



Robots in the Classroom

Mobile Robot Projects in Academic Teaching

Jörg Roth^(✉)

Faculty of Computer Science, Nuremberg Institute of Technology,
Nuremberg, Germany
Joerg.Roth@th-nuernberg.de

Abstract. The field of mobile robotics has a long tradition and due to recent developments, we expect a huge potential for the future. Expertise in the area of mobile robotics is important for computer science students. The topic has many connections to different other computer science areas such as computer vision, algorithms, planning, world modeling and machine learning but is also related to basic fields such as mathematics, geometry, statistics, graph theory and optimization techniques. Students have to consider hardware and real-time issues, meanwhile they have to deal with uncertainty of data that are based on sensors. Software development differs from typical desktop, office or client-server developments that often are in focus of computer science studies. As a consequence, academic courses on mobile robotics differ in many ways from other courses. A major issue: we have to get the students very quickly to a point to achieve progress in their projects. Moreover, the teacher has to have an environment to manage the different facets of the complex topics.

Keywords: Mobile robotics · Robot platforms · Robot simulation · Academic teaching

1 Introduction

This paper presents ongoing efforts to integrate mobile robot projects in the academic curriculum of computer science degrees programs (Bachelor and Master). The Nuremberg Institute of Technology is of type University of Applied Sciences. This means, its basic task is teaching whereas research, even though strongly encouraged, appears in second place.

Mobile robotics will play a major role in future society and industry. The applications range from household and service robots, medical robots, health care to industrial transport systems and autonomous driving. Besides general development of mechanical platforms, sensors, actuators and computing hardware, we experience great advances in the area of machine learning. For future computer scientist it is essential to have fundamental experience in the area of robotics to be prepared for future tasks.

The computer science degree programs provide a broad education in different areas. Thus, the topic mobile robotics often is an area of specialization. As a consequence, this topic usually is not represented in the student's program mandatory field. At the

Computer Science department of the Nuremberg Institute of Technology the field of robotics can be covered in

- compulsory elective subjects,
- student's projects,
- Bachelor and Master theses.

This means, the topic either occurs in the later part of the studies or in the Master's program. This is reasonable, as mobile robotics require competence in e.g. mathematics, data structures, algorithms, graph theory and programming.

A supervisor of courses on mobile robotics has to deal with different challenges. We identified the following areas:

- *Technical*: A robot platform has to be made available. This includes the actual robot (hardware), but also the environment to develop, test and upload software.
- *Didactic*: The large area of robotics must be tailored to a special amount of knowledge that fits into the respective program.
- *Organizational*: This includes the administration of the (costly) material, service, upgrading, but also supervising safety procedures.

In this paper we describe the experience of several courses in the area of robotics over many years.

2 Robots from the Technical View

2.1 The Robot Platforms

The integration of mobile robots in our courses started 2013 with the *Carbot* robot ([10, 12], Fig. 1, top, left). The main goal was to gather experience with a driving (car-like) robot. In 2017 we introduced a second, legged platform, the hexapod *Bugbot* ([11], Fig. 1, top, right) that additionally makes it possible to address 3D motion problems and gait execution.

The experience with mobile robots in the teaching area actually started some years earlier in 2010 with a first course. At this time, we used Lego NXT sets [1] to teach knowledge about wheeled robot construction and programming. Even though, the courses were a great success from the perspective of students, it was disappointing concerning the richness of contents. The reasons were:

- The mechanical platform mainly consisting of plastic material prevents the construction of precise and stable robots that are able to execute more complex tasks such as exactly driving planned trajectories.
- The sensors supplied with the sets are mainly of simple type (e.g. switches, ultrasonic). The respective output does not enable detailed world modeling.
- The computing power of the so-called 'bricks' (i.e. the computing components) was too weak to execute typical robotics tasks such as sensor data fusion or navigation.

The goal was thus in 2013 to introduce a new robot model, the *Carbot*. We gave up the ambition to make the robot construction as a part of the course. As this is typically a

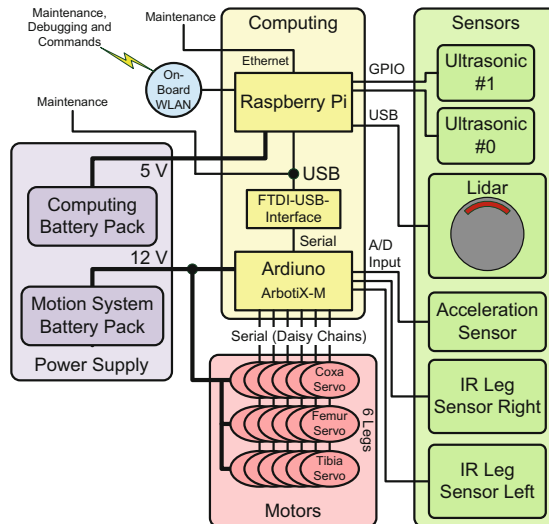
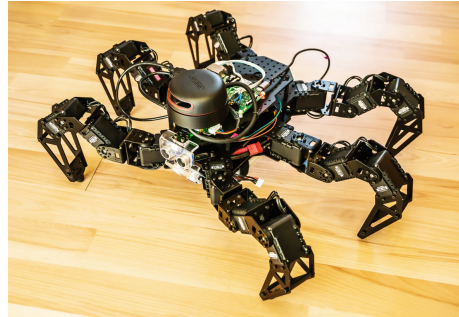
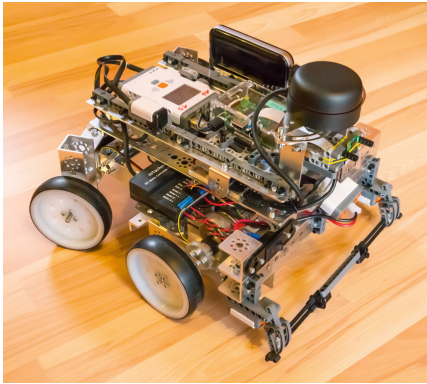


Fig. 1. Carbot (top, left) and Bugbot (top, right), Bugbot architecture (bottom)

mechanical engineering tasks and does not actually enrich the computer science skills, this was reasonable. Thus, the Carbot construction was completely predefined concerning the mechanics, motors and sensors. The students only should extend and modify the software components.

For the Carbot construction we relied on the Tetrax kit – it contained stable metal bars assembled with screws. In addition, strong driving motors, servos and motor controllers were part of the kits. The specific construction was a unique set up, entirely developed in our faculty. For the legged robot Bugbot we used the widely available Trossen Phantom Mark III kit that mainly contained a body, the hexapod legs and a motion control system based on Arduino. This kit provided an empty mounting plate, where we added additional computing boards and sensors.

Figure 1 (bottom) shows the architecture of the Bugbot. One goal was to have fully independent robots concerning power supply and computing power. Both robots carry

two batteries – one for the motors, one for the computing board. As the basic communication link the robots install a WLAN access point at startup. The developer’s PC or notebook can login to the respective WLAN cell to install new software, debug the robot’s software or send task descriptions, e.g. to drive or walk to a specific location.

Both robots have two computing components. One is for higher level tasks such as sensor processing, world modeling and path planning. Currently, this is a Raspberry Pi. The second computing component called the *Motion Subsystem* is represented by a smaller computing platform (from the viewpoint of computing power), but with more capabilities concerning motor control. On the Carbot, we use a NXT brick, well-known from former robot projects. On the Bugbot we use an Arduino that meets the embedded computing demands to operate the 18 servos (three for each of the six legs). The operation programs on the Motion Subsystem computing components are considered as fixed and are usually not subject to changes for e.g. a specific robot tasks. A control interface accepts motion commands (e.g. drive a curve, walk straight) that are autonomously executed. For technical reasons, also some of the sensors are attached to the Motion Subsystem.

Both robots contain a lot of sensors on board. The Carbot is able to map the world with the help of a camera in driving direction and a 360°-Lidar (Light Detection and Ranging). Quick obstacle detection is provided by Ultrasonic and Tactile sensors.

The Bugbot also carries a Lidar and is able to perceive small distance with two Ultrasonic sensors, mounted in different angles. To avoid falling downstairs, the front legs measure the distance to the bottom with two Infrared distance sensors. Both robots use an acceleration sensor to detect tilt angles, collisions and unexpected movements (e.g. falling).

2.2 Simulation Environments

At the time of first course that used the Carbot robot, it was obvious that the hardware was the critical resource. Even though we had three identical Carbots, it was critical to integrate them too careless into the student’s work. We quickly gave up the idea of weekly schedules and ‘robot hours’ to grant access to the three robots for testing. As the schedules had to be synchronized with battery loading times (some hours), the respective slot for testing for a complete class (e.g. with 18 students in 6 groups) would have been too small.

It turned out that an appropriate simulation environment was unavoidable [7]. This should easily be installed on the lab PCs, student’s notebook or home computing environment. The real robot should only be used for final testing and result presentation. We had great demands: the developed software should both run on real robot and simulator without modifying the code, even without the need of recompilation. Moreover, simulated motors, sensors and execution environments should be very close to real facilities, concerning timing, precision, errors etc.

Table 1 presents reasons to use a simulation environment in classrooms. Even though we have numerous benefits, we had significant efforts for developing the simulators. Because the respective robot platforms are highly specialized, it is not easy

Table 1. Ten reasons for robot simulation

Reason	Description
1. Convenience	The simulator is very convenient. We may execute runs at any place with a notebook.
2. Valuably of the hardware	Robots are costly (i.e. Bugbot more than 3000€). Some parts are unique or are not produced any more. Assembly of parts is costly in terms of human resources.
3. Number of robots	We have too few robots. In larger courses with e.g. 30 Students it is not possible to provide enough platforms, even if we form groups.
4. Complexity of usage	Real robots are difficult to handle. They are no consumer products. E.g. startup and shutdown require several steps.
5. Danger for users and environment	Even though our robots are no ‘killing machines’ their motors have a reasonable strength. They can easily harm persons and damage things.
6. Danger for robots themselves	Robots can easily damage themselves, e.g. uncontrolled servo motors can exceed joint limits or break cables.
7. Runtime preparations	The robot often is not prepared when needed. Typical problems: batteries are not loaded, software not updated or sensors not repaired.
8. Serving incomplete robot functions	Real robots only run, if <i>all</i> functions run stably, from sensor processing to action planning. Thus, we can only start with a new idea, if all was available. Simulators can replace unavailable functions by their ‘ultimate knowledge’. E.g. if real SLAM is currently not working, we can assume an ultimately precise (virtual) SLAM using the simulator’s internal robot position.
9. Difficulty of modifications	Modifications of the real robots are difficult. In the simulator we can e.g. add new sensors or change the robot’s geometry with only few lines of code.
10. Setting up test environments	To create real test environments for robots is costly and requires a lot of physical space. In the simulator we can e.g. build large mazes or climbing courses with convenient configurations.

to use a simulator of the shelves – the configuration effort would be very high and some functions would even not be possible to realize. We thus fully relied on own developed simulators.

Figure 2 shows a screenshot of the Carbot simulation environment. The main window contains a ground map with the virtual environment and robot, in addition we see the virtual camera image and Lidar scan. Figure 3 shows the more complex Bugbot simulation environment. In contrast to the Carbot, the Bugbot is able to walk over small obstacles and climb small steps, thus we need a physical simulation in three dimensions.

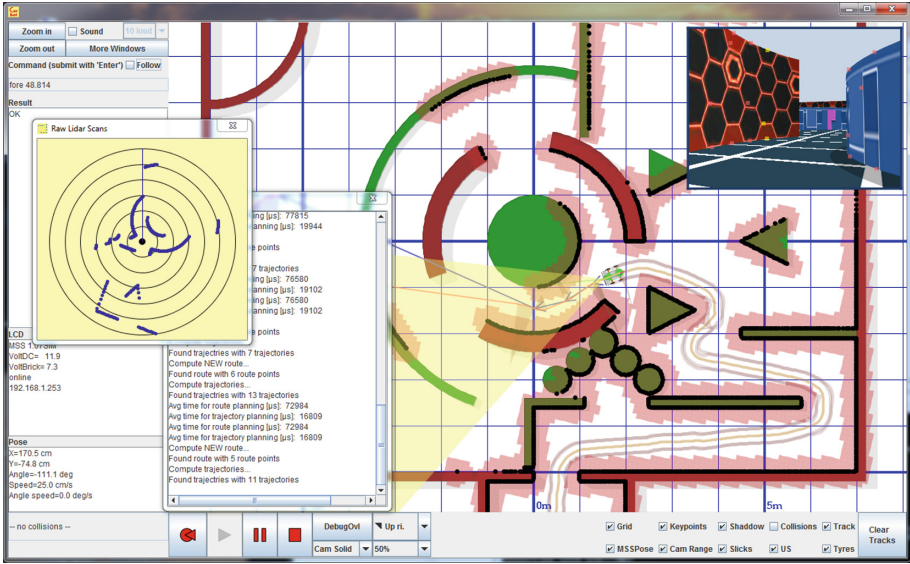


Fig. 2. The Carbot simulation environment

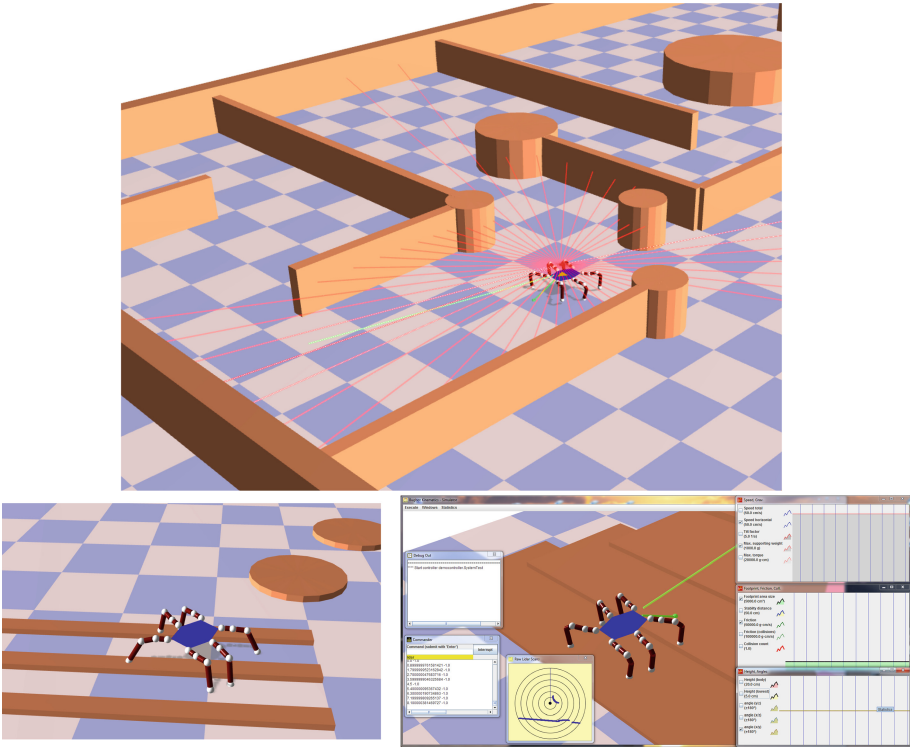


Fig. 3. The Bugbot simulation environment

Both simulators provide the following functions:

- simulation of physical effects, e.g. gravity, friction,
- modeling of virtual environments with obstacles, walls or areas of lower traction,
- scheduling of moving objects that represent, e.g. people who walk through the area,
- simulation of motors and sensors with their specific error and precision properties,
- simulation of virtual cameras – even though also kind of sensors they fundamentally differ in terms of complexity, API and image processing function,
- providing an execution environment for the Motion Subsystem code,
- generating graphical output for the user,
- generating statistics, e.g. measure the virtual damage due to collisions,
- execution of logging and debugging functions,
- state control, e.g. restart the virtual robot, pause a test run.

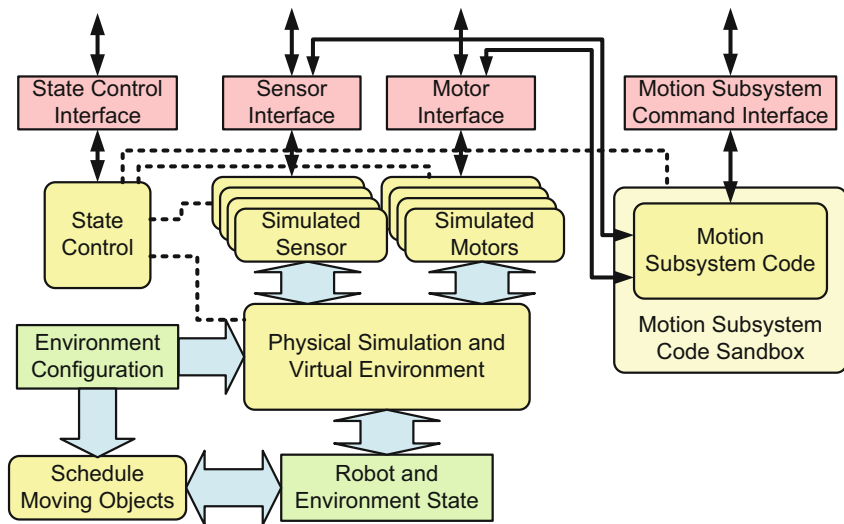


Fig. 4. Structure of the simulation system

Figure 4 shows the simulators' structure. Most important is the physical simulation component. It executes the natural law in the virtual environment, foremost gravity, collision of solid obstacles and friction [5]. It, e.g., lets the Bugbot fall down, if the center of gravity is outside the legs' support polygon. It is also possible to define parts of the bottom as more slippery (e.g. like ice surface) to evaluate, how good the trajectory regulation works. Until now we use an own development for the physical simulation component, but a current student's project tries to replace it by *Bullet Physics Engine* [3].

We simulate the motors on a very fine-grained level of single motor steps. To access the motors on the real environment in the same ways as in the simulator, we make use of the motor interfaces: on the Carbot we have I²C commands, on the Bugbot a serial protocol to control motors. With sensors we deal in the same way, whereas the

way sensors are attached is more heterogeneous. They are attached, e.g. via I²C bus, GPIO, A/D ports or USB. Thus, for each sensor, a software driver is realized that maps access commands to the hardware or simulation facility. This again ensures same control code between real and simulated robot.

The most complex ‘sensor’ is the HD camera. As it is too difficult to simulate each access command of the respective hardware interface, we developed a high-level driver to read the image stream from the real or virtual camera.

It is also very complex to embed the Motion Subsystem code into the simulation environment. The problem: both Motion Subsystem computing components have their operating system and runtime libraries – on the Carbot it is *NXJ* (a Java environment for the Lego NXT brick), on the Bugbot it is *Arduino*. In the simulation environment we thus have to reimplement all respective calls and provide equal functions. This is performed by the so-called *Motion Subsystem Code Sandbox* – also an own development.

2.3 Runtime Modes and Debugging

Debugging robot programs is crucial. Typical debugging mechanisms, such as single step execution or break points are not useful due to the multithreading and event-based nature of typical robot programs. Useful debugging information often is not a single variable value but has a geometric nature, e.g. measured distances, positions, world models, grids or visibility graphs.

As a consequence, we integrated two debugging mechanisms in the platform: *debug out* that is a traditional text-based log and *debug painting*. The latter allows the developer to paint simple graphics with coordinates in world dimensions (e.g. cm scale) on a 2D canvas. In the simulator, this output is directly painted on the 2D environment map. In Fig. 2, e.g., we see the occupancy grid painted on the ground map. The debug painting facility is very useful to get an impression of the robot’s current knowledge of the world and significantly simplifies debugging of control code.

The usage of debugging facilities is different in the three *runtime modes* (Fig. 5). The *robot controller* is the actual robot application, e.g. exploring the environment or navigating from point to point. In the *simulation mode* the robot controller and execution environment run in the simulator, all together run in a desktop application. From the view of software components, this is the ideal case as all software components run in the same memory.

Running the robot controller on the real robot, we have two cases. We still may use the desktop application for debugging. All debugging calls are redirected through a *debug bridge* that transfers the respective calls via WLAN to the desktop application. The developer can still see all debugging information on a convenient workplace, however, further data such as the (simulated) environment map is not available.

The last mode called *native mode* is the actual mode for real robots. Debug painting calls are directed to empty functions and debug out is passed to a text file for later analyses.

For all modes we have a component called *commander*. This allows a user at runtime to specify the robot’s tasks. E.g. a user could specify the position, the robot should drive or walk to. For this, the user enters a command that indicates the

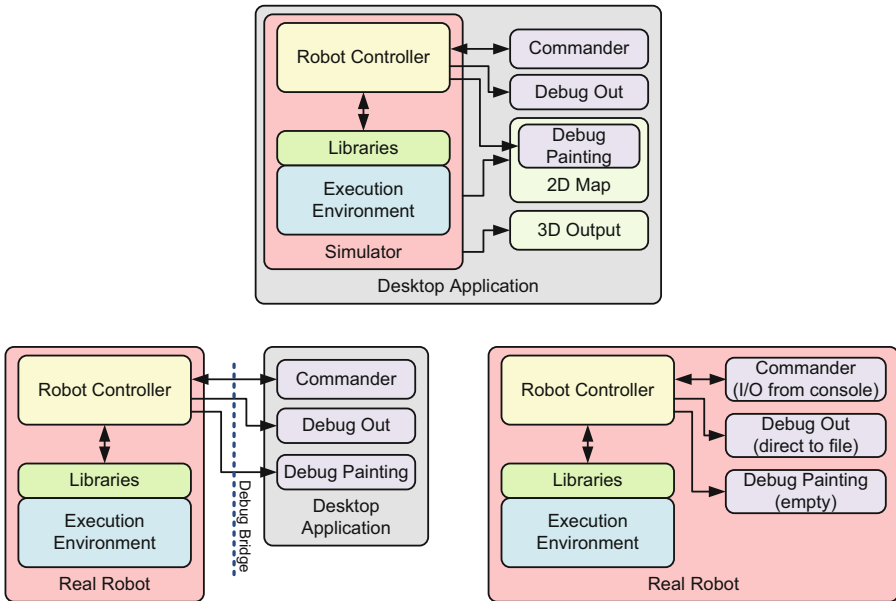


Fig. 5. Runtime modes, simulation (top), real robot with debug bridge (bottom, left), native mode (bottom, right)

movement target and the robot agrees replying ‘OK’ or disagrees, if the target was out of the reachable area. For more evolved robots, the commander could later be expanded by a convenient user interface, e.g. a touch panel – currently a command-line oriented way is sufficient. Like the debug calls, the commander may communicate with the robot controller in memory, via the debug bridge or, in native mode, via the operating system’s console streams.

In robot courses, the commander with its text-oriented nature is ideal for the lecturer to define test cases for the robot controller. They can e.g. be stored in text files (passed via clipboard) or even can be passed by command line at startup.

3 Robots as Subject of Teaching

3.1 Types of Courses and Contributions

The Carbot and Bugbot platforms were created to form an environment for robot research and to teach robotics skills in classes. As a university of applied sciences has teaching as primary task, a professor often has to combine research projects and student projects. Due to a lack of mid-level academic positions (e.g. PhD students) we have to shift some research tasks (e.g. implementing ideas) to students.

In the teaching area, we want to pursue the following goals with our robot platforms:

- They should serve for different course formats, e.g. elective subjects, Master theses.
- We want to transmit knowledge in different areas of mobile robotics such as computer vision, navigation, path planning, world modeling, kinetics [15], sensor data fusion, SLAM (simultaneous localization and mapping) [4] and machine learning.
- The students should get very quickly to a point to achieve progress in their projects. This is an important demand as the (short) time is limited by the respective course formats.
- The students should be able to fade out problems that are not part of their project.

The last two points are crucial. Robots are complex and combine a lot of different components. For a student it is very demanding to get knowledge about all these components in sufficient depth in the short time of a course. E.g. a student should prepare a Bachelor thesis in only five months. In this time she or he should read literature, prepare an approach, implement and test it and write the thesis. For a certain topic inside the robot project, the student should start immediately, without to get distracted by robot components that are not in the focus. Moreover, it should be possible to assume near-optimal performance of these components to evaluate the own work. To give an example: if a student should implement a new navigation algorithm, it is difficult to also consider obstacle detection from camera images. Obstacle detection on real robots often causes errors. As a result, the robot sometimes assumes obstacles at wrong positions. The correction of wrong obstacles would overload the student’s project, but they would significantly falsify the evaluation of the navigation approach. This again shows the benefit of the simulation environments. The simulator can provide theoretically correct obstacle detection, as we can put the virtual sensors into error-free mode. In addition, the self-localization can also be set to be optimal. As a result, the students are able to separately test desired effects of the respective approach.

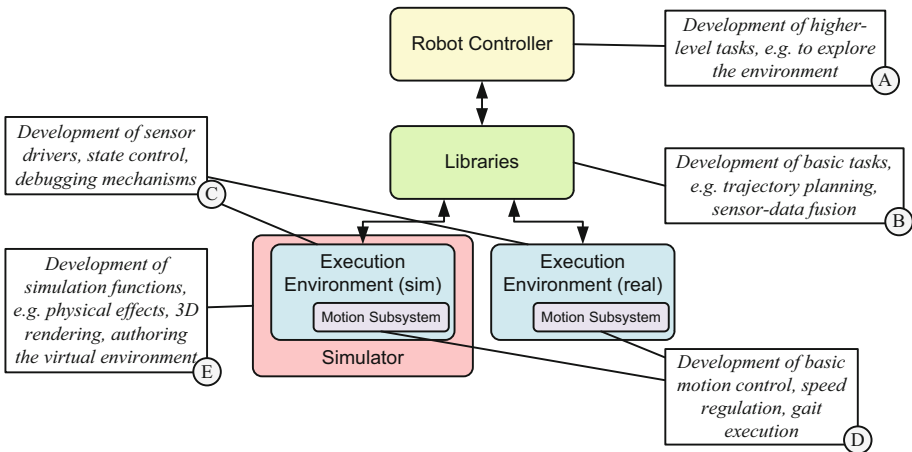


Fig. 6. The robot’s software components and potential tasks for students

Figure 6 shows the software components that may be topic of a student's project. The robot controller was already described in Sect. 2.3. From the development point of view it provides a certain service but strongly relies on libraries and is firmly embedded into the runtime environment.

The *Libraries* are a collection of general-purpose functions for mobile robots. They are permanently extended by new projects. Currently, the following functions and algorithms are covered:

- mathematical basics often used for mobile robot projects, e.g. matrix computation, equation systems, zero points of functions, quaternions, Eigen values, covariances;
- geometric and graph functions also often used for mobile robots, e.g. Voronoi diagrams, visibility graphs, intersection of geometric primitives, clothoids, polygonal operations;
- navigation and path planning, e.g. grid based A* [9], trajectory planning, trajectory regulation [13];
- gait modeling and execution, e.g. Tripod, Ripple, Ample gait [14];
- world modeling, e.g. point clouds, spatial indexing [8];
- localization and mapping, e.g. ICP [2], Hector-SLAM [6];
- robot runtime access and life cycle control.

The *Execution Environment* contains access drivers, serial protocols and basic platform services such as logging and debugging. It also contains the *Motion Sub-system*, described in Sect. 2.1. Finally, the *Simulator* maybe subject of extensions by students, e.g. the physical simulation engine can be extended to simulate more effects.

The respective contributions (A)–(E) (Fig. 6) are distributed among the types of courses. At our faculty we have:

- Master thesis: 8 months, single student,
- Bachelor thesis: 5 months, single student,
- Student's project: 12 months, 3–5 students,
- Compulsory elective subject: 15 weeks, 15–30 students.

Table 2 shows the courses of the last years. We see (C) and (D) only rarely occur, as they are very special to our platforms and do not transport general knowledge about robotics. In addition, they are 'mission critical' – errors or problems would significantly hinder future projects. Thus until now, these components are usually developed by the teaching staff and considered as stable.

Contribution of type (A) is ideal for elective subjects over a single term. The benefit: the students build their software upon a stable and well-documented API. In particular they can fade out the structure of the platform and internal components. It also is possible to let the students re-implement available functions that already were integrated into the Libraries. E.g., in one course, the students had to realize a Robot Controller which was able to navigate through unknown environment. As this function already was realized, we deployed a reduced library that did not contain navigation facilities.

Theses mostly were of type (B). Here, we heavily made use of students' projects that also advanced our research – the students often realized new ideas or existing approaches as a basis for comparison with our research ideas.

Table 2. Overview of robot courses

Subject	Type of course	Contribution
Visual SLAM	Master thesis	Carbot (B)
Trajectory planning with clothoids	Master thesis	Carbot (B)
Probabilistic path planning	Master thesis	Carbot (B)
Visual obstacle recognition	Bachelor thesis	Carbot (B)
Vector-based navigation	Bachelor thesis	Carbot (B)
Trajectory regulation	Bachelor thesis	Carbot (B)
Feature extraction	Bachelor thesis	Bugbot & Carbot (B)
Efficient point cloud data structures	Bachelor thesis	Bugbot & Carbot (B)
Incremental navigation	Bachelor thesis	Carbot (B)
Positioning with ISOVIST features	Bachelor thesis	Bugbot & Carbot (B)
Grid-based Lidar-SLAM	Bachelor thesis	Bugbot & Carbot (B)
Kinematics simulation for arthropods	Bachelor thesis	Bugbot (E)
Dynamic simulated obstacles	Bachelor thesis	Carbot (E)
Robot motion building blocks	Student's project	Carbot (D)
Visual processing	Student's project	Carbot (C)
Robot navigation	Student's project	Carbot (B) & (E)
Simulation of a 3D camera view	Student's project	Carbot (E)
Integration of bullet physical engine	Student's project	Bugbot (E)
Programming a navigation tasks	Comp. Elect. subject	Carbot (A) & (B)
Programming an exploration tasks	Comp. Elect. subject	Carbot (A) & (B)

Type (E) also rarely occurs. Similar to the runtime environment, the simulator is a critical resource. Moreover, it is more difficult to identify appropriate internal interfaces. These however are required to define students' projects in a way to made them assessable later. Until now we only have the replacement of the rendering engine and physical engine as project of this type.

3.2 Lessons Learnt

According to our faculty rules, each professor has to evaluate her or his course from the didactical view. However, we did not get enough quantity for meaningful statistics about the special approach. In particular, the evaluation mainly covers teaching qualities (e.g. quality of textbook or slides), not the actual topic of mobile robotic. In addition, it is very difficult to compare the students' feed back between different types of course and the different types of contribution (A)–(E). Finally, even though we get a large total number of projects, the number for a certain combination still is low. Nevertheless, we got a good impression. In this section we will discuss some lessons learnt.

The simplicity of deployment and installation of the simulator was appreciated by students. If the project was to develop a robot-controller (contribution type (A)), the students just have to copy a single binary file (of type .jar) and can start their

development and tests. Some students criticized the demands of the tools concerning computing power – actually older notebooks (often used by students) sometimes were too weak to smoothly run the simulator.

None of the courses generated noticeable problems with the robotic topic. In elective subjects all students successfully finished their project, however with different levels of software quality. As a large time of development was performed with the help of the simulator, the students welcomed the flexibility. In particular, the programming could easily be done in groups, partly at home, without waiting for access to the real robot. Also the debugging tools were positively received. Typical robot programs contain a number of threads without a traditional windows-based user interface. Thus, in particular the debug painting facility was a great benefit.

Debug painting is also used for the lecturer to evaluate a student's work. This also is usually very difficult, as the quality of a robot component can either only be evaluated as black box or by code inspection, both with certain drawbacks. If the lecturer requires to students to produce meaningful debug painting output, it is easier to get insights into the student's project and the involved data structures. However, students sometimes complained about the demand of meaningful debug painting output. This is, because this function does not belong to the application task in a closer meaning. Actually, on the real robot, there is no debug painting facility at all – the respective calls are redirected to empty procedures. Thus, debug painting usually means extra coding for the students.

From the teacher's view, the simulation tool also is a great facility. There are longer phases where the students can work without explicit supervision. This in particular is a great benefit, if we think about safety issues. Also costs of damaged parts or even completely damaged robots are an issue. In elective subjects, it is possible to only conduct final tests and presentations on the real robot – which then can easier be supervised by the lecturer.

It also turned out, we strongly benefit from a huge part of own developments. In the stressful time of a course, each interrupt is always a problem for students as well as for the lecturer. E.g. during the period of a Bachelor thesis, the student should not lose time because of technical problems. Deadlines are very tough. Thus, it is a great benefit to solve technical problems in a close loop. This would not be possible, if we strongly relied on third-party software, even if it was open source. Own developments also enable quick adaptations (even though sometimes 'quick and dirty') as a result of unpredicted new demands. These could be a result of adapting projects due to new insights after the course already has started.

The students also appreciated that the simulation may fade out real-world errors during testing and development. This allowed the students to first concentrate on their project's task. However, one student remarked that the navigation project of an elective subject was simplified too much and reminded to 'game development'. As a result, the lecturer must carefully decide, how to fine-tune the project and environment settings to meet the course demands.

Besides the benefits of the described approach, we also have to consider some drawbacks. The major drawback: the development of robots, runtime environments and in particular the simulators were very time-consuming. Until now we coded an amount of approx. 200 000 lines of code, of which only a small part of approx. 10% was

contributed by students. Whenever students contribute, a lot of post-processing was required. This was, because the students often do not achieve the level of software quality as required when contributing to long-term libraries. Often, formal corrections were necessary concerning package structures or naming of packages or classes. As student projects usually run parallel, the platform evolved by one project before finishing another. Thus, a costly merge of versions was required. As a rule of thumb, students' contributions are more crucial when 'low-level' (i.e. (D) and (E)) – such work should be avoided as much as possible for student projects.

4 Conclusions

Teaching mobile robotics skills still is a demanding task. Due to the amount of connected fields, it is a challenge to get the students quickly to a point to achieve progress in their projects in the short time of an academic course. Our mobile robots Carbot and Bugbot with their environments form an appropriate basis for a lecturer to offer respective courses. In particular the simulators keep the technical, organizational and safety demands as well as costs under control. Moreover, they help the students to quickly start with their development and keep the hurdle low for first successful experiences with robot programming. Further testing facilities such as debug painting help the students to debug their programs and help the teacher to assess projects.

Dependent on the respective teaching targets, some tasks are still demanding, e.g. development concerning the robot's execution environment, sensor drivers or simulators. Here teachers have to bear serious integration efforts to keep the entire platform stable and manageable.

In the future we want to extend the platforms to more support deep learning technologies. Currently, most of the robot functions are realized in a traditional manner, in particular functions related to sensor data interpretation. We expect many of these functions will be provided by deep neural networks in the future.

References

1. Bagnall, B.: Maximum Lego NXT. Variant Press, Winnipeg (2012)
2. Besl, P.J.: A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell.* **14**(2), 239–256 (1992)
3. Boeing, A., Bräun, T.: Evaluation of real-time physics simulation systems. In: 5th International Conference on Computer graphics and interactive techniques in Australia and Southeast Asia, pp. 281–288
4. Borenstein, J., Everett, H.R., Feng, L.: Where am I? Sensors and methods for mobile robot positioning. *Univ Michigan* **119**, 27 (1996)
5. Dupont, P.E.: Friction modeling in dynamic robot simulation. In: Proceedings, IEEE International Conference on Robotics and Automation, Cincinnati, 13–18 May 1990
6. Kohlbrecher, S., von Stryk, O., Meyer, J., Klingauf, U.: A flexible and scalable SLAM system with full 3D motion estimation. In: 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, Kyoto, Japan, 1–5 November 2011

7. Pepper, C., Balakirsky, S., Scrapper, C.: Robot simulation physics validation. In: PerMIS 2007 Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems, Washington, D.C., 28–30 August 2007, pp. 97–104 (2007)
8. Pomerleau, F., Colas, F., Siegwart, R.: A Review of Point Cloud Registration Algorithms for Mobile Robotics. Now Publishers, Hanover (2015)
9. Roth, J.: Navigation durch Flächen. In: 13. GI/ITG KuVS Workshop on Location-Based Application and Services, Jena, Germany, 22, 23 September 2016 (in German)
10. Roth, J.: Carbot, Internal Technical Reference, Nuremberg Institute of Technology
11. Roth, J.: Bugbot Kinematics, Internal Technical Reference, Nuremberg Institute of Technology
12. Roth, J.: A novel development paradigm for event-based applications. In: International Conference on Innovations for Community Services (I4CS), Nuremberg, Germany, 8–10 July 2015, pp. 69–75. IEEE xplora (2015)
13. Roth, J.: A viterbi-like approach for trajectory planning with different Maneuvers. In: Strand, M., Dillmann, R., Menegatti, E., Ghidoni, S. (eds.) IAS 2018. AISC, vol. 867, pp. 3–14. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-01370-7_1
14. Roth, J.: Systematic and Complete Enumeration of Statically Stable Multipod Gaits, under review
15. Yoshikawa, T.: Foundations of Robotics: Analysis and Control. MIT Press, Cambridge (2003)