accept asynchronous,



(A) ROS node-topic graph for a V-REP simulated e-puck

(B) TF tree

FIG. 2. ROS integration of each V-REP simulated e-puck

distributed sources of information (publishers), that function at various rates and can occasionally experience latency or packet loss [4].

The TF tree for a single simulated e-puck robot is presented in Fig. 2b. Each item in the tree is a coordinate frame, and a transformation vector and rotation quaternion between any pair of frames can be computed by the library [4]. For instance, consider that the distance sensor `ePuck0/base_prox0` would detect an obstacle that is 2cm away from the sensor. In this example, the controller needs to adjust the movement of the robot based on the position of the obstacle related to the center of the robot. This task is greatly simplified by requesting a transformation between the `ePuck0/base_link` and `ePuck0/base_prox0` TF frames. This task would remain straightforward even if the aim would be to obtain a transformation from a pair of TF frames that are part of different robots.

4. **Simulation characteristics**

In order for a simulation to be valuable in the development of robotics applications, didactic or not, there are certain characteristics of the simulation that must be offered. These characteristics are best observed when comparing simulations with real robot experiments, that they intend to replace.

A simulation is generally easier to prepare and more cost-effective than a real experiment [6]. This is particularly true for multi-robot experiments that involve large numbers of robots that have to be reprogrammed and repositioned before each trial run.

Another desired characteristic is simulation speed, where faster than real robot executions allow for a faster development cycle [6].

Accuracy is an important characteristic of any simulator, particularly in robotics where the aim is to replicate a real mechatronic system. In order to maximize simulator utility, this property should be adjustable [10, p. 92].

On the lower end of the accuracy scale, there are 2D simulators that consider perfect sensor measurements and occasionally use simple objects such as cubes in order to represent the world. These simulators usually compensate by offering a faster execution speed and repeatable experiments [10, p. 93].

On the upper end of the scale are the 3D simulators that introduce some (configurable) amount of noise into the sensor readings and also into the kinematics model of the moving robot.

The increase in simulation accuracy, and therefore complexity, can lower the performance of the simulator i.e the simulation speed. The trade-off between performance and accuracy is well studied, especially for the often used rigid-body dynamics [10, p. 95].

4.1. Simulation accuracy when compared to a real robot

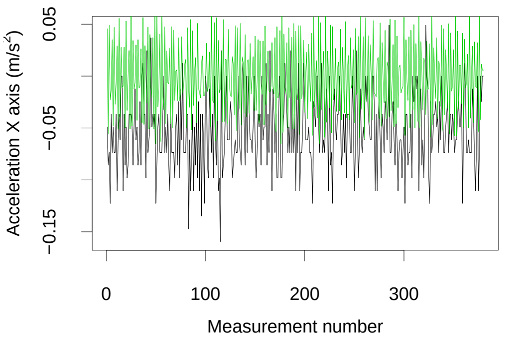
In this context of simulation accuracy, the proposed V-REP extension was evaluated in terms of sensor accuracy. For this purpose, the ROS information provided by the respective devices in simulation was compared with data obtained from real robot experiments. All of the numerical values where extracted from ROS recordings of the experiments, converted to CSV files and analyzed using the R statistical language [11].

Of particular interest in this analysis have been the accelerometer and proximity sensors. By studying the former, one can observe the precision of the internal dynamics engine used by V-REP while the latter is required by obstacle avoidance algorithms and thus has to replicate, as closely as possible, the real sensor parameters such as minimum, maximum range and noise levels, among others.

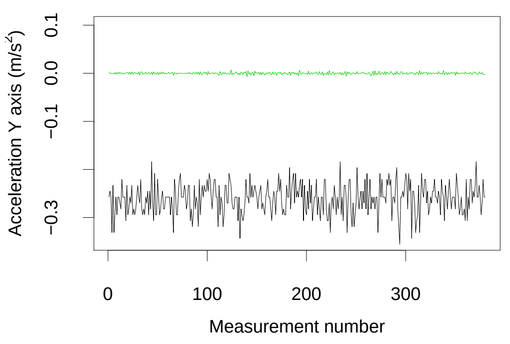
The accelerometer values for the simulation and real robot have been compared over each component in Fig. 3. A statistical summary of the observed values is shown in Table 1. The coordinate frames of all sensors and the robot center use the following convention: X is forward, Y points to the left and Z points upwards. Although the average values do not coincide, it can be

Table 1. Statistical summary of accelerometer values for straight movement for a real and simulated robot

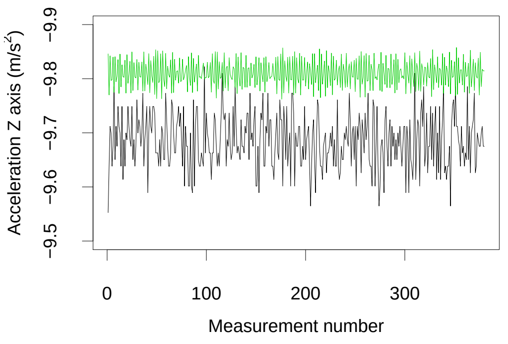
	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	St. dev
sim x	-0.06554	-0.03160	-0.00663	-0.00045	0.03301	0.07084	0.03630
sim y	-0.00677	-0.00151	-0.00017	-0.00010	0.00151	0.00663	0.00215
sim z	-9.85761	-9.83361	-9.80537	-9.81005	-9.78845	-9.76215	0.02562
real x	-0.15941	-0.07358	-0.04905	-0.05254	-0.03679	0.04905	0.03559
real y	-0.35561	-0.28204	-0.25751	-0.26087	-0.24525	-0.18394	0.03079
real z	9.55249	9.65059	9.68737	9.68479	9.71190	9.82226	0.04930



(A) X axis



(B) Y axis



(C) Z axis

FIG. 3. Accelerometer values recorded while moving on a straight path. Real robot values are plotted using black lines while simulations use a green color.

observed that the differences are not major and that the dynamics engine in V-REP does produce vibrations as the robot moves.

The only axis where the differences are notable is Y and this is most probably caused by the flatness of the plane onto which the robot is moving. In real life, the robot was placed on a wooden surface, while in simulation, the plane was perfectly flat.

The proximity sensors are a vital component of the navigation stack of any robot because they provide means of avoiding obstacles, following walls and, to some extend, building a map of the world. It is for these reasons that the sensors must provide similar range information in simulation and real life scenarios.

In order to evaluate the proximity sensors, a rectangular wall was placed in front of the real and simulated robots at a distance of 0.5 cm. This wall was detected by four front-facing sensors numbered 0, 1, 6, 7, where the first two are the closest to the center of the robot while the latter are farther away.

The robot was set to move backwards for three seconds at a constant speed, enough to exit the maximum sensing range of the sensors (5 cm).

In Fig. 4, there is a plot of the range that was detected by sensors 0 and 1, during real and simulated tests. The values obtained from V-REP simulated sensors are depicted as "Sim sensor 0" and "Sim sensor 1". It can be easily seen that V-REP distance sensors offer an exact minimum distance between the sensor and the target object that is within the configurable detection cone of the sensor [12]. On the other hand, real robot sensors, that function using infrared-light, introduce a certain amount of noise, even when the robot is stationary, due to lighting variations.

Such differences in noise level between real and simulated sensor values can negatively influence the response of the previously mentioned navigation algorithms. This was the main motivation for adding a configurable noise model to the proximity sensors of V-REP simulated e-pucks. In order to analyze the sensor noise, the real robot was placed in an area with no obstacles and the sensor values where read. Without noise, the returned value would be the maximum sensing range, 5 cm or 0.05 m. By subtracting the actual value from 0.05, the non-zero values that remained were all error values. For the specific e-puck used in the experiments, sensor 0 proved to be the most error prone because there where a total of 61 errors out of 380 measurements or otherwise said, a noise probability of 16%.

The next step towards implementing a noise model for the simulated e-puck was the analysis of the value distribution, for the real measurements. This distribution is presented using a histogram in Fig. 5a. Although not a perfect match, the Gaussian distribution was chosen to model simulated sensor noise, due to its simplicity, and is presented using a histogram in Fig. 5b. The distribution was configured to use the same mean and standard deviance as observed for the real sensor values. In the actual implementation, these parameters are user configurable.

Random numbers could now be generated according to the configured distribution, using a fast generator such as the Box-Muller algorithm [1]. This algorithm requires only a pair of uniformly distributed (pseudo) random values U_1, U_2 in the $[0; 1]$ interval, for use in equation (1).

$$X = \sqrt{-2log_e\ U_1} * \cos(2\pi\ U_2) \tag{1}$$

$$Y = noiseMean + noiseStDev \cdot X \tag{2}$$

The result, variable X, is a Gaussian distributed random number with mean 0 and variance 1 [1]. For use in the noise model, the variable is adjusted to use the user configurable mean and standard deviance, as shown in (2).

Once the Gaussian distributed random variable Y is generated, obtaining sensor noise is a matter of adding the Y to the sensor value, with a certain probability. In Fig. 4, the simulated sensor values that include generated noise

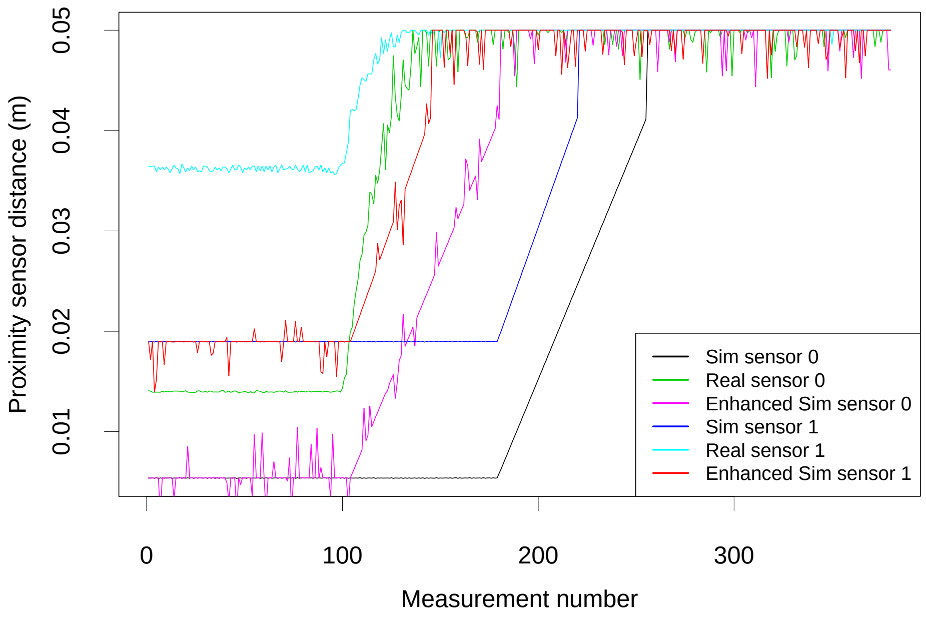
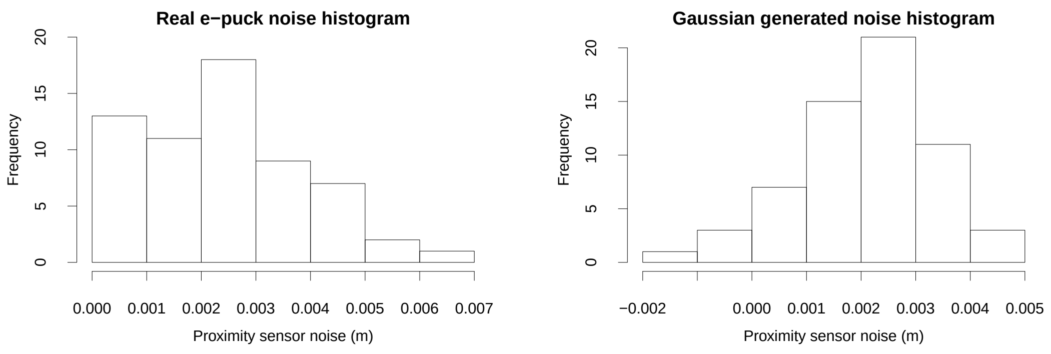


FIG. 4. Plot of the distance returned by proximity sensors 0 and 1 for real and simulated experiments.



(A) Real sensor (B) Simulated sensor

FIG. 5. Histogram representation of the sensor noise distribution

are labeled as ”Enhanced Sim sensor” 0 and 1. It can be seen that these values are more similar to the values returned by real sensors.

5. **Pedagogical case studies**

This section includes brief descriptions of several pedagogical use case scenarios of the presented software. These scenarios increase in complexity and can be used for hands-on robotics courses.

5.1. Sensor data live plot

One of the first tasks that have to be done before attempting complex robotics experiments is to ensure that information travels from one node to another and has the expected value, delay, time variation and other numerical attributes.

As ROS was designed with the UNIX philosophy in mind, i.e. using many small specialized programs [10, p. 4], there are several tools for this task.

For this example, a readily available plotting application, `rqt_plot`, is considered. This tool allows students to view a time plot of arbitrary numeric values that are extracted from ROS messages.

For instance, the plot from Fig. 6 was obtained by issuing the command `rosrun rqt_plot rqt_plot /ePuck0/imu/linear_acceleration`. Students can notice from this plot that the Z component of the acceleration has values close to $-9.8\ m/s^2$ while the other two components have value very close to 0. The actual variation of the values can be observed by zooming in, thus observing that even as the robot is standing still, there is a small variation in the accelerometer readings.

5.2. Multi-robot simulations

The use of multiple cooperating robots for a single robotics task is a widely studied topic because of advantages such as cost reduction, increase in efficiency and fault tolerance [8].

All of these advantages require a strong coordination mechanism and supporting communications and interaction mechanisms. Teaching multi-robot systems theory can be complemented with case studies involving simulated e-pucks along with the distributed communication/coordination mechanisms provided by the ROS environment.

A simple case study can be that of following another robot, an example often referred to as 'follow the leader'. Implementing this behavior using the e-puck is relatively straightforward given the circular positioning of the range sensors. A proportional controller can be used for this purpose.

The disadvantage of relying exclusively on direct sensor information is that the task depends on sensor parameters such as range. For the e-puck, this means that the leader and follower should be no more than 4 cm apart. This issue can be avoided by using the TF library that is available in ROS and requesting a transformation between the centers of the leader and follower robots.

An example of the utility of the TF library is presented in Fig. 7 where the leader robot (ePuck0) is out of the sensing range of the follower robot (ePuck1). In this circumstance, the user can request a transformation from the frame of the leader to that of the follower, using the global position as reference. The

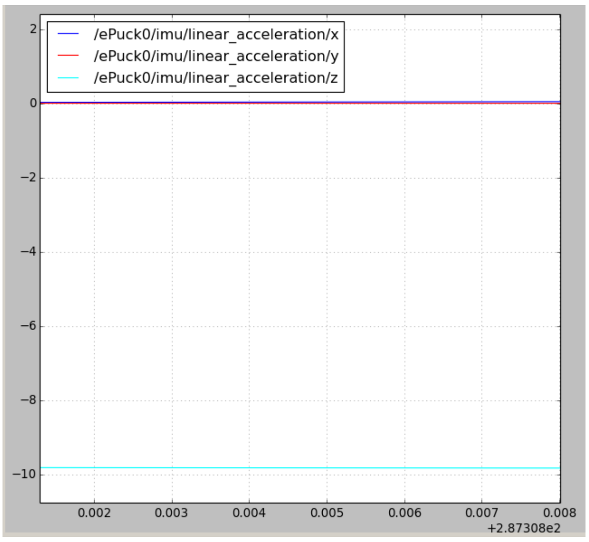
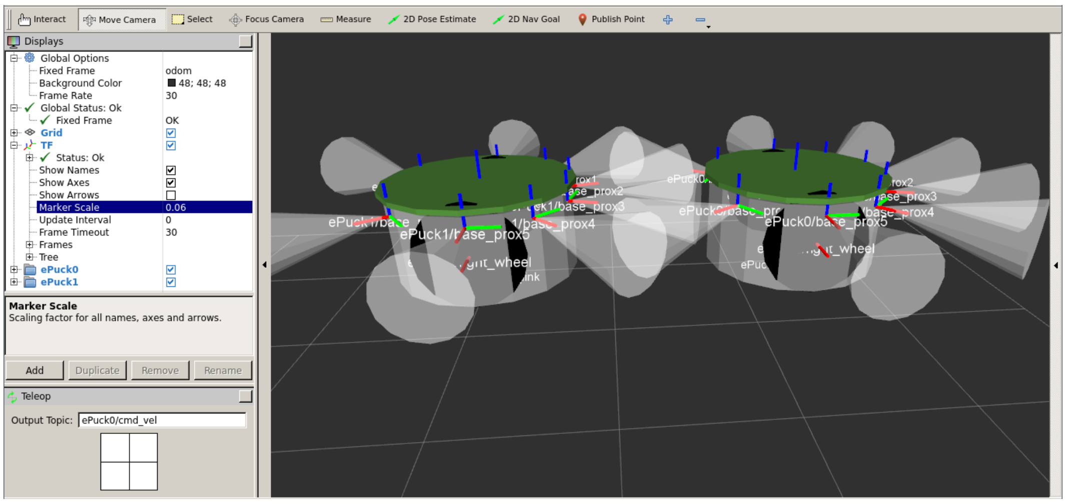


Fig. 6. Time plot of the accelerometer values (vertical axis) for a simululated e-puck



```
Global Options
  Fixed Frame          odom
  Background Color     48; 48; 48
  Frame Rate           30
Global Status: Ok
  Fixed Frame          OK
Grid                   ✓
TF                     ✓
  Status: Ok
  Show Names           ✓
  Show Axes            ✓
  Show Arrows          ☐
  Marker Scale         0.06
  Update Interval      0
  Frame Timeout        30
  Frames
  Tree
ePuck0                 ✓
ePuck1                 ✓

Marker Scale
Scaling factor for all names, axes and arrows.

Output Topic: ePuck0/cmd_vel
```

Fig. 7. Rviz visualization of the follow the leader experiment with two e-puck robots. TF frames are represented using axes and labels.

resulting transformation will consist of a translation and rotation and will be used to generate linear and angular speeds, the output of the controller.

6. Conclusions

Robotics can be both fascinating and difficult to comprehend by students because robotics courses generally involve complex algorithms and strategies.

These concepts can be easier to comprehend by testing each concept on simple mobile robots that are controlled using high level middleware software. The robots allow for an in depth view of the real challenges faced by mobile robots while the middleware, ROS in this case, allow for a fast deployment of ready-made software components, with the objective of studying the interaction between them and also implementing high level control strategies.

All of these advantages imply a certain financial cost because although most middleware projects (including ROS) are free software, the actual robots can have a high price. A sufficiently accurate computer simulation of the robot and its operating environment can provide students with a much needed tool that allows for fast experimentation of various concepts.

This paper has presented an extension for the V-REP simulator that allows one or more simulated e-puck robots to be exposed to the ROS middleware using the same interfaces as the real robot thus allowing the same ROS code to control both real and simulated robots without any modifications. The precision of the simulated devices found on the robots was analyzed by comparing their output with real robot values. A simulated noise model was proposed and applied to the sensors in order to compensate for the initial noise-free values that did not reflect real-life conditions.

Further work involves similar adaptations for other types of robots that have distinct characteristics that the simulation must reflect. For instance, quadrotor drones require an accurate particle simulation model that has to be coupled with inertial measurement units and various specialized sensors 3D optical flow motion detection sensors.

REFERENCES

[1] *G. E. Box* and *M. E. Muller*. A note on the generation of random normal deviates. In The annals of mathematical statistics, **vol. 29**, no. 2, pp. 610–611, 1958.

[2] *C. Crick*, *G. Jay*, *S. Osentoski*, *B. Pitzer* and *O. C. Jenkins*. Rosbridge: Ros for non-ros users. In Robotics Research, pp. 493–504. Springer, 2017.

[3] *A. G. Florea* and *C. Buiu*. ROS extension of the e-puck robot simulated in V-REP. `https://github.com/andrei91ro/epuck_vrep_ros`, 2018.

[4] *T. Foote*. Tf: The transform library. In Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference On, Open-Source Software workshop, pp. 1–6. 2013.

[5] *N. Hawes*, *C. Burbridge*, *F. Jovan*, *L. Kunze*, *B. Lacerda*, *L. Mudrová*, *J. Young*, *J. Wyatt*, *D. Hebesberger* and *T. Kortner*. The strands project: Long-term autonomy in everyday environments. In IEEE Robotics & Automation Magazine, **vol. 24**, no. 3, pp. 146–156, 2017.

[6] *O. Michel*. Webots: Professional Mobile Robot Simulation. In International Journal of Advanced Robotic Systems, **vol. 1**, no. 1, pp. 39–42, 2004.

[7] *F. Mondada*, *M. Bonani*, *X. Raemy*, *J. Pugh*, *C. Cianci*, *A. Klaptocz*, *S. Magnenat*, *J.-C. Zufferey*, *D. Floreano* and *A. Martinoli*. The e-puck, a robot designed for education in engineering. In Proceedings of the 9th Conference on Autonomous Robot Systems

and Competitions, **vol. 1**, pp. 59–65. IPCB: Instituto Politécnico de Castelo Branco, 2009.

[8] *L. E. Parker*, *D. Rus* and *G. S. Sukhatme*. Multiple Mobile Robot Systems. In Springer Handbook of Robotics, pp. 1335–1384. Springer, 2016.

[9] *M. Quigley*, *K. Conley*, *B. 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, **vol. 3**, p. 5. Kobe, 2009.

[10] *M. Quigley*, *B. Gerkey* and *W. D. Smart*. Programming Robots with ROS: A Practical Introduction to the Robot Operating System. O'Reilly Media, Inc., 2015.

[11] *R Core Team*. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, `https://www.R-project.org`, 2017.

[12] *E. Rohmer*, *S. P. N. Singh* and *M. Freese*. V-REP: A versatile and scalable robot simulation framework. In 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1321–1326. 2013.