# A Platform for Experimental Education of Control Science

**Alen Turnwald** [*], **Francisco J. Garcia** [**], **Dina Martynova** [**], **Zhijie Lin** [**], **Wen-An Zhang** [***], **Steven Liu** [*]

[*] *Institut for Automatic Control, University of Kaiserslautern, Germany, Email: {turnwald , s.liu}@eit.uni-kl.de*
[**] *Student at Institut for Automatic Control, University of Kaiserslautern, Germany, Email: garciar@rhrk.uni-kl.de, martyn@rhrk.uni-kl.de, zhijie9lin@gmail.com*
[***] *Department of Automation, Zhejiang University of Technology, HangZhou, Email: wazhang@zjut.edu.cn*

**Abstract:** This paper introduces a concept for control educations based on a combination of a simulation and a real robot. It is suggested to split the procedure of controlling a robot into three steps. First, students work with a Matlab/Simulink model for rapid design and testing of control algorithms. The practical application is split into two steps. Robot Operating System (ROS) is used for task distribution and communication between different parts of the robot. In this framework, first the students implement the algorithms on an embedded processor, here Raspberry Pi, that is connected to a realistic robot simulator Gazebo. The last step is then to replace the simulated robot with the real robot by taking real sensor signals instead of those generated by the simulator. This concept is applied in the educational program and some students were asked about their opinions and experiences. Finally, an extension is presented to the robot built by the students to demonstrate the flexibility of the concept.

*Keywords:* Control Science Education, Virtual and Remote Labs

## 1. MOTIVATION

In this paper we introduce an educational lab concept for control science and robotics that is defined as a combination of a virtual multi-user simulation environment and a corresponding experimental platform.
The main objectives of the development project can be listed as:

- Hands-on experience for students on different levels of design and implementation
- An illustrative and practical example for better understanding of the control science contents
- An easy-to-access platform for every student
- Closing the gap between the theory and praxis of control engineering in the education

Based on the objectives above, the main requirements were defined to be:

- An attractive robot system promoting control science
- Covering a large application field such as different control methods, observers, path planning and following etc.
- A flexible and extendible system allowing future enhancements and innovative student developments
- based on low-cost and available hardware and open-source software

## 2. FROM THEORY TO PRAXIS

The main concern of this project is how to accompany students on the way of their learn process from the theory to the praxis of control engineering keeping always in mind the relation between these two. For this purpose, a three-level concept is defined:

(1) Rapid design and test using idealized simulations
(2) Implementation and test of algorithms on real platforms in combination with simulations
(3) Final implementation on a real system

Hereby, the educational objectives can be achieved in a sequential manner in every step of which the focus is shifted from the theory to the praxis. Below, the concept is explained in detail.

### *2.1 Rapid design and testing under ideal conditions: Simulink*

At the first level, the content of the lectures and different methods are applied and tested under idealized conditions so that the student does not need to deal with several disturbances and uncertainties arising in the praxis. Here, the focus is only on the theory and a visual and intuitive evaluation of the designed control loops. For this purpose, a Matlab/Simulink model is provided that includes the system model and an animation for visualization. Using this model, students are able to rapidly evaluate the control methods and the affects of changing different parameters.

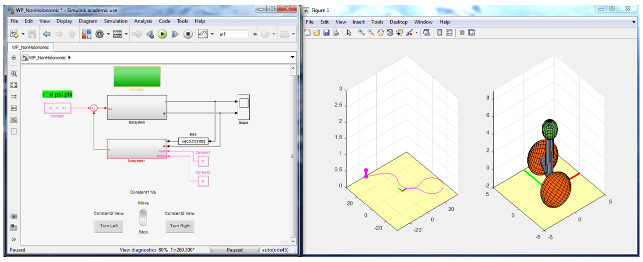
Fig. 1. Simulink-model including animation

At the same time, the animation illustrates the outcome of the control loop that can be understood intuitively. Several tasks such as stabilization, state observing and navigation can be implemented in a simple manner.

In our curriculum for instance, students learn to analyze the system features using Matlab such as stability of an equilibrium, controllability and observability defining different system outputs. Further, they design linear controllers for the input-output transfer function as well as state space controllers by pole placement or LQR algorithm. Finally, the designed controllers are tested and validated applying them the provided Simulink model. Figure 1 shows a screen shot where a linear controller is applied to the system to stabilize and simultaneously follow a commanded path.

The next step after designing the controller and understanding the theoretical aspects is the implementation on a real platform. However, there are many challenges that make a direct transfer from Matlab/Simulink to the real system enormously difficult. On one hand, there are the programming difficulties in lower-level languages such as C++, processing and correct interpretation of sensor data and actuation commands, data-type and rate conflicts etc. On the other hand, the student faces practical challenges of a real system such as parameter uncertainties and asymmetries in the model, unknown or unmodeled effects such as static friction and motor dead-zone, unexpected faults e.g. caused by electronics errors etc. Thus, it is reasonable to split the practical implementation into two further steps to tackle the mentioned challenges in a sequential manner.

At this point, we would like to mention that there exist many different concepts and approaches to lead the step from the controller design to the practical implementation. Only to name some, there are concepts to directly connect Matlab/Simulink to a hardware such as by QUANCER (2012). Offered by National Instruments®, the combination of LEGO hardware and Labview (NI (2017)) is also used very often. These concepts have the advantage of the direct and automatic implementation on the hardware. However, they use licensed software or hardware that are not available at every university or educational institution. The concept introduced in this work uses, except for Matlab/Simulink that is in the most cases available at universities, open-source software and low-cost commercially available hardware. To give a rough estimation, a simple small wheeled robot including a Raspberry Pi 3 and an IMU as in the figure 2 costs around €100 each. As a drawback, after the controller design in Matlab developing the corresponding lower-level code such as C++ and Python is required. A review of the existing educational simulators

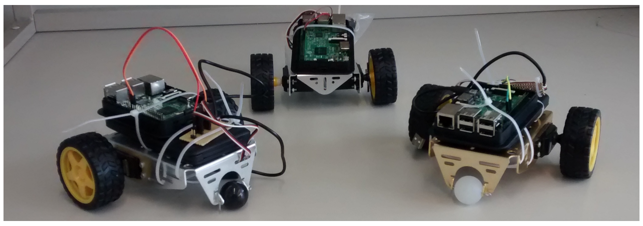
Fig. 2. Simple wheeled robots built by students for educational purposes

is given in J. V. Teixeira (2015). A list of available simulation environments and their features including the one used in our concept can be found on Smashing Robotics (2017). Also, a description of different available hardware and software applied in the robotics education is given in M. Merdan (2017).

On the software side for the implementation, a Linux operating system (Ubuntu) is applied that can be run on (almost) any available hardware and is open-source. The required flexibility and extendability is achieved by using Robot Operating System (ROS (2017)) for task distribution.

ROS is a flexible and open-source framework for programming robots. It can be seen as a task distribution system that splits a large task, for instance stabilization of an electromechanical system, into several subtasks or **nodes** such as: *read* the sensor data, *calculate* the controller gain, *write* the control command to the actuators. The most important advantages of ROS that makes it the best choice for our concept are

- Hardware independence: Any node within the ROS network can run flexibly on any available hardware unit
- Large community: A large amount of students, researchers and hobby-programmers use ROS to develop their robotic tasks and exchange their results and experiences on the Internet. This makes possible that often a required piece of code for a certain task already exists. The approved software codes developed by the students at our institute for instance, are at the moment available at:
  *https://github.com/francisc0garcia/suricate_robot*
  **Note** that the material provided on this repository are still under development.
- Inherent communication and data exchange between the nodes.
- Compatibility with a large number of available interfaces

ROS provides a structured communications layer placed over a host operating system. It was designed initially to accomplish specific challenges of large-scale robotic applications at Stanford University and Willow Garage, but over the time, it became a standard and widely used framework around the world (Quigley et al. (2009)). The main components of ROS can be summarized as nodes, messages and topics. Processes that perform computation are called nodes. Messages are strictly typed data structure such as integer, floating point, Boolean and also (personalized) structure types like *odometry* or *pose*. Nodes send and receive data messages by *publishing* or *subscribing* them to

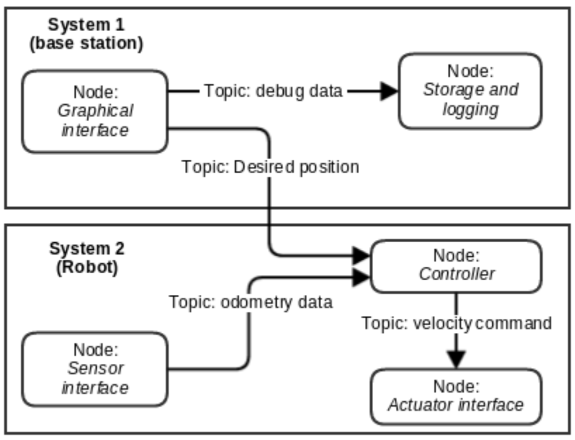

Fig. 3. Schematic structure of an example ROS network

a specific topic. Multiple publishers and subscribers allow large flexibility and scalability of the overall system.

Figure 3 shows a basic example for a ROS network by which a user communicates with a robot using a base station PC to move the robot. The overall task is subdivided into small tasks (nodes) which realized either on base station or an embedded system inside the robot. In this example, graphical interface node allows users to command desired positions. Sensor interface node transmits the odometry from sensors to the controller node which computes a velocity command to drive the wheels.

### 2.2 Programming Almost the Real System: Gazebo

**Gazebo** (Gazebo (2017)) is a free robot simulation tool that uses different physic engines to simulate physical systems as realistic as possible including well-done graphical interfaces.
Gazebo simulates physical systems in a port-based manner connecting subsystems together to build the overall system. We provide the students with the ready-to-use simulation of the robot, however extension of the provided robot or change and replacement of it can be done easily following instructions provided by the Gazebo community. Also, as common for open-source tools, there are already lots of interesting models developed and available on the Internet(Gazebo (2017)).

One of the advantages of Gazebo is that custom plugins can be defined and used to interact with the simulated robot such as different sensors or actuators. By that, realistic components can be defined, for instance to emulate a certain type of sensor installed on the robot. In fact, many available components such as specific IMU-sensors and cameras can be directly emulated since the corresponding plugins are already developed by the community and provided online.
Further, certain plugins allow us to embed the simulated robot into the ROS network such that almost every other software component can be used directly for the real system without any changes. In fact from the point of view of the ROS network, there is no difference between the signals generated by the real sensors and those generated by the emulated sensor as a Gazebo plugin. This way, a large part of the implementation task can be done without the real

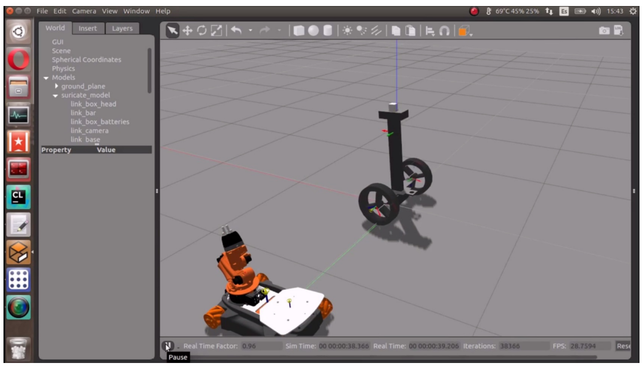

Fig. 4. Gazebo Simulation for the robot Suricate. Another robot is also placed in the world for size comparison.

robot such as signal processing, controller implementation and different visualization and user interface tasks. The controller and other nodes can be tested and evaluated on Gazebo before the simulation is replaced by the real system. Figure 4 shows an screen shot of the simulation world provided.

As processing platform for the practical implementation, commercial mini-computers were chosen that are low-cost and widely available such as **Raspberry Pi** (Pi (2017)).

The structure of the ROS network used with Gazebo simulation is shown in Figure 5. The controller node as well as the sensor data publishers and the twist subscriber that sends a velocity command to the wheels are running on the Raspberry Pi as in the actual robot. On a PC, the graphical user interface and the joystick node are running to send a desired position to the robot. All of the nodes of course, can also run on a single computer, as the controller in the real robot experiment can run on a local computer too.

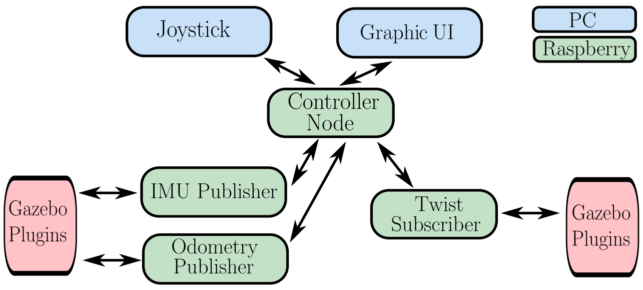

Fig. 5. ROS network structure running the simulation by Gazebo

The two steps above can be done by the students in parallel locally on the private computers without needing any further hardware. After validating the applied software, the last step can be approached to implement the algorithms on a real robot.

### 2.3 Suricate: A Wheeled Inverted Pendulum

The robot *Suricate* (Figure 6) was developed within student projects at the institute of control systems for control education purposes inspired by F. Grasser (2002). It is a combination of the benchmark system, inverted pendulum, and a wheeled robot to allow additional navigation and

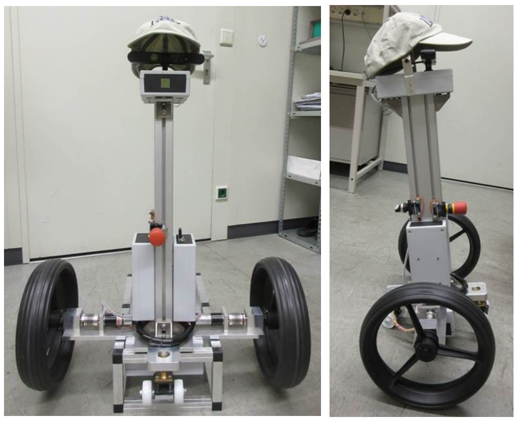
Fig. 6. Suricate Robot

path tracking tasks.
There are many low-cost components used such as Raspberry Pi®, ODROID®, Arduino® micro, ASUS Xition Pro® camera, Adufruit® BNO055 IMU (by Bosch®) and many others.
For the body, standard aluminum profiles are used that allow easy assembling and extension. A power distribution unit provides tow supply lines, one low-power for the processors and sensors and a high-power to supply the motor drivers. A dedicated micro controller (Arduino micro) is in charge of a active switch to turn off the high-power line if necessary due to safety.
Different standard interfaces are used to communicate among components. An overview on the physical connections and interfaces is given in Figure 7.

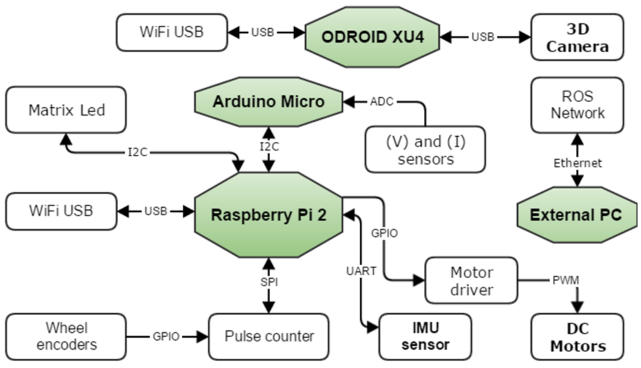

Fig. 7. Physical connections and interfaces

The ROS network is extended by other necessary nodes such as *visual odometry* that calculates the orientation of the robot based on the images received from the camera or *security check* that monitors the state of the batteries and some critical entities to turn off the hight-power line if necessary. However, the basic structure is unchanged and the nodes run with the Gazebo simulation can be directly taken over. Figure 8 illustrates the overal ROS network in the robot.

Although the robot consists of many different parts and components it can be used by the students easily for control educational purposes. In fact, only to verify the controller and observer algorithms developed in Matlab on a real robot the students only need to rewrite their code in C++ or Python. Then, they can directly test their results using Gazebo and if everything works well without any major change on the real robot. Note that the basic structure is unchanged and the nodes run with the Gazebo simulation can be directly taken over. A data logging and monitoring node run on the local computer provides the data after every experiment for post processing and evaluation. A video is generated by the student who developed Suricate and made available on YouTube (LRS (2016)).

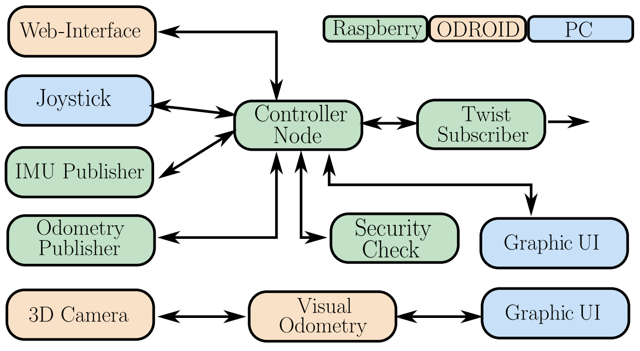

Fig. 8. ROS network structure running on the real robot

## 3. SOME FEEDBACK FROM THE STUDENTS

For evaluation and in order to improve the concept, we asked tome students for feedback on how useful they found this concept, if they have experience with other methods and their suggestions for improvement. Some of them are summarized below.

(1) I have some experience with robot programming, especially using AVR micro-controllers. I also worked with Lego Mindstroms using C. what I like about the combination Gazebo-ROS was the possibility for rapid testing of algorithms and codes. Challenging was the process of getting started with ROS as a new environment. The main difference to other concepts I am aware of is the abstraction of the hardware. The advantage is that the objectives such as controlling can be achieved very fast without taking care of the hardware difficulties.
(2) I have already experience with other platforms such as AVR. Positive was the possibility to figure out the complications in the system before working with the real one. Negative was the new-world of ROS that I needed to get used to firs.
(3) My project with Gazebo and the real robot helped me to experience the effects of simplifications that are often done in theory, for example, linearization and assuming the homogeneous property of the system. About ROS and Gazebo, I like the simple possibility of changing the controller parameter and rapid testing.

In general, students seem to like the concept because of the low-level programming possibility and that the can learn how to easily work with robots in a realistic environment. On the other hand, getting used to ROS seems to be the main challenge. For this, a short introduction on ROS could be helpful since this could be used in many different applications, also later in the engineering profession.

## 4. EXTENSION EXAMPLE: VISUAL ODOMETRY

One of the important aspect of this concept is the extendibility of the practical platform. In a recent student project for instance, the robot was equipped by a 3D camera and a more powerful minicomputer, ODROID. The combination of the camera and ODROID was defined to be the "head" of the robot where also an LED-matrix provides a better appearance. The main purpose of the head within this student project is the estimation of the robot pose using image processing tools. Here, yet supplements from the large online community is used directly or with minor modifications. Taking advantage of the flexible ROS network, the students are able to work on the image processing tasks such as visual odometry, path planning, obstacle detection and avoidance etc separately. This way, two independent student teams can work on the robot, each setting up a ROS network on their physical platform. These can be easily merged or connected by just defining a single ROS-master.

The extension by the visual odometry was done within a separate student project. First, a model of a 3D camera was added in Gazebo by using available plugins. The streamed images from the camera are passed to visual odometry node. After image processing in the visual odometry node, the resulting pose information is published to other nodes such as visualization node.

Based on Lepetit et al. (2009), the algorithm for visual odometry was generally divided into two steps: finding corresponding key points in images for tracking object points and estimation of pose with correspondences. In reality, matching of corresponding points in images is challenging due to factors such as noise in image and motion blur. Fortunately in Gazebo, with an ideal environment a high correctness ratio of found correspondences can be achieved, thus improving the robustness of estimated robot pose.

After the development of visual odometry algorithm was finished, it was tested on the real robot Suricate. The software structure is almost the same as for the simulation, only the model of the 3D camera is replaced by a real camera. During on-line tests, in order to obtain matches with high quality, the raw images were blurred with an image smoother for reduction the high frequency noise, and the corresponding result is showed in figure 9, where the lines bind the corresponding key points in images. If

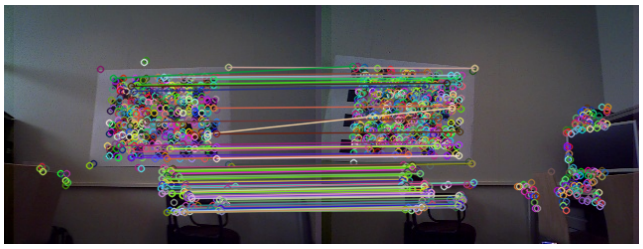
Fig. 9. Matching result with application of image smoother

the matching result is acceptable, the data is proceeded to the second part for pose computation. In contrast to the simulation, the necessary internal camera parameters for pose estimation are unknown here, so they need to be calculated through camera calibration with help of a ROS-tool in advance.

## 5. CONCLUSION

A concept for control educations is introduced based on a combination of a simulation and a real robot. A three-step manner is suggested where students start with a Matlab/Simulink model for rapid design and testing of control algorithms. The practical application is consists of two steps. In the framework of Robot Operating System (ROS), first the students implement the algorithms on an embedded processor, e.g. Raspberry Pi, that is connected to a realistic robot simulator Gazebo. Then the simulated robot is replaced with the real robot by taking real sensor signals instead of those generated by the simulator. This concept is applied in the educational program and some students were asked about their opinions and experiences. Finally, an extension is presented to the robot built by the students to demonstrate the flexibility of the concept.

## REFERENCES

F. Grasser, Aldo D'Arrigo, S.C.A.C.R. (2002). Joe: A mobile, inverted pendulum. *IEEE transactions on industrial electronics*, 49(1).

Gazebo (2017). Gazebo simulation. URL `www.gazebosim.org`.

J. V. Teixeira, M.S.H. (2015). Educational robotic simulators: A systematic literature review. *Nuevas Ideas en Informtica Educativa.*

Lepetit, V., F.Moreno-Noguer, and P.Fua (2009). Epnp: An accurate o(n) solution to the pnp problem. *International Journal Computer Vision*, 81(2).

LRS, U.K. (2016). Suricate: A wheeled inverted pendulum. URL `www.youtube.com/watch?v=Nd9sih3DkKQ`.

M. Merdan, W. Lepuschitz, G.K.R.B. (2017). *Robotics in Education.*

NI (2017). Inverted pendulum with lego nxt and labview. URL `www . k12lab . com / articles / inverted-pendulum`.

Pi, R. (2017). Raspberry pi. URL `www.raspberrypi.org`.

QUANCER (2012). Linear inverted pendulum experiment for matlab/simulink users. URL `de . scribd . com / document / 246377111 / Linear-Inverted-Pendulum-Workbook-Student-1`.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Mg, A. (2009). ROS: an open-source Robot Operating System. *Icra*, 3, 5. doi:http://www.willowgarage.com/papers/ros-open-source-robot-operating-system. URL `http://pub1.willowgarage.com/{~}konolige/cs225B/docs/quigley-icra2009-ros.pdf`.

ROS (2017). Robot operating system. URL `www.ros.org`.

Smashing Robotics (2017). Most advanced robotics simulation software overview. URL `www . smashingrobotics . com / most-advanced \ -and-used-robotics-simulation-software`.