Introducing modern robotics with ROS and Arduino, including case studies

Submitted: 28th May 2013; accepted: 25th July 2013

Igor Zubrycki, Grzegorz Granosik

DOI: 10.1431/JAMRIS_1-2014/9

Abstract:
This paper describes our experience with introducing modern robotics through Robot Operating System. ROS framework allows rapid robot prototyping and gives access to many state-of-the-art robotic solutions. It is however, software oriented and requires its users to understand well software development ideas and methods. While teaching undergraduate students ROS, we came up with some solutions how to introduce it to people without a deep background in computer science. The paper presents our Mymodel robot *application that simplifies modeling of the robots using URDF format and some Arduino based programs. We have also reported results of students' projects.*

Keywords: *ROS, Arduino, Robotics education*

1. Introduction

The robotics curriculum must contain the laboratory stage. This is absolutely necessary to familiarize students with real robots, their control systems and software. However, an interesting approach is to proceed this stage by modeling and simulation. For several years we have been using two convenient applications to teach students how to model and simulate robots and the whole robotic stands, namely: the combination of Robotics Toolbox (for Matlab) with RoboWorks, and the EasyRob software [7]. These programs provide tools to build graphical models of robots, to manipulate them, and analyze kinematics and dynamics. Recently, much more powerful solution appeared that can support both simulation and real control stages of robotics curriculum.

ROS (Robot Operating System) is an unified and robust framework for robot modelling, control and visualisation [9]. It is a more and more popular tool for rapid prototyping of robot software as it provides an easy way to integrate, test and reuse algorithms constructed by robotic community around the world. And it is an open source, too. However, because of its capabilities and scope, ROS has a fairly steep learning curve [10]. This problem is more distinct if the user has only a little background in computer science, what is the case for the bachelor course in Automatic Control and Robotics at the Lodz University of Technology. We believe though, that the benefits of using ROS are vast and worth our work of finding skillful methods, easy to use tools and appropriate knowledge, to involve even less "computer science type" students to use this modern robotic tool. In this paper we will describe methods and tools, that worked best in our case. Arduino is the hardware platform we have employed in this quest.

2. Motivation

ROS is a tool used by robotic teams worldwide when designing large robotics systems. The main reasons for its popularity, that also led us to the introduction of ROS for our students, are as follows [9]:

1) ability to rapid prototype. There is a multitude of tools and libraries that were created around ROS. It is possible to connect and "pipeline" these tools literally in a few hours. Because of that, relatively small teams and beginning students do not need to "reinvent the wheel" and can create entire robotic applications.
2) modern software architecture. ROS is a modern software architecture that allows to connect easily different applications and devices. Users can build systems where most of processes work in parallel and on different machines without building multithreading or networking procedures by themselves.
3) "Thin" ideology. Programs to be used in ROS do not need to be highly integrated. There only has to be a small executable running that exposes program's functionality to ROS or extracts some information from it. This opens the way to reuse all specific tools that were created outside ROS.
4) Ease of debugging, visualisation and logging. ROS has a number of tools that enable users to check and save system's state. Users can see system's state graph (*rxgraph* tool), plot different variables online with *rxplot*, or visualise whole robot and its sensors readings by using *Rviz*. All data can be easily archived and replayed by using rosbag.
5) ROS is well documented and supported. Beginners can find tutorials online, there are two ROS books [6] [8] and lively ROS forum [2].
6) Free and Open Source. Most of ROS packages is open source and has straightforward licences [3]. This simplifies development and allows to engage wide range of contributors both from within and outside academia. Open source gives also partial guarantee that long time development will be possible – even if library creator stops development, we will be able to improve and compile it by ourselves.

However, the introduction of such sophisticated and modern framework to our students involved some difficulties, these are the most important we have faced in our work:

1) transition from simple single threaded systems. Our students have different computer science skills but the experience of most of them is limited to single threaded applications – designed in the MATLAB environment or on devices such as Atmega microcontrollers. To use ROS effectively students have to understand its structure and philosophy – mainly the need to treat each functionality as a single part – that will become *ROS Node*, and which will communicate to others only by using *ROS Messages*.

2) ROS is an open framework and most of its power comes from the possibility to use other teams' solutions. Unfortunately, these solutions are usually "proofs of concept" and using them becomes an integration task, which is difficult for the people with little experience with the software development.

3) ROS is a Linux tool and to use it effectively, users have to have experience in this environment. Basic parts of ROS are available as Ubuntu Packages but more advanced and less popular ROS tools are only available as sources on the Github or other Internet revision control services. Users need to know basics about version control, compiling, makefiles, etc.

4) ROS framework is rapidly developing, there are many versions that have different APIs, different tools and functionalities. Tutorials that work in one ROS version, sometimes do not work or even mislead in other – what is utterly frustrating for the beginners.

3. Target group

As the Robot Control Department we provide robotic related courses for students from different faculties of our university. We teach basic robotics, robot control – that are mainly about industrial robotics as well as more advanced subjects such as mobile robotics, vision systems or rehabilitation and service robotics. We understand, that a large part of these courses can be based on ROS, what would allow students to work with single framework or even on one project throughout different courses.

To derive and test solutions we have conducted series of workshops for the second year of the second year of the bachelor course in Automatic Control and Robotics (Fig. 1). This group had already learned some basic engineering subjects, programming (C++, MATLAB) and had several courses on the electrical engineering. Unfortunately, these students had very little experience with subjects from computer science curriculum – software development, object oriented programming, etc. Therefore, training them to use ROS turned out as a challenge.

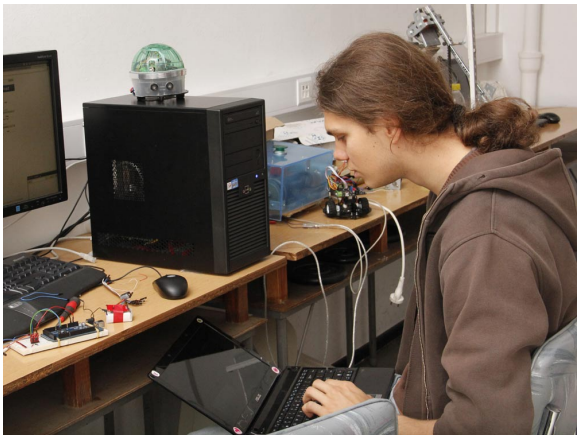

Fig. 1. Picture from one of our ROS introduction workshops

4. Solutions

To enable our students working with ROS we came up with a number of ideas. At the beginning we planned to base our teaching mainly on the internet tutorials [4] and a textbook [6]. Students were supposed to build small applications that would contribute in our bigger projects. Meetings were to be spent on discussing matters related to robotics.

Unfortunately, because of the weaknesses of ROS, described in Section 2, our students, especially the less computer adept were unable to start practical work, even after completing all beginners tutorials. Also discussions with them proved that they do not understand the ROS nomenclature and basics of work.

We understood, that our students would learn ROS faster if we used their existing skills and divided their work into practical modules, through which they would be guided.

4.1. Arduino and rosserial

We have found that students weren't able to grasp basic ROS ideas of using Publish/Subscribe and Services based only on the tutorials. What helped us enormously was introduction of *rosserial* package and *rosserial_arduino* [1].

The *rosserial* is a protocol that allows to connect different hardware to ROS system using serial connection. The type of activity (Subscriber/ Publisher or Service) and a message format are defined directly on the device, with the use of *rosserial* libraries. There is also a python node on the host computer that forwards messages and makes nodes created on the device visible to ROS. As long as devices use *rosserial* format to communicate using serial connection, they can have any functionality. It provides an excellent way to communicate with simple real time devices such as microcontrollers or sensor arrays. We have found that *rosserial_arduino* – the extension of *rosserial* for Arduino platform [1] – can significantly help in integration of custom hardware with ROS and is also a convenient way to teach our students ROS. There are several reasons for that:

1) Arduino has excellent documentation, IDE and

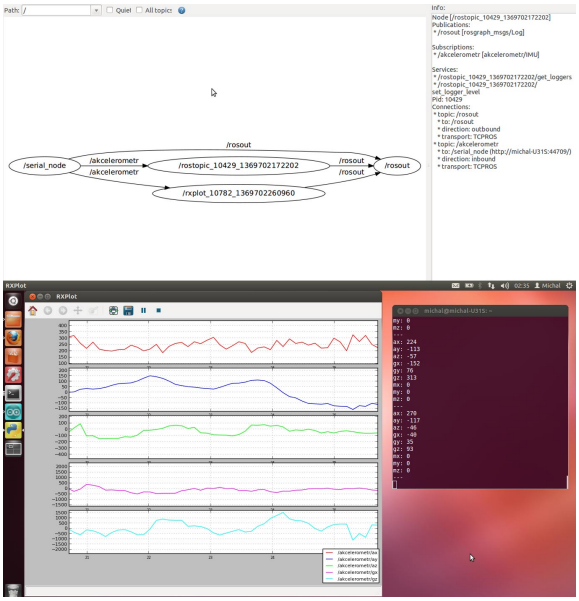

Fig. 2. ROS Computation Graph created by our student Michal Kielan to read IMU sensor (upper picture), visualization of the readings from this sensor (lower picture)

Fig. 3. Micromouse, a small differential drive robot based on a modified Arduino board, used on our workshops and controlled through bluetooth by ROS

community support. We introduced it independently with our robotic platform – robo mouse, small differential drive robot based on modified arduino board.

2) Using real sensors or actuators, connected to Arduino, helped our students see and understand ROS functionality, and benefit from it.

3) Students were able to do "real work" and to use ROS in their own projects – this was an enormous motivation.

To test this approach we have used small mobile robots (see Fig. 3) equipped with differentially driven wheels with encoders, IR distance sensors, sonars, RC receiver, BlueTooth modules, and Romeo controller, which is Arduino-compatible and integrated with H-bridges.

Students realized several projects: sonar reading, chaotic movements of the robots with obstacle avoidance, remote controlled mouse, web-controlled mouse. Additionally, we have demonstrated other projects: smartphone-controlled mouse, sensor glove readings.

4.2. Working in groups

We have also found that working in groups – pairs or trios – made learning more effective. Students shared their experiences from working independently and explained to each other how different functionalities of ROS work.

To make teamwork easier we have set up a forum and encouraged students to use Git-based source code management (which we taught them by using Internet teaching games).

Teamwork motivated students – it was harder for them to explain delays or lack of progress to their peers than to us. It also involved their different skills – each of the students could work on the part of the project he felt best in. This somehow reflects the ROS philosophy.

4.3. Robot modeling and debugging

Students from our test group have already passed Introductory robotics course and have gained some theory on industrial robots, forward and inverse kinematics, and manipulator dynamics. This knowledge could be easily illustrated by ROS set to work with simulation tools such as Gazebo. It has a built-in transformation system that can dynamically construct a transformation tree – known from kinematics. In order to further prepare the robot's visualization or simulation the URDF model is required. URDF – unified robot description format is XML based structure that describes important robot properties [5]:

- shape and dimensions (*link* element's *origin* element properties and *collision* element properties)
- visual properties (*link* element's *visual* element properties)
- inertial properties (*link* element's mass and inertia properties)
- robot joint characteristics and limits (*joint* element's properties)
- placement and properties of sensors (*sensor* element's properties)

Example of URDF description and resulting tree are shown in Fig. 4.

Even though the structure of URDF file is quite clear, students had some problems to create these descriptions from scratch. The biggest reason for that is the number of steps required to launch such a file to visualize robot and manipulate its joints. It would be much easier if they could interactively modify a file and see results immediately.

Online app Mymodel robot We have created a tool to simplify testing of URDF model files by presenting them directly in the web browser. It is a network tool, that does not require any installation on the students behalf – only a modern browser (that supports WebGl format) is needed. Our aim was to make usage of this

```
<robot name="KAWASAKIRS03N">
  <link name="base">
    <visual>
      <geometry>
        <box size="2 2 0.1"/>
      </geometry>
      <material name="black">
        <color rgba="0 0 0 0"/>
      </material>
    </visual>
    <collision>
      <geometry>
        <box size="2 2 0.1"/>
      </geometry>
    </collision>
  </link>
  <link name="zl00">
    <visual>
      <geometry>
        <box size="1.4 1.7 0.5"/>
      </geometry>
      <material name="AlmostWhite">
        <color rgba="0 0 0.9 0"/>
      </material>
      <origin rpy="0 0 0"/>
    </visual>
    <collision>
      <geometry>
        <box size="1.4 1.7 0.5"/>
      </geometry>
    </collision>
  </link>
  <joint name="JT1" type="revolute">
    <parent link="base"/>
    <child link="zl00"/>
    <axis xyz="0 0 1"/>
    <limit lower="-2.7925" upper="2.7925" effort="1" velocity="1"/>
    <origin xyz="0 0 1.1"/>
  </joint>
</robot>
```

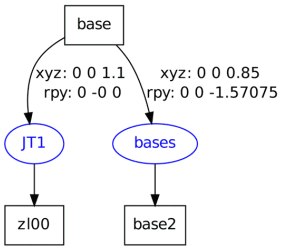

Fig. 4. Example URDF description and resulting tree

tool as simple as possible. Users only need to put their URDF file into form field, which is then parsed into a robot model and shown in the browser with appropriate control sliders for all movable joints. *Mymodel robot* application diagram is shown in Fig. 5.

The application that does most of the processing is created using modern web applications libraries and it follows MVC (Model View Controller) model where data is isolated from its view. The most important advantages of the proposed application are:

1) Beginning users do not install Linux and ROS Framework to start working with URDF files
2) Tool is easy and straightforward, allowing fast model testing and evaluation
3) Users have an easy way to show their models to other people
4) As a web tool, there is a guarantee that each student uses the same tool, what simplifies debugging

Our tool was well received by the participants of our workshops, we have introduced it also to students in our normal curriculum, where they could write and test models of industrial manipulators, an example of a Robot model created by our student is shown in Fig. 6. Also members of ROS Users forum were interested in our tool. We have received a number of emails with suggestions for further development or usage. One of them was suggesting to use it on ROS websites to show the manipulable image of the robot model used in the project.

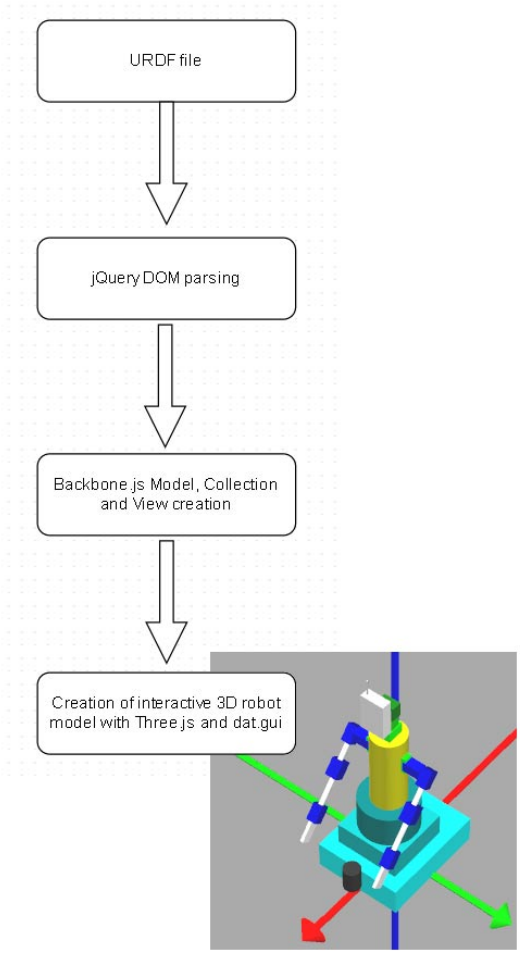

Fig. 5.* Mymodel robot *aplication diagram

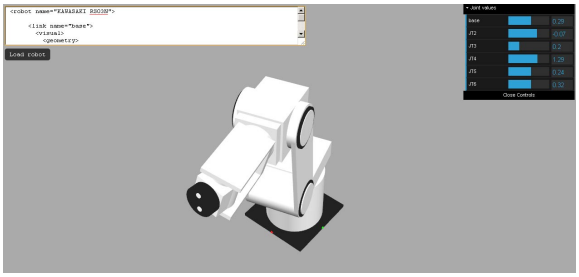

Fig. 6. KAWASAKI R303N robot model created by Pawel Guzdraj and Patryk Blesinski with* Mymodel robot *application

Currently, there are some other projects that aim to make browser based tools for ROS – even move whole Rviz visualization tool to browser. We expect that the learning curve for these tool will be still rather steep as they are too sophisticated. From our experience and ROS forum suggestions there is a need for simple, straightforward tools that can be used by beginners.

4.4. Working on big projects

Students of Automatic Control and Robotics course usually plan to become engineers. Because of that, they are entirely focused on acquiring practical skills and receiving experience that would be appreciated by

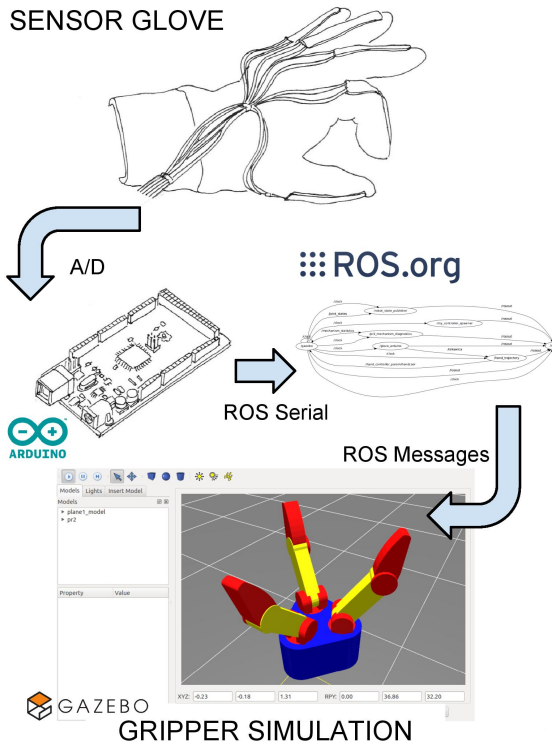

Fig. 7. Structure of our Sensor Glove acquisition and control system [11].

their future employers.

To motivate them more, we tried to involve even the beginning students in our "real" work: they could participate in modeling a mobile manipulator which we are designing or be involved in the preparation of robotics contests like Robotour.

ROS is used in all of our current projects and we demonstrated to our students the functionality that is available. We have spend considerable time describing and demonstrating our in-house manufactured sensor glove that was designed to control a three finger gripper [11] – right now we are in the process of connecting a real gripper to ROS so that the sensor glove will not only be controlling a Gazebo simulation(shown in Fig. 7) but also a mechanical unit.

As a result students became much more motivated as they could see real live applications of ROS framework. They could also relate to ROS better – their questions have become specific.

5. Students' projects and feedback

The group of students from before mentioned course has been involved in several projects: building robotic quadcopter, preparing modules to use in Robotour race (mainly localization and 3D mapping), high definition vision system for robots.

5.1. Quadcopter

This project is in the early stage of testing components and building mathematical model of the driving system consisting of BLDC driver, motor, and propeller. Therefore, ROS is used to easily acquire, save and process large amount of data. Three nodes have been prepared:

- Communication with low level controller
- Main program processing data and publishing them in a convenient form
- Acquiring and saving data to a disk file

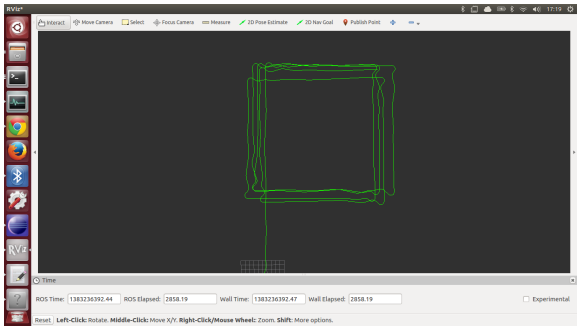

Fig. 8. Rviz used for showing robot's path after running several closed loops.

We can also in-depth analyze some other data coming from the controller, e.g. errors or control signals, as shown in Fig. b.

5.2. Robot localizer

This project shows the potential of ROS to build some parts of operator's console to track robot's position on the prescribed path, and specifically include:

- wireless communication between operator's computer and the robot via Bluetooth link,
- receiving information from sensors (encoders, IMU module)
- processing the received data
- displaying various information interesting for the user
- graphical representation of the position of the robot
- drawing graphs of various waveforms
- calculation of the robot's control signals

As a few students were involved in the project the subtasks were allocated and with help of ROS communication structure sharing and exchanging of all information was seamless. One of the programs was responsible for communicating with the robot using Arduino on board. This process simply publishes data received from the robot and sends back to the robot control data. The other process reads data from the topic, calculates control signal and publishes it for the use of the former program. Substituting data processing program, as well as adding any other nodes using data from the robot is very easy. Using ROS tools we can visualize interesting signals, as shown in Fig. 8.

We can also in-depth analyze some other data coming from the controller, e.g. errors or control signals, as shown in Fig. 9.

Yet another way to access data is printing them directly in the terminal window associated with time stamp, as shown in Fig. 10.

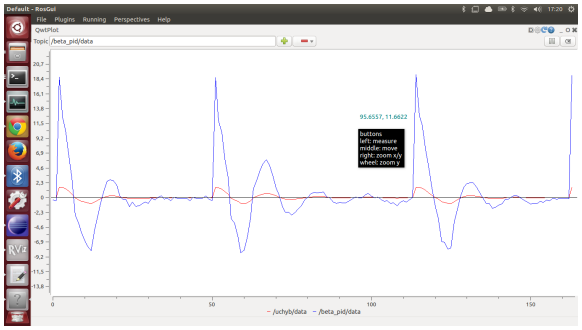

Fig. 9. Time plot of the heading error(red) and control signals (blue).

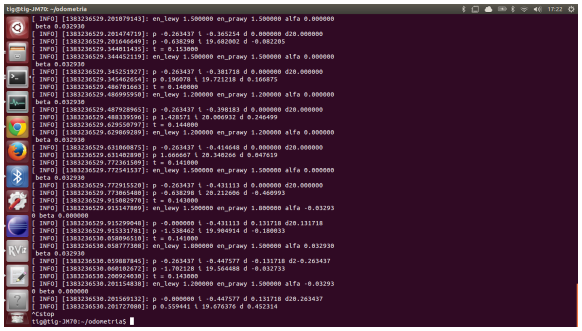

Fig. 10. Terminal access to topic data using rostopic echo command.

5.3. Robotour contest

ROS was also used as the main framework for autonomous robot designed for Robotour contest. This outdoor competitions allows equipping robots with GPS receivers and additional sensors to keep vehicle on pavements in the park, also based on the information available on open-street-map. In this particular case students had to solve number of problems:

- communicating with the low-level controller based on Arduino,
- acquiring and interpreting data from the GPS receiver,
- gathering and interpreting data from the laser scanner,
- capturing and processing data from the camera,
- planning route and calculating control signals.

Students prepared for this matter network of eight nodes communicating within ROS. Again the main advantage of ROS environment appeared seamless integration of different programs prepared by different people from a team. The second important lessons learned by students is reuse of software components – in all shown cases modules related to communication with Arduino were the same.

Project members appreciated numerous tools available in ROS for monitoring and visualization (Rviz, RosGui), as well as rich internet resources and lively forums. Even in a rather big Robotour project it was observed that ROS overhead is very small and the load of the average laptop computer running several nodes is relatively small.

6. Conclusions

Our main conclusion of the work we have already done with ROS is that the best way to introduce this framework is to use simplified solutions. Our students are not experts in computer science and have little experience in typical software development. Yet they have broad knowledge in other disciplines that can be used to introduce them to ROS. Making use of physical devices such as Arduino boards with sensors makes ROS functionality easier to understand as well as gives more motivation than just a simulation.

Experiments with online tools convinced us that this approach is also attractive. We can introduce students to some parts of ROS functionality without having them install the whole ROS system. This will be especially valuable in normal curriculum where time and students' motivation is limited.

Students work more effectively with some guidance (in addition to tutorials and books) and when divided into groups working together on the same project they can teach and motivate each other.

Our last observation is that it is important to show students some "impressive demos". ROS is very broad and students need to have reasons to explore it. After showing them our applications which use Kinect, sensor gloves or smartphones they were much more motivated and wanted to increase their knowledge.

We assessed students' knowledge and motivation by inviting them to participate in several projects where both ROS and Arduino could be used. Then, we systematically checked their progress and directly asked them questions about their knowledge and opinions of ROS and what they plan to do in the future with the knowledge and skills they acquired. In surveys, when asked what features of ROS they find the most valuable they pointed: lightweight, strong ROS community, ease of visualization and modularity. As the most troublesome disadvantages of ROS students indicated:

- rather steep learning curve demotivating some of them,
- frequent changing of versions (Fuerte, then Groovy, now Hydro) makes it hard for different students to have the same environment,
- tools such as Gazebo became less integrated and more difficult to use for beginners.

Nevertheless, we are observing snowball effect – new students are encouraged by seeing what more experienced students did. Also, we have more in house experience with solving and explaining recurring problems and we can help students better

AUTHORS

Igor Zubrycki* – Lodz University of Technology, Stefanowskiego 18/22, 90-924 Lodz, e-mail: igorzubrycki@gmail.com, www: www.robotyka.p.lodz.pl.

Grzegorz Granosik – Lodz University of Technology, Stefanowskiego 18/22, 90-924 Lodz, e-mail: gra-

nosik@p.lodz.pl, www: www.robotyka.p.lodz.pl.
*Corresponding author

ACKNOWLEDGEMENTS

Research was partially supported by the National Centre for Research and Development under grant No. PBS1/A3/8/2012.

REFERENCES

[1] "rosserial tutorials", October 2011. Accessed: 20 April 2013.

[2] "ROS answers forum", May 2013. Accessed: 5 May 2013.

[3] "ROS developers guide", May 2013. Accessed: 5 May 2013.

[4] "ROS tutorials", May 2013. Accessed: 3 May 2013.

[5] "URDF tutorials", May 2013. Accessed: 6 May 2013.

[6] P. Goebel, *ROS By Example*, Publisher is empty!, 2013.

[7] G. Granosik and D. Zarychta, "Application of simulation software for robot modeling". In: *Proc. of VII National Robotics Conference*, Ladek Zdroj, pp. 101–109.

[8] A. Martinez and E. Fernández, *Learning ROS for Robotics Programming*, Packt Publishing, 2013.

[9] 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.

[10] B. Smart. "Teaching robotics with ROS: Experiences, suggestions, and tales of woe", June 2012. Accessed: 4 May 2013.

[11] I. Zubrycki and G. Granosik, "Test setup for multifinger gripper control based on robot operating system (ros)". In: *Robot Motion and Control (RoMoCo), 2013 9th Workshop on*, 2013, pp. 135–140.