

Teaching children a programming language with robots

MAGISTERARBEIT

zur Erlangung des akademischen Grades

Magistra

im Rahmen des Studiums

Informatikmanagement

eingereicht von

Lisa Vittori

Matrikelnummer 9627042

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze
Mitwirkung: Dipl.-Ing. Lara Lammer

Wien, 01.04.2015

(Unterschrift Verfasserin)

(Unterschrift Betreuung)

Teaching children a programming language with robots

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Magistra

in

Computer Science Management

by

Lisa Vittori

Registration Number 9627042

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze

Assistance: Dipl.-Ing. Lara Lammer

Vienna, 01.04.2015

(Signature of Author)

(Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Lisa Vittori
Hannah-Arendt-Platz 9/8, 1220 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)

(Unterschrift Verfasserin)

Abstract

Robots have been considered in teaching programming, because they are seen as a vehicle to motivate younger students and introduce them to programming and engineering principles, especially computational thinking (abstraction, generalization, algorithm, modularity, decomposition and problem solving). When teaching introductory programming two main problems are identified in the related literature and studies: One is the loss of motivation, which can be observed in students not doing exercises or dropping out of computer science education or courses, and which is also remarked by students in interviews. Second is high failure rate during exams or failing a whole course. Qualitative analysis of the reasons shows that many students have only surface knowledge of the relevant topics.

Educational robotics seems a good tool to counter these two problems. Therefore I developed a curriculum for introductory programming with educational robotics. Robots give the opportunity to use constructionism or learning by making as a central didactic method. As a leitmotiv throughout the curriculum I use the story of Sasbot - Seek-and-spot robot – which gives the students a goal to reach. Since this goal is far away at the end of the course, I use quick-win situations for students like challenges and small competitions to keep them motivated through out the course. Team work is used as an explicit teaching tool. To counter the problem of having only surface knowledge, I try to introduce every new concept with as much relation to previous knowledge as possible, similar to the idea of anchor graphs and meaningful learning. Since the problem of designing a program is identified to be one of the most problematic concepts by several studies, I show explicitly how to make design steps, which is additionally supported by the tangible environment with the robots.

The concept is intended for introductory programming courses in technical high schools especially with focus on computer science. This leads to the necessity of teaching a programming language that is actually used by companies. Due to this context, I used Botball and the programming language C, because it allows me to focus on basic concepts like variables, functions and loops.

The curriculum was evaluated in a case study with 45 students, who attended a programming workshop with robots consisting of five workshop blocks with three hours each, by analyzing the influence of the two basic problems: The motivation was evaluated with questions at different times of the workshops regarding the ongoing interest and additional qualitative observations of the students' behavior during the workshop times. To analyze the knowledge gained exam-like questions were used and solutions to the challenges and competitions during the workshops were additionally graded. I also supplemented this with protocols of the approaches the students took when solving a particular task.

The evaluation of the workshop shows that the motivation was kept throughout the workshop and especially the challenges and competitions helped raising the motivational level after a longer session of explanations. An improvement of the exam results could not be clearly evaluated because of the differences between the workshop environment and the target context. The indications are promising.

Contents

1	Introduction	1
2	Didactic methods and educational considerations	3
2.1	Teaching programming	3
2.2	Teamwork and collaborative learning	5
2.3	Didactic methods	6
2.4	Robotics in Education	7
2.5	Summary	10
3	Educational robotic systems	11
3.1	Lego Mindstorms	11
3.2	Botball	14
3.3	Robotino	16
3.4	Additional educational robotic systems	18
3.5	Different approaches in educational robotics	19
3.6	Summary	19
4	Concept	21
4.1	Context	21
4.2	Existing curricula	21
4.3	The Robot	22
4.4	Basic concept	22
5	Curriculum	29
5.1	The Beginning	29
5.2	Program structure	33
5.3	Function calls	37
5.4	Introduction of the first challenges	45
5.5	Defining functions	48
5.6	Functions with parameters	57
5.7	The clearance competition	63
5.8	Using sensors	65
5.9	Loops	69

5.10	The find the spot competition	72
5.11	Summary and outlook	74
6	Evaluation	75
6.1	Method	75
6.2	Results	81
6.3	Discussion	87
7	Conclusion	91
A	Additional material	93
A.1	Running a program on the controller	93
A.2	Sensor screen	98
B	workshop slides	99
	List of Figures	123
	List of Tables	127
	Listings	129
	Acronyms	131
	Bibliography	133

Introduction

Learning a programming language is considered a difficult task and many students attending an introductory programming course loose interest and motivation for enhancing their programming skill after the first few weeks, resulting in high drop out rates. Educators around the world face the same or similar problems [63, 74].

A reason why learning to program is considered difficult lies in the perception that programming is also boring [31]. Experience shows that even interested students can loose their motivation very quickly, when presented with the difficult tasks of learning a programming language. But only highly motivated students spend the time necessary to program exercises, which is considered a key to learn programming successfully by educators [32, 37] and by the students themselves [37, 72].

The problems with introductory programming courses are evident for some time now (cf. [67]) and the main consequences, which are still unsolved on a large scale, are:

1. Loss of motivation in programming courses
2. High drop out or failure rates in exams

Jenkins suggests that these two problems are related because students cannot distinguish if computer programming “is boring because it is difficult or difficult because it is boring [...]But they remain adamant that it is both” [32].

Previous work shows that there is a connection between motivation and learning. Lin et. al. show that a high intrinsic motivation is positively related to grades [41] . So all efforts to enhance motivation should also affect the performance in exams positively. It is also stated, that failure in tasks or exams leads to decreased motivation, so an improvement in school achievements can lead to an enhanced motivation [20].

To counter these problems I propose the usage of a specialized curriculum using robots as the main tool. The curriculum is based on didactic approaches which have already proven to be successful in the problem areas especially with the focus on the usage of robots in the educational

environment. These approaches are analyzed in chapter 2 “Didactic methods and educational considerations”.

As I need to choose one educational robotics system I compare the robotic systems currently used in the light of the given context and requirements for the curriculum in chapter 3 “Educational robotic systems”. The full curriculum is described in the following chapter 5 “Curriculum”.

The curriculum was evaluated in a case study with voluntary workshops for students age 13 – 15. The evaluation method and the results are presented and discussed in chapter 6 “Evaluation” followed by a the conclusion in chapter 7 “Conclusion”.

The Appendix contains additional material for the curriculum as well as the slides used in the evaluation workshop.

Didactic methods and educational considerations

2.1 Teaching programming

According to Kansanen the didactic triangle can be used as a tool to understand subject didactics, especially the didactic relation between teacher and his or her influence of the study behavior and thus the interaction between student and content (Figure 2.1) [33].

Berglund and Lister argue that most approaches in teaching programming focus only on the content part of teaching. While the content part is indeed important, it is necessary to understand that the triangle as a whole needs to be more researched. Especially the relation of the content and the student and the relation between teacher and student are core factors to explain motivation [12].

So to teach programming we need to keep in mind two relations who are important for the students success:

1. The relation between student and the content, where the student needs to have a motivation to learn the content

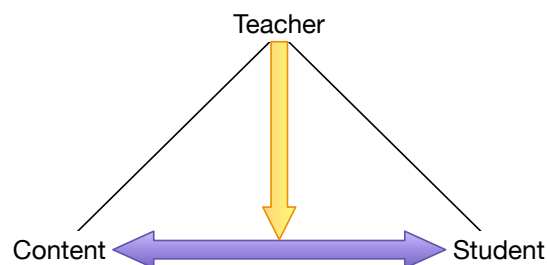


Figure 2.1: didactic relation in the didactic triangle [33]

2. The teachers influence on this student-content-relation allowing the teacher to induce additional motivation.

One way for teachers to enhance motivation and promote interest is in setting reachable and fascinating goals.

Motivation and Goals

Goals play a large role in motivation and promote self-regulated learning [78]. Campbell and Bolker present a successful case study of how immersion into a real world problem and an overall ongoing story line can lead to enhanced student motivation and success. But they also underline their experienced problems with some students who had difficulties with this approach and suggest to have a step-by-step course. [19]

Covington suggests that giving students plenty of achievements and goals they can reach enhances the students' motivation [20]. Roberts reports the enhancement of student involvement and motivation due to smaller challenges and competitions throughout a computer science course including a challenge at the beginning with Karel the Robot simulator [62].

So ideally both approaches - one big story line and small challenges and competitions throughout the course - are included in a curriculum to accommodate the different learning behaviors of as many students as possible.

In the curriculum I developed, I use an ongoing story of a little robot having the duty to find lost things. The implementation of this robot behavior in a small scale is the prime motivator for teaching new programming concepts. Additionally I use small exhibition like challenges and competitions after two three chapters to keep the students motivated.

Knowledge presentation

Aside from the motivational factor knowledge presentation is another key factor influencing the student-content-relationship of the didactic triangle, where the teacher has significant impact. Cognitive research regarding programming instruction shows that learning to program bears several difficulties which are well-known since several years. Especially "Designing a program to solve a certain task" together with "Dividing functionality into procedures" and "Finding bugs from my own program" are considered the most difficult parts in programming [37].

Linn and Dalbey identify three main links for an ideal chain of accomplishment when learning programming:

(a) single language features, (b) design skills, and (c) general problem solving skills.

which are dependent on each other [43]. This means the mastery of general problem solving skills occurs simultaneously with the mastery of design skills. This is very difficult. Linn and Clancy therefore developed a concept, where teaching design explicitly is the key factor [42].

Several studies come to the conclusion that most students have only a surface knowledge of programming [77]. This is equally problematic according to the chain of accomplishment above. Additionally Bransford et. al. state that

To develop competence in an area of inquiry, students must: (a) have a deep foundation of factual knowledge, (b) understand facts and ideas in the context of a conceptual framework, and (c) organize knowledge in ways that facilitate retrieval and application. [14]

So the language features and building blocks must be presented in a way that enables students to link them together and build a strong network of knowledge. Mayer additionally emphasizes the need for anchoring ideas to make the learning experience meaningful. Meaningful learning thereby describes the process in which a new fact is associated with already present facts, so that the new fact is absorbed in the knowledge base of the learners mind [45]. Mead et. al. present *anchor graphs* showing a mean to plan the teaching of anchor concepts in a curriculum in a more coherent way to facilitate the actual cognitive and learning theories as much as possible [48].

In the curriculum I developed, I avoid presenting stand-alone-facts. So first the curriculum establishes a coherent image of the programming process inside the computer related to observations the students made and emphasized with systematic practice games. Then I teach the key programming concepts with as much relation to previous known facts as possible. The path I used is similar as the one shown with the anchor graphs.

Procedural Programming and objects first approach

To counter the problems with program design and limited understanding on the surface teaching an “objects first approach” has become increasingly popular especially with teaching Java [35]. But shortly after a second trend has emerged that postulates going back to a procedural approach [59]. Both approaches are debated intensively in specialized mailing lists and on conference boards without any clear outcome. However many participants in the discussions agree, that teaching objects first needs special considerations due to the difficulties in the approach lying in the complexity of the object orientated concept [7, 8, 16]. In this light it is also interesting that Mead et. al. had more difficulties in developing an anchor graph for object orientation than one for algorithmics [48].

Since the more recent studies suggest a return to the classic method and studies about difficulties show that concepts regarding object orientation are still considered the most difficult concepts (cf. [72]) and in the light of the above mentioned problems I favor the procedural approach.

2.2 Teamwork and collaborative learning

Collaborative learning and teamwork can greatly assist students who struggle with programming and are losing interest and motivation in the subject [73]. McKinney and Denton report additionally that teamwork experiences throughout the course lead to higher course success rates and deeper learning as well as higher interest and having fun. [47]

But teamwork has also some downsides most of all the problem of “free riding”, where one group member prefers not to contribute to the groups efforts. This problem leads to many follow-up problems, mainly the “free rider” might not be able to learn the necessary topics associated

with the project and the workload of the rest of the team is increased [15]. This problem can be reduced with reduction of group size [28].

In the curriculum I developed I use only small group sizes (2 – 3 students) and a special task rotation. I divide each tasks according to standard software engineering steps (design, implementation, test) and give one student the responsibility for one step. After each task the responsibilities are rotated so that each student is required to contribute to the group efforts.

2.3 Didactic methods

Didactic methods have evolved over time but the foundations of most didactic methods which are now widely used, lie several years back. When working with robots *constructionism* seems the logical answer to the question which didactic method to use [4, 36, 69].

Constructivism and Constructionism

Constructionism was developed on the principle of constructivism [60]. The term constructivism is mostly based on the work of Piaget, a psychologist for children and pedagogy. The main differences between the constructivist approach and the dominant learning theories at that time is that knowledge does not exist independently from the learner. Instead the learner constructs his knowledge derived from a searching process, in which he examines, questions and analyses tasks and experiences [6].

Although Piagets work concentrates on the internal structure and building of knowledge consequences for education can be derived:

1. *teaching is always indirect: Kids “don’t just take in what’s being said”. Instead, they interpret what they hear in the light of their own knowledge and experience. They transform the input.*
2. *the transmission model, or conduit metaphor, of human communication won’t do: To Piaget, knowledge is not information to be delivered at one end, and encoded, memorized, retrieved, and applied at the other end. Instead, knowledge is experience that is acquired through interaction with the world, people and things.*
3. *A theory of learning that ignores resistances to learning misses the point: Piaget shows that indeed kids have good reasons not to abandon their views in the light of external perturbations. Conceptual change has almost a life of its own. [2]*

Ben-Ari reminds us that when applying constructivism to computer science it is essential to remember that the students have usually no underlying model of how a computer works. So the students need to construct their knowledge from the ground up. Since constructivism states that misconceptions are essential for the construction of new knowledge not having a model of the computer causes a lot of difficulties [10].

Wulff suggests the following phases of instruction when using constructivist pedagogy for computer programming:

1. *Initial exposure through lecture time or assigned reading*
2. *Brief review (especially when using assigned reading)*
3. *Guided practice activity*
4. *Individual or group programming assignment*
5. *Evaluation of learning achievement (not always necessary) [79]*

Constructionism shares the idea of constructing knowledge with constructivism. But constructionism takes this approach a step further and adds the idea that constructing knowledge happens better when the learner is engaged in “constructing a public entity”. Therefore the simplest way to describe constructionism is learning-by-making. It enables learners to play around and develop their programs in a more creative way. The learner is guided by the way the work progresses and does not need to stick to a premade concept [54]. In such a way the focus of constructionism arises on the individual learner rather than the development of knowledge as a whole [2].

Stager et.al. describe eight big ideas behind the Constructionist Learning Lab which were postulated by Seymour Papert upon creation of the Constructionist Learning Lab [68]:

- *The first big idea is learning by doing. We all learn better when learning is part of doing something we find really interesting. We learn best of all when we use what we learn to make something we really want.*
- *The second big idea is technology as building material. [...]*
- *The third big idea is hard fun. [...]*
- *The fourth big idea is learning to learn. [...]*
- *The fifth big idea is taking time – the proper time for the job. [...]*
- *The sixth big idea is the biggest of all: you can’t get it right without getting it wrong. [...]*
- *The seventh big idea is do unto ourselves what we do unto our students. [...]*
- *The eighth big idea is we are entering a digital world where knowing about digital technology is as important as reading and writing. [...]*

So using robots in a fun way to teach programming fulfill already the first three big ideas of the constructionist learning lab. The challenges and competitions I set for motivation are there to enhance the fun factor and additionally giving the students interesting tasks they can actually *do* themselves. This leads to implementing the sixth idea because the challenges and competitions are designed to be indeed challenging for the students and with the robots having some fault tolerance when applying the movement commands no student will be able to fulfill a challenge without having some problems with it.

2.4 Robotics in Education

Stager identifies five general ways how robots are used in education [69]:

- *Robotics as a discipline*
- *Teaching specific S.T.E.M. concepts*
- *Thematic units (robots as a mean to transport knowledge for the given theme like airports or factories)*
- *Curricular themes (similar to the item above is the central point a real world problem with the enhancement of robots who can bring a solution to the problem)*
- *Freestyle (robots as a tool for self-expression)*

Although all these fields have been reported with successful approaches, most of the available studies have non-experimental character and therefore do not allow for quantitative analysis. Nonetheless there are several studies showing, that the use of educational robotics can increase academic achievement in specific STEM related areas. But there were also papers presenting no significant increase while using robots as a teaching vehicle. Benitti identifies several aspects mentioned in the researched papers making the usage of robots successful [11]. These are among other things:

- Small working groups (2-3 students)
- Tasks the pupils find relevant and realistic to solve
- The role of the teacher (the attitude of the teacher towards the tools they use influence the students perception).

In the curriculum I developed, I teach programming (a skill) via robotics using small working groups and motivating tasks. So the first two of the mentioned success-factors correlate with the already described actions in the curriculum to prevent problems with team work and motivation. My personal motivation in using robots is my fascination with the huge amount of possibilities the robots represent and the growing field of artificial intelligence. Additionally as a teacher I am intrigued by the way students interact with the robots and the fascination of the students.

Robotics and constructionism

Although Papert developed LOGO, a programming language, as a mean to implement his constructionist believes in school teaching [52], teaching programming is mostly not situated in the constructionist principles. Beynon and Roe identify two problematic issues in the constructionism context:

- *extraneous activity – Much of the learning associated with model-building is computer-programming specific: it is concerned with manipulating programming language commands, procedures and parameters rather than with developing knowledge of geometric concepts or abstract thinking strategies;*

- *planning rather than exploration – Classical programming is not conceived as an iterative experimental process: programmers are encouraged to plan and preconceive their application rather than to develop a model in an open-ended fashion where its significance can emerge during the development. [13]*

The appliance of robotics as a vehicle of learning to program can help circumventing these problems. It is already stated in [49] that the use of robots in an simulated laboratory environment is a direct implementation of the constructionist idea by Papert.

The problems a robot has are easily conceivable by students, e.g. how do I make the robot move from point A to B. Stager identified that the mean to engage students in constructionist learning is giving them a good prompt and robots give the students the possibilities to find a field of interest where the learning of the programming language is a byproduct [69]. Therefore the robots give teachers the chance to devise assignments which are derived from the environment in which the students are living and therefore it is easier to develop a meaningful goal for the students.

It is not surprising that Petre and Price observed that the use of robots can motivate children to face problems, which are considered difficult but fundamental to programming and engineering. They identified one motivating thought in “making it work”. [55].

In addition Sullivan identifies three aspects of robotic education that leads to students using their thinking and science process skills used in his research:

1. *the tool-rich nature of the environment*
2. *the immediate feedback built into the system*
3. *the open-ended and extended nature of student inquiry. [71]*

Altin and Pedaste analyzed the didactic approaches which were used in robotics education. Although the approaches base more or less on constructionism the concrete methodology differed. The following successful approaches were identified:

discovery learning, collaborative learning, problem solving, project-based learning, competition-based learning, and compulsory learning [5].

To implement these findings the methodologies I used in the curriculum I developed are

- collaborative learning: in building robot teams who need to solve a task, students are prompted to discuss the themes in a group environment.
- problem solving: the challenges are problems the students need to solve with the presented means or even through the invention or inquiry of new means.
- competition-based learning: to further the motivation for the students to deal with the necessary concepts, competitions are a suitable impulse.

Teaching programming with robots

Major et.al. show that using robotics as enhancement for teaching introductory programming is effective as exhibited in several studies. The most frequent approach is using Lego Mindstorms NXT and Java at the university level. The paper also suggests that there is strong indication that using an simulator as addition to the use of physical robots can provide additional benefits [44].

Black and white box approaches

Shifting from “black box” to “white box” paradigm is one of the current discussion points when teaching robotics or with robots [3]. Resnick et. al. make a strong case for using a white box approach as it leads students to “begin to view scientific investigation as a process in which they can take part, day to day, creatively and pleasurably” [61]. But as Kynigos shows sometimes compromises have to be made to enable children to engage in meaningful and challenging activities resulting in a “black-and-white-box” approach [36]. Detiskas and Alimisis give an example in using a pre-build robot when focusing on teaching programming [23].

2.5 Summary

In my curriculum I combine constructionist elements with a story-driven approach to give the students a meaningful starting point. The story-development and therefore the sequence of the presented knowledge is based on the findings regarding the chain of accomplishment and the linking of anchor concepts. To further the accessibility to the new technology I use a “black-and-white-box” approach and teamwork together with challenges and competitions to enhance motivation.

Before I explain the detailed concept in chapter 5 I will compare different educational robotic systems to show which robotic system can be used with the chosen approach.

Educational robotic systems

There are several educational robotics systems available. A complete overview would go beyond the scope of this master thesis, so I will give only a brief synopsis of the most well-known systems in literature. In addition there are several methods using a white-box approach with a micro controller. Since the curriculum focuses on teaching programming instead of teaching all aspects of robotics, a ready-made educational system hence a system with a black-box or a black-and-white-box approach is preferable.

3.1 Lego Mindstorms

Lego Mindstorms is a robot system from The Lego Group a company based in Denmark [39]. Its name is derived from the Paperts book “Mindstorms: children, computers, and powerful ideas [53]” [17]. Lego Mindstorm has three different versions, where the version identifier relates to the version of the central controller

1. robotic command explorer - the first version of the LEGO Mindstorms controller (RCX)
2. next - the second version of the LEGO Mindstorms controller (NXT): the basis of most of the recent research papers regarding Lego Mindstorms.
3. evolution 3 - the current version of the LEGO Mindstorms controller (EV3)

The Lego Mindstorms kit is available in a customer version or an educational set. Besides the central controller the Lego Mindstorms kit includes

- 3 motors
- several sensors, including a touch sensor, a ultrasonic sensor and a light sensor and/or a color sensor
- many LEGO pieces to build the mechanical part of the robot

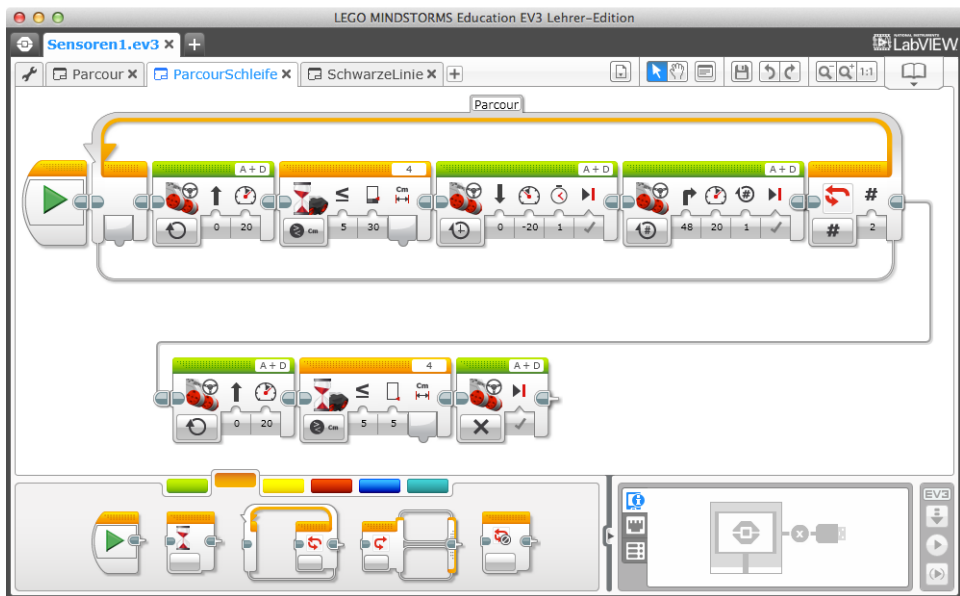


Figure 3.1: Lego Mindstorms EV3 IDE

The exact contents vary from version to version but all kits include the controller, sensors, motors and building pieces. The LEGO pieces are compatible with other LEGO Systems especially LEGO technic and can therefore be expanded easily. All components can be purchased individually.

Graphical IDE

The Lego Mindstorms controller can be programmed with a specialized integrated development environment (IDE) which can be downloaded from the Lego Mindstorms site [39] and can be installed on Windows and Mac OsX Systems. The basis are graphical building blocks which are based on the visual programming language LabVIEW (cf. Figure 3.1). Instead of using the provided IDE, there is the possibility to use an existing LabView Environment.

RobotC

RobotC is a C-based programming language with an IDE for writing and debugging programs [64]. For the IDE a license needs to be purchased (a 10 days trial version is available) and the only supported operation system is Windows (XP, Vista, 7 and 8). According to the LEGO Mindstorms frequently asked questions (FAQ) RobotC is the only official supported computer language environment aside from the graphical IDE presented above [40].

To use RobotC you need to install a corresponding firmware on the Lego Mindstorms EV3 brick. An appropriate button is included in the downloadable IDE. The basis of RobotC is the C programming language, where the motors and sensors can be accessed by the way of control arrays (cf. Figure 3.2) where the significant values can be written or read as integer values.

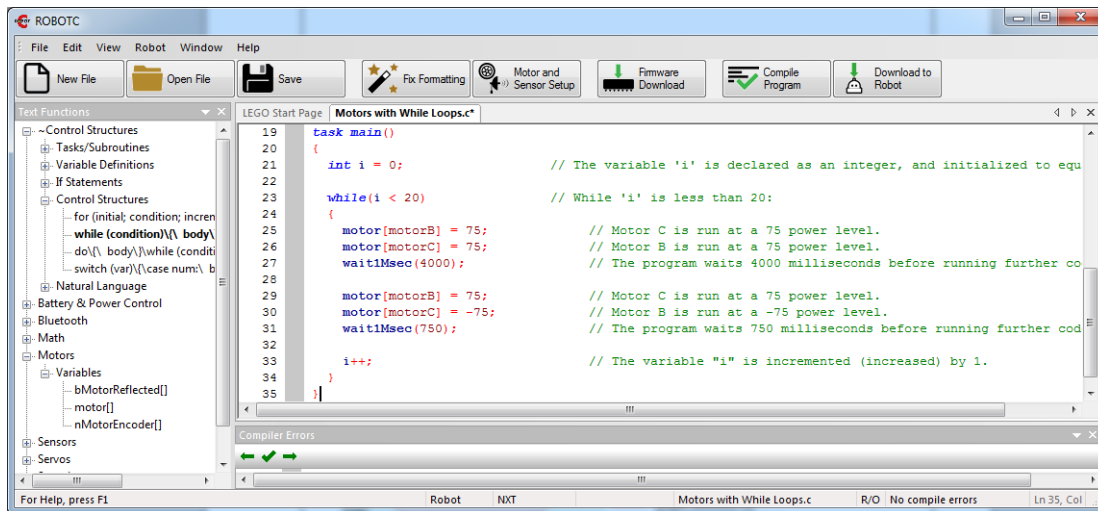


Figure 3.2: RobotC IDE

Teaching programming with RobotC has the advantage that the setup of the environment is relatively easy. The language provides an application programming interface (API) to teach the basics like variables, arrays, functions and control structures as shown above and there are some curricula with this approach using the NXT-brick [18,26]. The main difficulties with RobotC lie in intensive usage of arrays. Arrays are one of the most difficult subjects in early programming. Lahtinen et. al. show that students and teachers consider arrays more difficult than parameters and of similar complexity as other structured datatypes [37]. The teachers and students asked by Milne and Row rank the difficulty even higher on a comparable level with variable scope and constructors [50].

Java with Lego Mindstorms

The programming language Java is available for the Lego Mindstorms robots through the open source project LeJOS. LeJOS is based on the ARMv5 port of Java standard edition (SE) Embedded version as stated by Oracle [51]. Since it is an open source project the preparation for the Java virtual machine (JVM) on the Lego Mindstorms EV3 brick is relatively complicated especially for an unexperienced user and it requires an SD card for every robot which should run the JVM as the controller will boot from this card to replace the existing operating system for the Lego Mindstorms EV3 brick with an operating system including Java (and excluding the support for LabView).

To program the Lego Mindstorms robot there is a plugin available for the Eclipse IDE which is widely used as professional IDE for Java [24]. The eclipse plugin provides a configurable interface to the LeJOS API and to the Lego Mindstorms Controller. An example program is shown in Figure 3.3

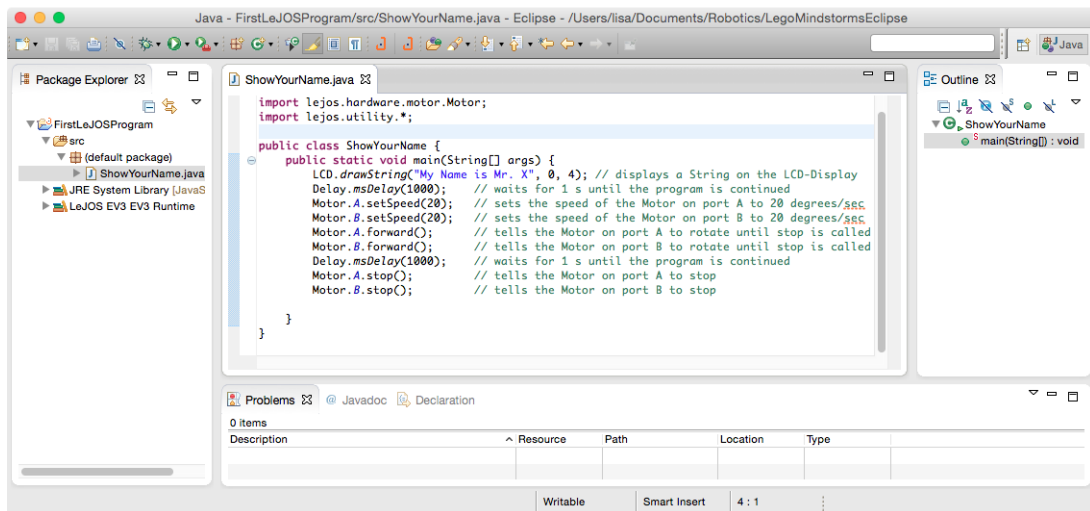


Figure 3.3: LeJOS with Eclipse

Teaching programming with LeJOS Lawhead et. al. applied Java with Lego Mindstorms in introductory programming with the NXT brick. This curriculum focuses on object-oriented concepts which are demonstrated with the physical objects present on the robot (e.g. the motors and sensors) [38]. The main difficulty with the use of LeJOS aside from the complex set up is the early use of object oriented programming techniques (c.f. section 2.1).

3.2 Botball

Botball is an educational robotics program developed by the KISS institute of practical robotic (KIPR) and is very popular in the USA. Its goal is to engage middle and high school students in robotics activities centered around a team-oriented tournament, which is structured in several regional tournaments culminating in a global tournament held at the global conference on educational robotics [70]. Participating robots must be composed of parts included in a predefined robotics kit, which can be bought directly through the botball homepage [27].

There are two different kind of botball sets: the elementary botball kit (also called junior botball challenge robotic kit) and the full botball kit. The elementary kit contains amongst other things:

- KIPR Link Controller (designed by KIPR)
- Starter LEGO bag - Assorted LEGO Pieces
- Motors - (2) SG-5010 Standard Motor, (2) SG-5010 Black Gear Motor
- Sensors - Light Sensor, Small Touch Sensor, Large Touch Sensor, Long Lever Sensor, ET Sensor, (2) Small IR Sensor
- selected mechanical parts

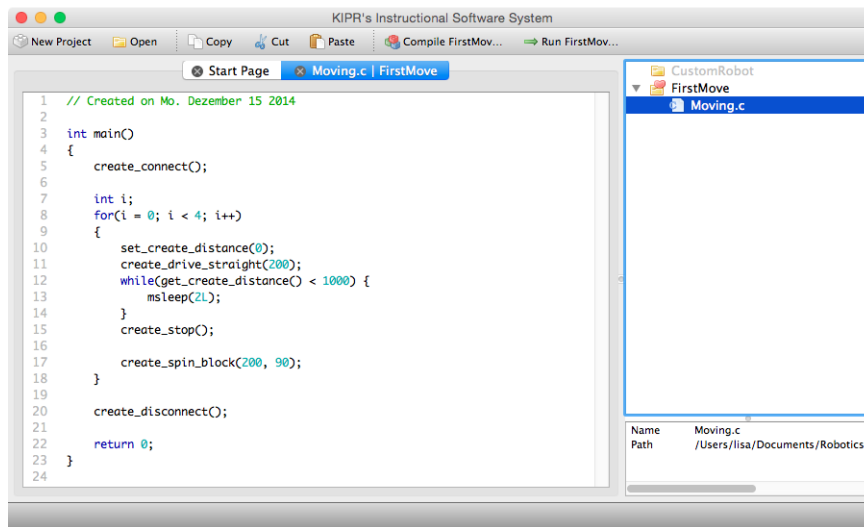


Figure 3.4: KISS IDE

The full botball kit features additionally a second KIPR Link Controller, the iRobot Create (a basic robot that is able to move around similar to a vacuum cleaner, so a robot does not have to build from scratch) and two video cameras suitable for the vision system of the controller.

On the website there is a building instruction for a basic moveable robot that is used in the provided programming curriculum and can be constructed using the elementary kit. Additionally the Create can be used as basis for teaching programming. However, because of the abundance of additional material available, the limits for building robots are not easily reached.

KISS IDE

The KISS IDE is designed for programming the KIPR Link Controller and uses the programming language C (cf. Figure 3.4). One big advantage is the packaged simulator for the KIPR Link Controller. It enables students to pre-test their creations before testing them with their robot, as long as their creations are similar to the described demonstration roboter (cf. Figure 3.5).

Teaching programming with the KISS IDE The easy setup and the free software make the botball system very accessible for teachers. The main functionalities of the robot are controllable through basic C-functions with primitive datatypes as parameters, which makes them relatively easy accessible for novice programmers. The main disadvantage lies in the restricted availability in Europe with an mail-order through the web shop being the only option.

Java on the KIPR Link Controller

Although Java is advertised with the KIPR Link Controller there is currently no official technical support for this approach. There are several reportings on the internet on how to install a Java

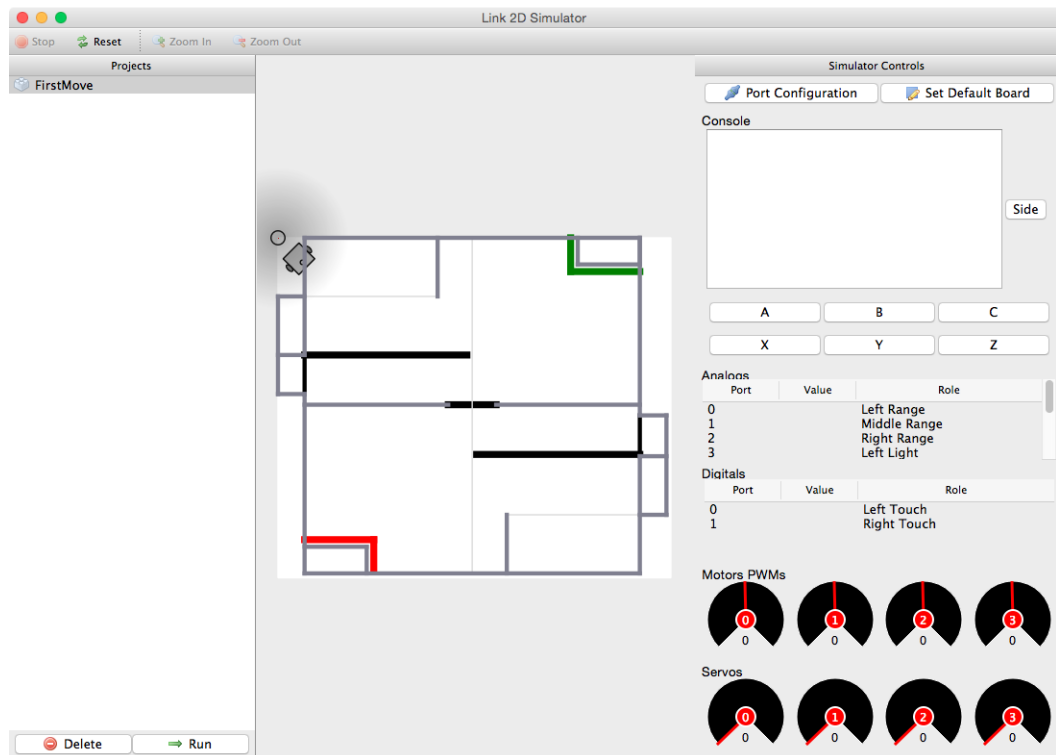


Figure 3.5: Simulator for the Link Controller

Virtual Machine based on the Virtual Machines available for the CBC and the CBCv2, the two preceding controller generations.

3.3 Robotino

Robotino is an educational robotics system developed from Festo, a company specialized in automation and industrial control based in Germany. Its basis is a moveable robot with an omnidirectional drive, several integrated sensors (infrared sensors and a color camera, inductive and optical sensors are additionally available) and an embedded personal computer (PC) to the component object model (COM) express specification. It can be expanded with several pre-built components like an electric gripper, a fork lift, a laser scanner or an WLAN access point [25].

Robotino is used in a special competition during Robocup and in a worldwide profession challenge called SKILLS and is also very successful in empowering aspiring african engineers [75].

Robotino View

The Robotino can be programmed using a graphical IDE called Robotino View. In Robotino View programs are created using data flow charts connecting several predefined function blocks

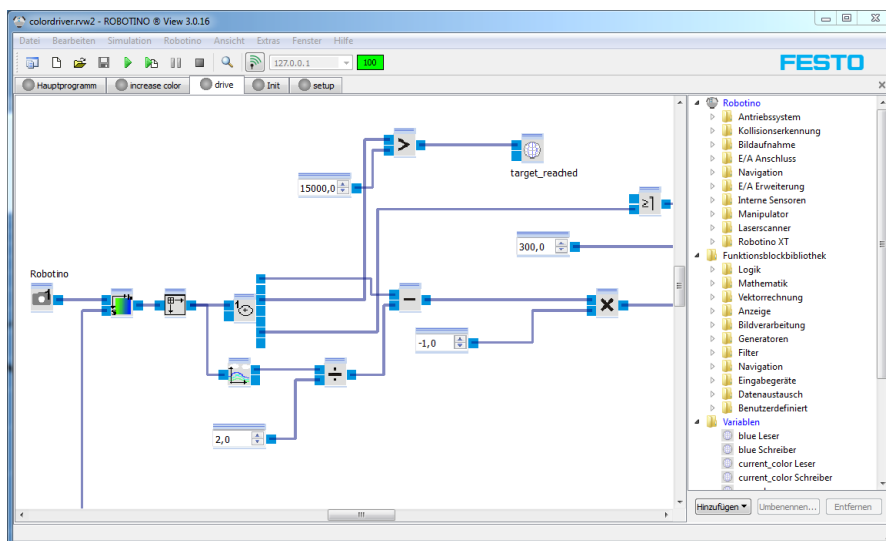


Figure 3.6: Robotino View

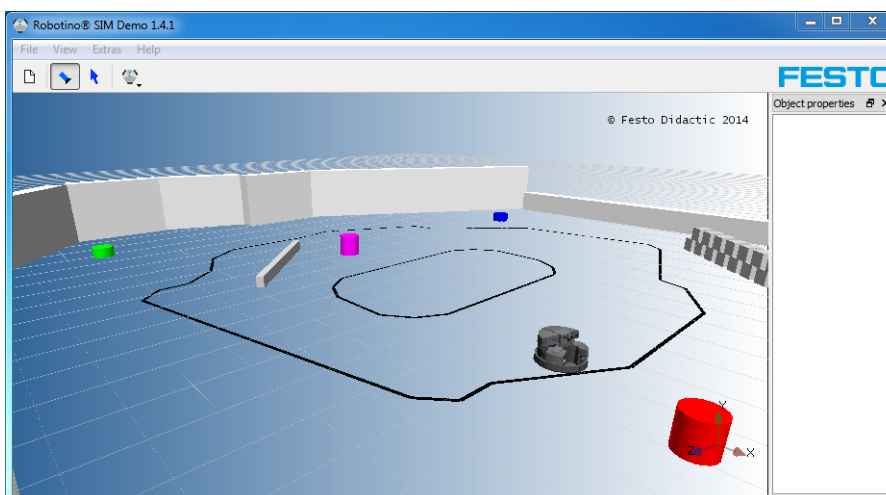


Figure 3.7: Robotino SIM simulator

(cf. Figure 3.6). This graphical environment can be used in junction with an extra available simulator program, robotic SIM. With this simulator you can build a virtual 3D environment for the robot that enables students to test their program in the virtual world before applying it to the real robot (cf. Figure 3.7)

Programming C++ or Java with the Robotino

The Robotino provides an C++ API for the Robotino and additional Java Wrappers for this API. On Windows systems the supported development platform is Visual Studio (cf. Figure 3.8). An

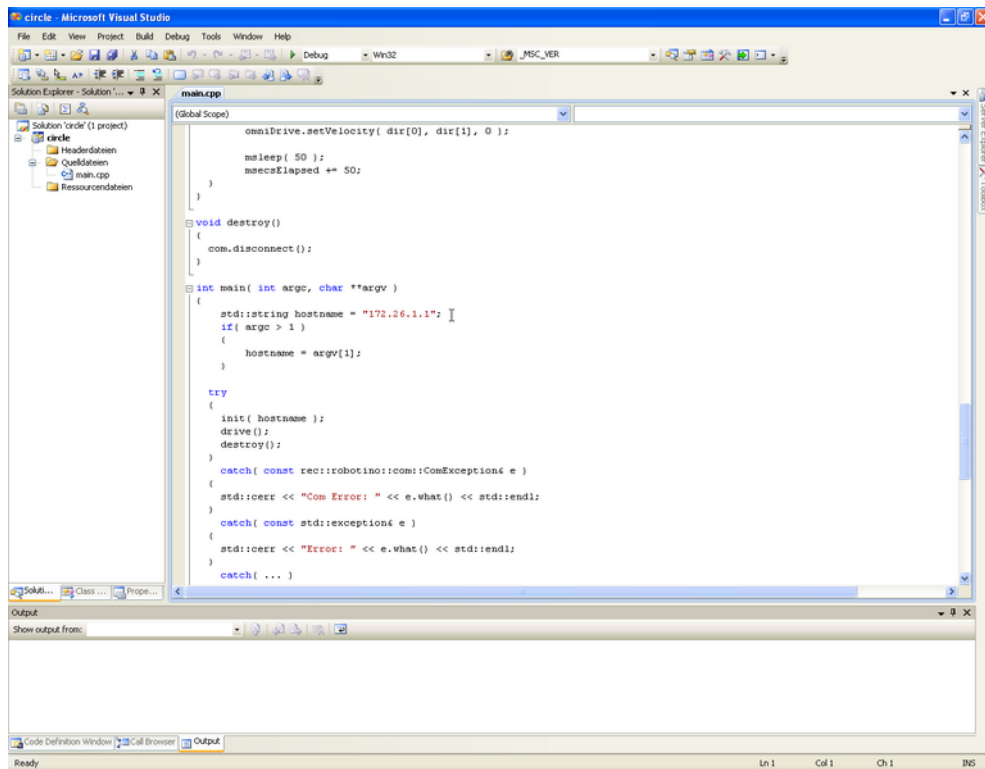


Figure 3.8: Robotino C++ in Visual Studio [58]

additional port for linux systems is also provided.

Teaching programming with the Robotino The C++ as well as the Java API are completely object orientated. The provided tutorials require the student to override existing classes to achieve simple driving. Additionally arrays are also used in the introductory tutorial although the usage of arrays can be avoided. Since objects, classes and inheritance are quite advanced topics ((c.f. section 2.1) the usage of the Robotino in an introductory programming course is difficult.

3.4 Additional educational robotic systems

The robotic sets mentioned above are by no means a complete listing of the available robotic systems. Ruzzenente et. al. try to give a systematic overview for tertiary education, where they categorize the different robotic systems:

1. Non-versatile kits: Manipulators - e.g. Servobotics RA-02 Robotic Arm, Robot Arm Trainer, Lynx
2. Non-versatile kits: Household Robots - e.g. Pioneer Robot 3DX, Khepera III Robot, Hemisson, iRobot Create, MiaBot, WowWee Rovio, E-Puck

3. Non-versatile kits: Robotic Aircrafts - e.g. Skybotix's Coax Helicopter, Parrot AR. Drone, AscTec Quadrotor Pelican
4. Non-versatile kits: Humanoid Robots - e.g. Aldebaran Robotics Nao
5. Versatile kits - e.g. Boe-Bot, Stingray Robot, VEX, FischerTechnik, Qfix and the already mentioned Lego Mindstorms [66]

3.5 Different approaches in educational robotics

There are several educational robotic systems which are designed to use a different approach to attract younger students or students who are not initially interested in technology. As such they only support a graphical IDE. In this category fall exemplary

- Pico cricket [65]
- Lego WeDo [34,46]

Several other approaches do not use a prefabricated set and build a Robot around a central micro controller. Some interesting approaches are found in following projects:

- Mattie as an educational platform which enables children to work on their first robot prototype from different views and starting points [29]
- the Poppy project which tries to build an open platform for 3D printed robots around an arduino micro controller [56]

3.6 Summary

There are several educational robotics systems available. For teaching introductory programming sets making a fast simple robot construction possible are suited best, because the focus of the curriculum lies on the software development opposing to electronics and mechanics necessary for self-made robots based on a micro controller.

The goal of this work is teaching a programming language which is used on an industrial level. For that reason a graphical environment can be an additional asset which enables the teacher to show some concepts in a different light but the graphical programming environment is not sufficient for the selection of a specific robotic system.

So the Lego Mindstorms set, the Botball kit and the Robotino are all candidates for the use as fundament for an curriculum in introductory programming.

The Robotino is used as the competition set for mobile computing in a worldwide profession challenge called SKILLS and is successfully used by the participants to solve problems close to reality. For teaching introductory programming the Robotino is not optimal because of its price. For usage in an introductory programming class many robots need to be available, so that the ideal group size of two to maximum three students can be adhered to. Additionally the high

level concepts needed to use even simple motor movement in C++ and Java hamper the usage for novices.

Lego Mindstorms has been used successfully with the NXT or RCX brick using different approaches and different programming languages [18,44,76]. But unfortunately these researches lack the avenue how to overcome the difficulties of the complex programming concepts needed for the beginning (arrays in C, objects and static constants in Java). There are also some studies pointing out these issues [9,30].

The Botball kit is not affected by the above mentioned difficulties. The elementary set is priced only slightly above the Lego Mindstorms set and the software is without any charge. Even the full Botball set is priced well below a Robotino system and therefore better affordable in greater quantities. Although Java has only very limited support and is hence not suitable in a classroom context, the programming API for the language C is very low level and the first programming concept necessary is the concept of function calls with literals, a concept being equally necessary in any other beginning of teaching programming (eg. output commands). Additionally the provided simulator adds to the benefits of using Botball as well as the availability on multiple operating systems.

In our workshops we use Botball because of the easy approach to the programming concepts in the programming language C and the possibility to for small student groups because of the availability and affordability of the set.

4.1 Context

This curriculum is aimed at teaching programming with an industrial used programming language on an introductory level using robots as a way to enhance student participation and motivation. The target audience are students aged 13 to 18 who are interested in technology and engineering, e.g. attending a technical high school. In this context the curriculum should cover the goals for the accredited syllabus for software engineering in technical high schools for computer science in the first year¹.

Therefore the contents of the curriculum cover abstract programming concepts like variables, function calling and defining and control structures. Moreover this means a procedural approach is specified and objects have to be avoided.

4.2 Existing curricula

There is already a curriculum for teaching their students how to program the Botball robot. The main goal of this curriculum is to teach enough basics to enable the students to participate in the tournaments. This curriculum is not publicly available and is adapted and enhanced every year to meet new requirements due to controller changes or to enhance student involvement.

Additionally there is a website based on an older version of the controller, the CBC that offers an online curriculum (<http://botballprogramming.org/>). Since the code changes for the new controller are minimal, most of this online course still works with the new Controller. Still this online resource mainly purposes to teach the handling of the robot in terms of enabling the students to drive around and working with sensors in view of the tournament. The focus lies on explaining the functions available to access the various sensors and actors. There are some explanations about basic programming concepts, but only briefly and not as a main goal. [21]

¹Anl. 1/5 BGBl. II Nr. 300/2011 “Lehrplan der höheren Lehranstalt für Informationstechnologie” found at <https://www.ris.bka.gv.at/Dokumente/Bundesnormen/NOR40131765/NOR40131765.pdf>

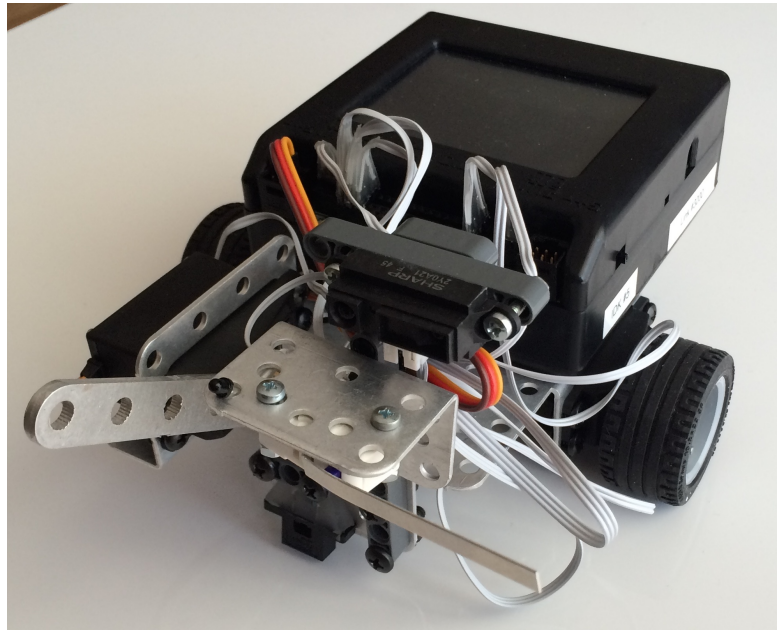


Figure 4.1: The robot build for the workshop

4.3 The Robot

The robot is derived from the original demo-bot built of Botball used in the workshops preparing the teams for the tournament. Aside from the controller it consists of:

- 2 continuous rotation motors directly attached to the wheels
- one servo motor attached to the front (for applying a grappling arm, a bulldozer plate ...)
- one touch sensor
- one IR-distance sensor
- one reflectance sensor
- one light sensor

The sensors are all attached to the front so they can be used for navigation (figure 4.1).

4.4 Basic concept

The main curriculum is based on the concept shown in figure 4.2. The aim of the individual parts of the concept is to enhance the motivation and offer the basis for understanding the technical relations on a deeper level.

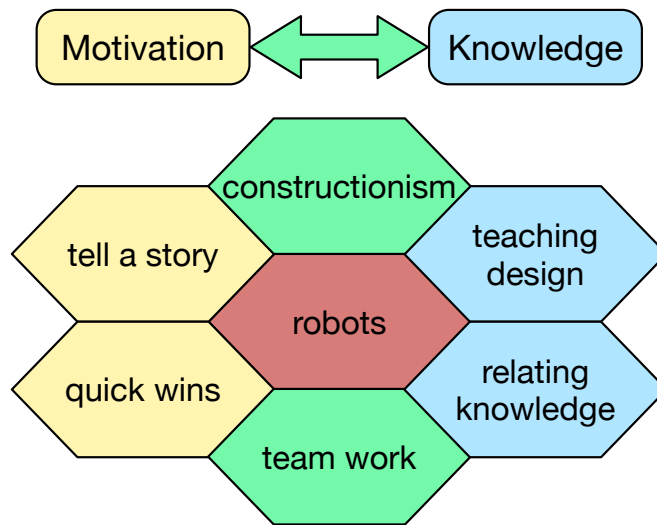


Figure 4.2: Basic concept

As with several other available curricula the presented concept focuses on the content-student relation of the didactic triangle. The implementation of the teacher part needs to be adapted to the teachers using this work for themselves.

Constructionism

Using robots lays the foundation of having elements of constructionism throughout the curriculum. The eight big ideas forming the Constructionist Learning lab are always central in the curriculum design. With the robots student can develop their own approach to technology. Additionally, students are easily capable of developing the given assignments further on their own according to their interests.

When introducing new concepts I use steps similar to the ones Wulff suggested for using constructivist pedagogy (cf. section 2.3):

1. Initial exposure through explanation of the new concept
2. Small guided examples (called warm-up tasks)
3. Group assignments (challenges and competitions)
4. Evaluation of the learning achievement (according to the course setup for grading)

Only the set-up of the robot requires more explanation, but with many small warm-up tasks to get the students started before the first challenges can be issued.

To enhance constructionist elements it would be ideal to let students build their own robot, which depends on the availability of robot kits. Having a robot for each team means the robot kit is reserved for that team during the course period. Having a pre-build robot allows for usage throughout several parallel courses.

Tell a story

It is possible to teach just one feature or concept after the other but to give the students a meaning of what they do, it is necessary to have a leitmotif guiding the students from one concept to the next. It may be necessary to step aside and have tasks and explanations outside this central theme but it should deliver the main motivation for introducing a new chapter.

I encourage the students to imagine their own dream robot during the first lesson. So then I can introduce my dream robot: “Seek-and-spot-robot” abbreviated SaSbot (in german “Suchen-und-Entdecken-Roboter” abbreviated SuEbot). The duty of SaSbot is to seek and find my keys or my mobile phone when I mislaid them somewhere in my apartment. Then I use SaSbot for introducing the basic programming concepts and robot functions.

SaSbot has the advantage of being realizable with the means the students have with the Botball system and the programming techniques. So there is no need to motivate the topics I introduce extra and in the end the students can have their own SaSbot if they follow the curriculum.

Not every student can relate to SaSbot (some students want a battle robot, some students want a homework robot, etc.). So I encourage students to think of their own applications of the presented techniques. Ideally the students develop their own story of what they want to do.

Quick wins

To simply follow a long term goal may be too tedious and SaSbot may not be an objective the students find worth pursuing. So it is necessary to present them with short term achievements they can reach after a small amount of learning units.

For the curriculum this means I present a new concept together with some simple tasks I call “warm-up tasks” since the goal is to familiarize the students with the new concept and make them ready for the challenge. The solution to a warm-up task will be shown and discussed together after some time, so even students who had difficulties with this “warm-up task” will have a workable solution. After the warm-up tasks there will be a task called challenge, because the student teams need to find the solution for this task on their own. There will be support and assistance from the teacher(s) present but no general solution will be presented.

To enhance the interest in these challenges some of these tasks can be chosen individually from a pool with some of the challenges being more of an exhibition type of challenge. Some other challenges will be issued in form of a competition. Especially the individual tasks provide students who are struggling with programming with the opportunity to accomplish something on their own because the tasks are designed to have wide range of difficulty levels. The competitions on the other hand may give students an incentive to push their limits and reach for a better solution.

Team work

Most institutions cannot afford one robot for every student. So teamwork is a necessity. But teamwork can also be very beneficial in terms of motivation and learning, since there is always a

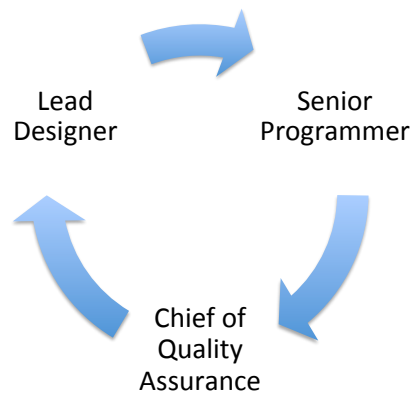


Figure 4.3: Team positions in a 3 person team

team member who you can ask, when requiring help at a specific point. And if all team members have no clue together there is still comfort in the fact, that they are not alone (cf. section 2.2).

But team sizes need to be small and it is necessary to make sure that all team members contribute to the work. As I usually work with three student teams I divide the task of programming into three parts derived from classical software engineering:

- Software Design
- Programming
- Testing

To enhance the importance of each part (especially of the non-programming-parts) team members are assigned special names to encourage students to fill out the role. The three roles are:

- Lead Designer
- Senior Programmer
- Chief of Quality Assurance

After each programming task or challenge the roles should switch in a clockwise order. So the Lead Designer becomes the Senior Programmer, the Senior Programmer becomes the Chief of Quality Assurance and the Chief of Quality Assurance becomes the Lead Designer. So every student needs to take part in the task.

As a second benefit this division encourages to actually make a design step before the actual programming. Experience shows that bringing the students to make a design before starting to code is very difficult since most students only view this as additional and unnecessary work. With this division a student is solely responsible for the design so for this student the design is not additional because it is his (only) contribution to a solution.

Testing on the other hand is more natural with tasks including robots since it is necessary to try the program with the robot on the field. And nearly all times it is inevitable to correct the program after the first run.

When working with teams consisting of two students I suggest combining Lead Designer and Chief of Quality Assurance to have the coding part set apart from the other tasks.

Relations to previous knowledge

When explaining new concepts I try to emphasize the similarities to previous taught knowledge and common patterns. This gives students the chance to memorize facts more easily and gain a deeper understanding. Only at the very beginning when explaining the IDE and trying an example program the concrete explanation is delayed until the next lesson. There is already much information the students get at the first lesson and this may be to overwhelming.

This approach might take a bit longer than showing commands on a “Write this down to make the robot do this or that” basis but I want the students to understand what they are doing. And with constant repetition of facts and structures the sustainability of the knowledge will be increased.

The knowledge relations are supported by figures enhancing the relations and making the patterns of the language syntax more visible. These figures allow the students who do not like long explanations to identify the key concepts and structures right away.

Teaching design

Designing a program from scratch is difficult even for more experienced programmers. For novices this is even more true. So the design of a program is a step most students omit resulting in getting lost in the assignment or having a program consisting of “spaghetti code”.

Designing a program is considered a necessary step by experienced programmers and there is a vast toolset of graphical elements intended to help design a program (e.g. flowcharts, UML, Nassi–Shneiderman diagrams, ...). But as experience shows students are not very inclined to draw their designs whether on paper or with a drawing program. Therefore it is advisable to let them write down their design as comments². Comments are also an unpopular programming part and because of that often omitted. So it may be possible to achieve one goal with a two-folded approach, i.e. using comments as design tool.

I also introduce function definition as a design tool. When thinking about steps the robot has to follow to achieve its goal, it is natural to have these single steps written down as functions. With introducing the definition of functions very early the students can also develop a habit of writing functions and thinking in design steps.

It is very difficult for students to develop a complete design for the assignment. Some simply despair at the thought of the whole assignment. So I encourage developing one step after the other which is natural for most students. When presented with the task to let a robot go forward, turn and go back to the starting point, many students start by simply trying the forward move-

²The idea of using comments as design tool was born at the instructor summit of the Botball 2015 season

ment. While encouraging this bricolage approach it is easy to talk about putting this first part of moving forward in a separate function which can also be used for the “go back” part.

Structure of the Curriculum

Every content section consists of two parts:

1. Didactical considerations: which is intended for the teachers use and where I explain what I am teaching and why I am teaching it this way.
2. Content for students: Here I give an example of a way how the lesson can be taught for students. It is written like a student handout, but I focus on the approaches. So repetitive parts are not as much included as I would include with a real student handout.

To present the concept I concentrate the work on the beginning and some key concepts. It is by no means intended to cover a whole semester. This would exceed the timeframe of this work but with the basic concepts covered the follow-up lessons can be developed in the same manner.

Curriculum

5.1 The Beginning

Didactical considerations

For the start several explanations are necessary to build a frame, where the students can tie new knowledge up. So the first two chapters are a bit heavy on the theoretical side, but this will help to build the foundation for future knowledge. The goals for the students are:

- understanding of the programming process (write source code, compile, run)
- comprehending the terms source code, compiler, machine language and binary code
- getting familiar with the robot, especially with the controller
- getting familiar with the IDE
- understanding the basic structure of a program

This part builds the basis frame to which the students can anchor their further knowledge. To loosen up the theoretical part at the beginning I work with some student activities aside from the work with the robots to consolidate the presented knowledge and bring some fun to the theory.

As a starting point I ask the students if they have talked to their computer eventually. Nearly everybody has done this, some times in terms of curses. This leads to the terms of binary code and machine language and the need for compilers. To emphasize this fact the students can program a fellow student with binary code. They are given a fixed set of instructions with a translation in binary code.

The goal is for one student to write a program for a fellow student using binary code instructions. The fellow student has to follow the instructions he receives. Afterwards a discussion about readability of binary code can lead to the next step: the need of translators.



Figure 5.1: The computer does not understand human language

To consolidate this fact another round of “program your classmate” can be done with the intermediate step of a programming language and a translator. After the students understands the steps necessary to program a robot the robots can be used.

The process of compiling and running the first “Hello, world!” example-program on the robot should be done together. Experience shows that it is difficult when doing it for the first time, but students get accustomed to it very quickly. It is necessary to emphasize and repeat the link between the single steps for getting the robot to work and the theory of compiling discussed before.

Content for students

Most people have talked to their computer. But the computer did not respond as is shown in Figure 5.1. This is no surprise because a computer understands *machine language* only which is a *binary code*. Binary code is a way of representing numbers (e.g. tab 5.1).

0	1	0	0	0	1	0	1	$= 0 \cdot 128 + 1 \cdot 64 + 0 \cdot 32 +$ $0 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 =$ $64 + 4 + 1 = 69$
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
= 128	= 64	= 32	= 16	= 8	= 4	= 2	= 1	

Table 5.1: Example for an 8-bit binary code

But a computer does not only store numbers. To store other content than numbers they need to be encoded e.g. like characters are encoded in ASCII or unicode (table 5.2) or like colors are encoded in RGB or CMYK. The instructions a computer can understand are also encoded in

binary code	decimal number	character
01000001	65	A
01100110	102	f
00100001	33	!
00110001	49	1

Table 5.2: Example for an ASCII encoding of characters

binary code.

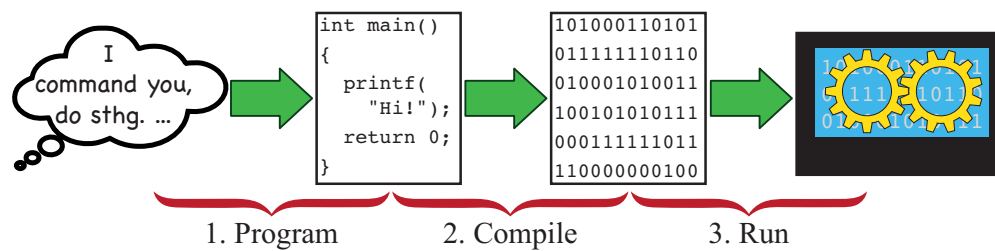


Figure 5.2: The principle of giving a computer an instruction it understands

Group activity: program your fellow student The table 5.3 lists machine language instructions (binary code) for programming a “student robot”. Write a program for a team member or class mate using this binary code. Then give this program to the designated student who becomes a student robot. The student robot has to follow the instructions step by step as written down by you.

instruction	machine language instruction
one step forward	00000001
three steps forward	00000011
turn through 90°	00000100
stand up	00001000
sit down	00001001

Table 5.3: Binary code for instructions

Binary code is hard to write and hard to understand for humans, but it is the native language of computers and robots. So to improve the human-computer-relationships translators have been invented.

Since there are no translators from natural language to machine language students still need to learn a foreign “computer language”. This language is called *programming language* (like C, C++, Java, Javascript, php, python, ...). Such a translator is called *compiler* and it *compiles* a specific programming language to binary code for a computer (e.g. a C-compiler compiles programs written in C to machine language). So to write a program which a computer understands there are several steps needed shown in figure 5.2.

Team activity: program your fellow student using a compiler Build teams consisting of three students. Each team has the following members:

1. Programmer: writes down the instructions using programming language.
2. Compiler: translates the instructions to machine language. A compiler is *very strict*. If an instruction has errors (like a missing letter, written with upper-case letters or a space between a letter and a number) a compiler cannot translate this instruction

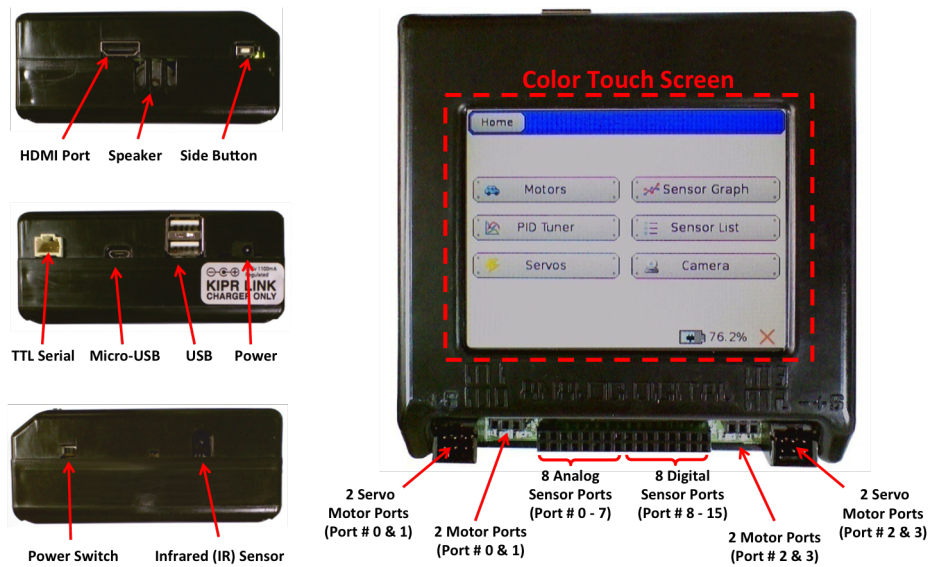


Figure 5.3: The KIPR link controller (picture courtesy of KIPR)

3. Robot or computer: follows the instructions it is given and thus *runs* the program. A robot or computer is *very strict*. It can only follow the instructions it receives in binary code.

Table 5.4 shows the instructions in programming language and machine language for this activity.

instruction	programming language instruction	machine language instruction
one step forward	step1	00000001
three steps forward	step3	00000011
turn through 90°	turn90	00000100
stand up	stand	00001000
sit down	sit	00001001

Table 5.4: Programming language instructions with binary code translation

Now it's time to take the theory to the computer and to work with the IDE. The IDE is the place where the program is written. It provides a simple "Hello, World!" program as an example. To run this program the instructions must be compiled, so that the robot can understand them. The part of the robot which receives the instructions and controls the motors and sensors is called *controller*. The controller is thereby the equivalent of the human brain. As a mean of communication it has a touch screen where it can write messages for the operator (5.3).

When computer and controller are connected via USB the program written in the programming language C on the computer can be compiled for the robot controller. If no errors are detected the program must be run on the controller itself. The necessary steps are:

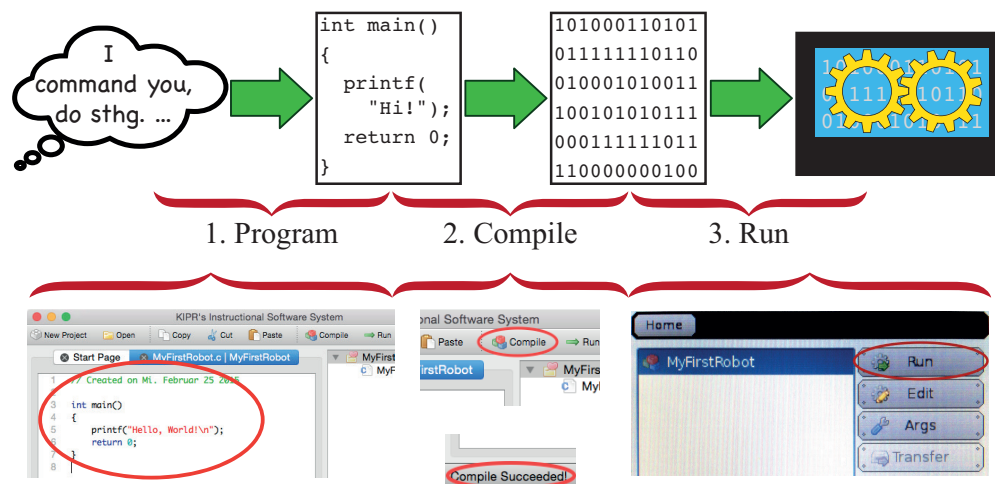


Figure 5.4: The principle of giving a computer an instruction it understands combined with the steps necessary on the computer and controller

1. Write a program or load an example program in the IDE on the computer.
2. Connect the robot (which needs to be turned on) with the computer via USB and select the robot as compile target.
3. Compile the program and check if the compilation was successful (if there are no programming errors)
4. Download the program to the robot
5. Run the program on the robot controller directly

For a detailed explanation see appendix A.1.

The relation between the steps above and the principle of giving a computer an instruction it understands is shown in Figure 5.4.

5.2 Program structure

Didactical considerations

Now that the students know the basic steps of getting a program to run on the controller the program itself can be examined and analyzed. The goals of this section for the students are:

- understanding the structure of the program and the main function
- understanding the structure of the printf-function call
- comprehending the terms syntax, comment, block and function

```

1 // Created on Mi. Februar 25 2015 ← Comment
2
3 int main()
4 {
5     printf("Hello, World!\n"); ← main area of the
6     return 0;                 program
7 }

```

Figure 5.5: The basic structure of a program

```

1 // Created on Mi. Februar 25 2015
2
3 int main() ← header of the
4 {         ← start of the block
5     printf("Hello, World!\n"); ← body block of the
6     return 0;                 ← main-function
7 } ← end of the block

```

Figure 5.6: The basic structure of the main function

- identifying syntax errors

Since the syntax of a computer program consists of several repeated structure elements it is necessary to highlight these structure elements as soon as they appear. This enables the students better to identify these elements later on as well, store these structures and patterns into their brain and refer to them when needed.

To consolidate the structures presented the students should try some simple scenarios and analyze together with the teacher the differences and the occurring structures.

Teaching syntax errors explicitly is the first step to enable students to cope with their own errors and find solutions to them. Learning to analyze errors is a long process and the presented activity is only the first step.

Content for students

First of all there is a difference between a comment, which is not interpreted by a computer, and the main part of the program where the actual programming takes place (figure 5.5). The main part itself is structured again in a header and a body block (every head needs a body to survive). The body block is delimited with a pair of curly braces (figure 5.6). The header-body-theme will be repeated when talking about custom functions and the control structures.

The actual instructions for the robot are placed inside of the main-function-block i.e. inside the curly braces marking the begin and the end of this block. Instructions are also called statements. The statements are usually indented to emphasize the block structure and make it more visible. Every statement inside the block is terminated with an semicolon (figure 5.7). So the robot gets actually two instructions:

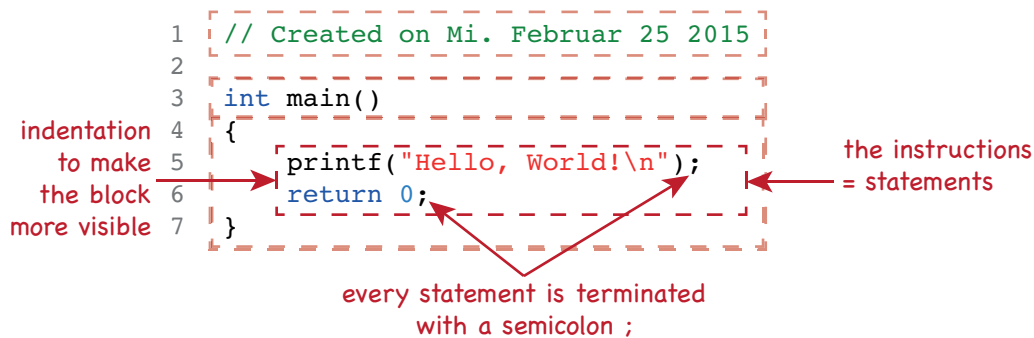


Figure 5.7: The area where the statements are written

1. `printf("Hello, World!");` This tells the robot to write the text “Hello, World!” on its screen (without the quotation marks).
2. `return 0;` This tells the robot to end the program with no errors.

Comments can also be placed inside the main part of the program. Remember they are actually ignored by the robot. Try to run the following program on your robot (listing 5.1).

```

1 // Created on Mi. Februar 25 2015
2
3 int main()
4 {
5     // prints the text Hello, World!
6     printf("Hello, World!\n");
7     return 0;
8 }

```

Listing 5.1: Comment inside the main-function

What happens when you try writing two `printf` commands? What is the difference between this program and the previous one? Try it out (listing 5.2).

SaSbot – Learning to communicate Before tackling the original assignment of SaSbot the robot needs to be able to tell me something. The example program already shows the means of accomplishing this – the `printf`-command. So all we need to know is how we utilize this command to make the robot tell a text we want it to tell. Before listening to more theory it may be best to just try it out.

Before you start working on your own, we need to look at possible *syntax errors*. Syntax errors occur when one does not adhere to the program structure and the instruction set known to the robot. The term *syntax* is generally equated with grammar since it also encompasses the rules under which structure elements and instructions can be used. The compiler is very strict.

```
1 // Created on Mi. Februar 25 2015
2
3 int main()
4 {
5     printf("Hello, World!\n");
6     printf("Hello, World!\n");
7     return 0;
8 }
```

Listing 5.2: Two printf commands

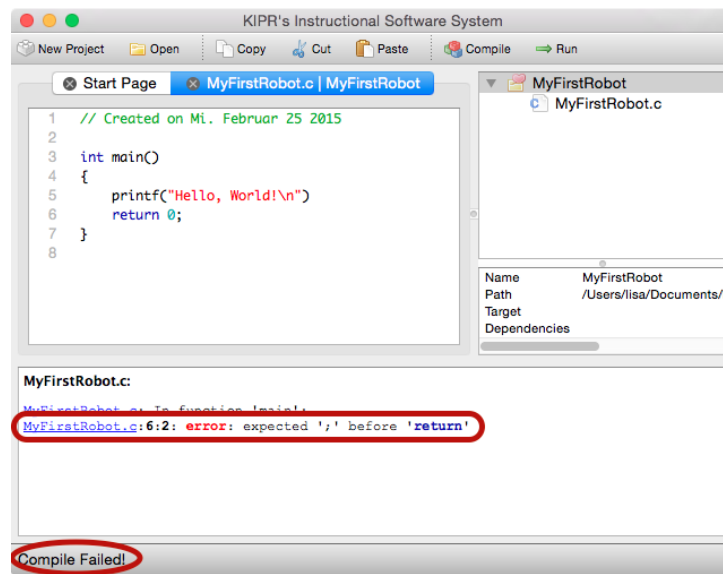


Figure 5.8: Syntax error in the KISS IDE (missing ; after the printf statement)

If it encounters something which is not correct it does not do the translation. Instead it points out, where the error might be and what it guesses is the reason for the error (figure 5.8). Figures 5.9 and 5.10 show more error scenarios.

Warm-up task: error burst Try the presented scenarios together with some more on your own and protocol the findings. Starting point is the “Hello, World!” example program (shown in figures 5.5 to 5.7). Then effectuate the presented error and try to compile the program. Analyze the shown error-message and write the results in the protocol shown in table 5.5. It is required to correct the induced error after each step otherwise the previous errors will affect the error messages. The first 3 lines are filled out as an example. The line number should show the line number stated in the error message. Note that not all scenarios produce an error message. These scenarios likely result in semantic errors in contrast to the syntax errors.

```
1 // Created on Mi. Februar 25 2015
2
3 int main()
4 {
5     printf('Hello, World!\n');
6     return 0;
7 }
8
```

MyFirstRobot.c:

MyFirstRobot.c: In function 'main':
MyFirstRobot.c:5:9: warning: missing terminating " character
MyFirstRobot.c:5:2: error: missing terminating " character
MyFirstRobot.c:6:2: error: expected expression before 'return'
MyFirstRobot.c:7:1: error: expected ';' before '}' token

Figure 5.9: Syntax error in the KISS IDE (missing " after the Hello, World! – only one error but many error messages)

```
1 // Created on Mi. Februar 4 2015
2
3 int main()
4 {
5     Printf('Hello, World!\n');
6     return 0;
7 }
8
```

_internal_target__c.o, MyFirstRobot.c.o:

MyFirstRobot.c.o: In function 'main':
MyFirstRobot.c:(.text+0x5880): undefined reference to 'Printf'
collect2: ld returned 1 exit status

Figure 5.10: Syntax error in the KISS IDE (printf is written with a upper case P)

Warm-up task: greetings As a first real programming exercise modify the displayed text, so that your robot greets every team-member. As an additional challenge try to have the robot write every greeting in a separate line.

5.3 Function calls

Didactical considerations

After the more theory loaded discussion of the program structure on a large scale the students move on to analyze a function call. the goals of this section for the students are:

- Understanding of the structure of a function call (including parameters but without return

Error scenario	Line no.	Error message from the compiler
Delete ; at the end of line 5	6	expected ';' before 'return'
Delete " near the end of line 5	5	missing terminating " character + more
Write the p from printf in upper case (Printf)	-	undefined reference to 'Printf'
Delete the curly brace { in line 4		
Delete the curly brace } in line 7		
Delete one slash / in line 1		
Write two slashes // at the beginning of line 4		
Delete the i out of the word main in line 4 (so the line says <code>int man()</code>)		
Add a second semicolon ; to the end of line 5		
Delete the zero 0 in line 6		
Delete the entire line 6		

Table 5.5: Error scenarios the students should try and record

value)

- Understanding differences between string literals and numeric literals
- Understanding sequence
- Using different functions to make the robot move.

Understanding the structure of a function call is the first step of empowering the students to use the API documentations which enables them to write programs for their own needs. After discussing the structure of a function call the students should reflect on their previous task of greeting. Sequential processing of the statements is not visible to the students because all text lines seem to appear at once. Only in the placement of the text the sequence is visible.

With highlighting the sequence with the function `wait_for_milliseconds()` the students become more aware of the one after the other nature of code execution. If there is time a brief introduction of how a processor works is recommended.

With the introduction of the motor functions driving becomes possible. This is usually the first highlight in the course and after this point larger challenges become possible. It is also the first task complex enough to allow thinking about the task and the solution, aka. the software design (cf section 4.4).

When thinking about design is established, the moment is ideal to introduce the different roles of working in a team with the robot (cf. section 4.4).

```

1 // Created on Mi. Februar 25 2015
2
3 int main()
4 {
5     printf("Hello, World!\n");
6     return 0;
7 }

```

Figure 5.11: Using a function in contrast to a structural statement

Figure 5.12: The structure of a function call

Content for students

SaSbot – communication theory After SaSbot can send us messages that we can read, we need to understand how we can use SaSbot’s functions to give it instructions. For sending something to the display the robot uses one of his functions. Functions can be imagined as abilities of the robot in contrast to instructions used for structuring the program itself (figure 5.11). The usage of the `printf`-function itself follows a pattern which is common to the usage of every function the robot has (figure 5.12). Now evaluate your previous greeting program (a possible solution is shown in listing 5.3). Think about the following questions:

- What does the parameter-value configure?
- The computer executes the three `printf`-functions one after the other (= *sequential*). Is this visible, when observing the output?

So let’s look at another function call (figure 5.13). It is structured just the same way as calling of the `printf`-function. The only difference lies in the type of the parameter. Instead of text there is a number and therefore there are no enclosing quotation marks (and the color is different too). This function makes the *program* (not the robot!) wait for *n* milliseconds where *n* is the value of the parameter. So when it is used between the `printf`-function calls the sequence nature of program execution will be visible (listing 5.4).

Note: placing a `wait_for_milliseconds` call after the last `printf` will have no visible effect since the program will only wait a bit before shutting down finally. After trying this program think about the following questions:

```

1 // Created on Mi. Februar 25 2015
2 int main()
3 {
4     // Greeting of Lisa
5     printf("Hello, Lisa!\n");
6     // Greeting of Nicole
7     printf("Good day, Nicole!\n");
8     // Greeting of Andrej
9     printf("Hi, Andrej!\n");
10    // Terminating the program
11    return 0;
12 }

```

Listing 5.3: Greeting of three team members

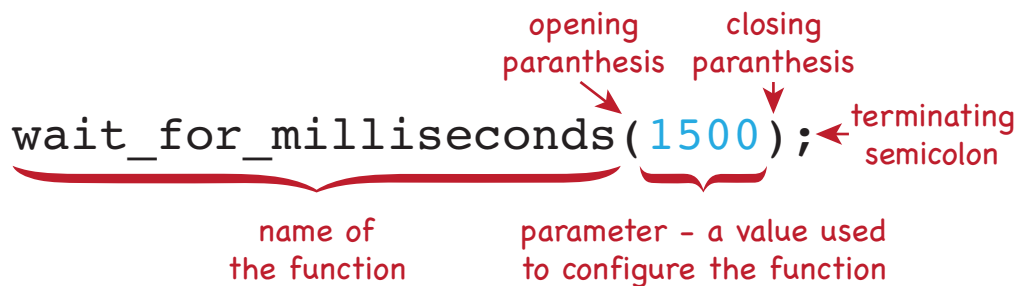


Figure 5.13: The structure of the wait_for_milliseconds function call

- What is different when writing a number and writing a text as parameter?
- What is different between the execution of this program and the execution of the previous program?
- What happens when you change the values of the wait_for_milliseconds parameters? Try different ones (I suggest values less than 10000).

SaSbot – Learning to drive When you need to search for something you need to be able to move around. So the first step for SaSbot in his quest of becoming the best seeker droid is learning to drive around. The robot is equipped with 2 motors which are attached directly to their correspondent wheels – one motor for one wheel (figure 5.14). There are several functions being able to control one motor. The one which is most easy to control is `mov` (abbr. for *move at velocity*). It turns the one specified motor with a defined velocity on (and the motor stays on until instructed otherwise). The structure of calling this function follows the shown rules with the exception that this function has two parameters (cf. figure 5.15):

1. the number of the motor which should be controlled

```

1 // Created on Mi. Februar 25 2015
2
3 int main()
4 {
5     // Greeting of Lisa
6     printf("Hello, Lisa!\n");
7     // wait for 1500 milliseconds
8     wait_for_milliseconds(1500);
9     // Greeting of Nicole
10    printf("Good day, Nicole!\n");
11    // wait for 1500 milliseconds
12    wait_for_milliseconds(1500);
13    // Greeting of Andrej
14    printf("Hi, Andrej!\n");
15    // Terminating the program
16    return 0;
17 }

```

Listing 5.4: Slow greeting of three team members

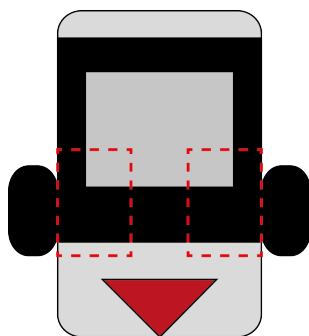


Figure 5.14: The motors of the robot (one attached to each wheel)

2. the velocity the motor uses to drive in ticks per seconds (one turn of the wheel are 1000 ticks). The maximum velocity allowed is 1000 ticks/s.

Since `mov` turns a motor on there is need of a command to turn a motor off again. The appropriate function to do so is `ao` (abbr. for *all off*). This function turns all motors off. Since there are not any configurations necessary for turning all motors off, the function does not have any parameters at all (figure 5.16)

Designing a program

Now that all functions necessary for driving are introduced it is required to think about how these functions need to be put together for achieving a movement for the robot. Maybe it is helpful to

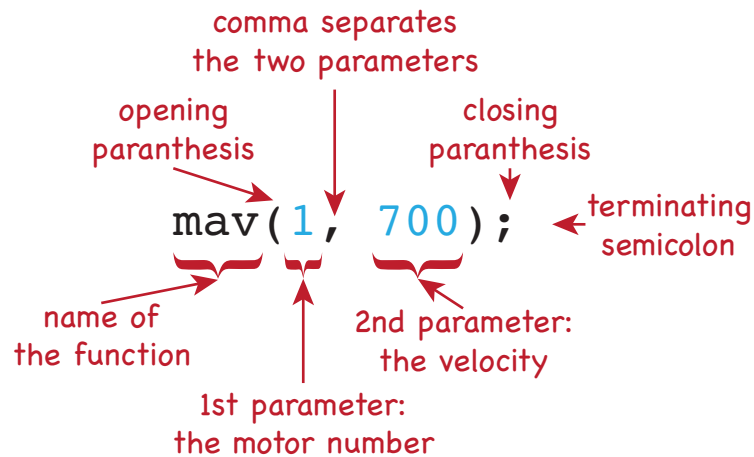


Figure 5.15: The structure of the `mav` function call

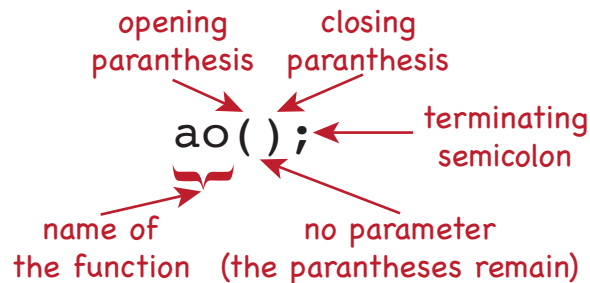


Figure 5.16: The structure of the `ao` function call

ponder the following questions:

- How many motors are mounted on the robot?
- What happens if you turn only one motor on (if unsure try to walk with only one leg moving)?
- How is it possible with the shown functions to turn two motors on?
- When is it sensible to turn the motors off again? Directly after they were started?
- What happens with a running motor if the function `wait_for_milliseconds` is called?

So a possible approach for moving the robot is listed below:

1. Turn both motors on
 - a) Start motor no. 1
 - b) Start motor no. 2

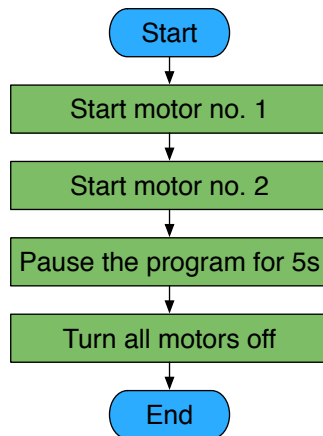


Figure 5.17: The design of the first movement program as a flowchart

2. Pause the program for 5 seconds
3. Turn all motors off

The design is also shown as a flowchart in figure 5.17. As a first step we can write down our program design into our program with comments (listing 5.5). Afterwards it is only a matter of

```

1 // First movement
2 // Created on Fr, March 6 2015
3
4 int main()
5 {
6     // Start motor no. 1
7     // Start motor no. 2
8     // Pause the program for 5 seconds
9     // Turn all motors off
10    // End the program
11    return 0;
12 }
  
```

Listing 5.5: The design of the first move program written as comments

writing the correspondent function to the comment and the program is ready to run (listing 5.6).

Warm-up task: faster and faster Try to make the robot move with different velocities in one program. Start with a slow velocity and increase the velocity after a few seconds. You can further increase the velocity after an additional few seconds, so that the robot moves with 3 different velocities.

```
1 // First movement
2 // Created on Fr, March 6 2015
3
4 int main()
5 {
6     // Start motor no. 1
7     mav(1, 700);
8     // Start motor no. 2
9     mav(2, 700);
10    // Pause the program for 5 seconds
11    wait_for_milliseconds(5000);
12    // Turn all motors off
13    ao();
14    // End the program
15    return 0;
16 }
```

Listing 5.6: The complete first move program

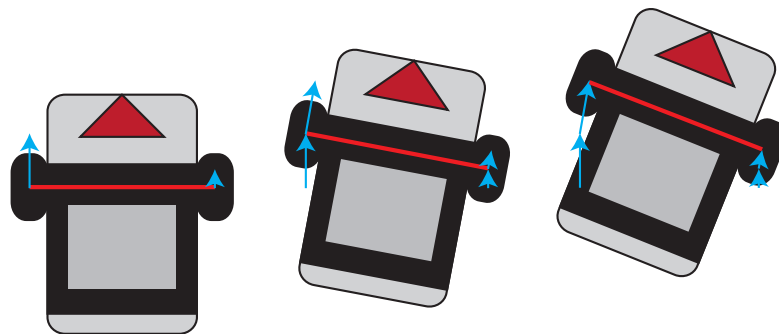


Figure 5.18: Different velocities per wheel let the robot drive a curve

Understanding movement

Most students will experience a slight turn when the robot moves, even if both velocities used in the `mav`-function have the same value. This is due to little differences in the mechanical construction of the chassis (e.g. mounting of the wheels) and in the abrasion of the mechanical parts of the motors. When wheels move with different velocity you get a curve (figure 5.18).

The knowledge of this fact can be used to

1. increase the difference between the two motor velocities to make a sharp turn
2. use a small difference between the two motor velocities to compensate the mechanical differences (when the left wheel is turning always a bit slower than the right wheel even

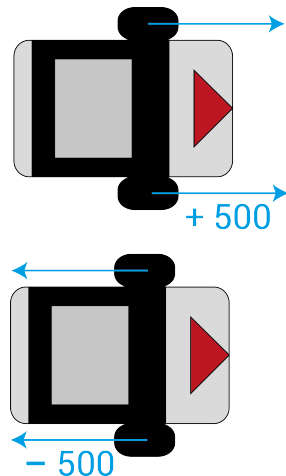


Figure 5.19: A negative number for the velocity parameter results in driving backwards

when they should go at the same velocity then increasing the velocity for the left wheel a bit might make the robot go a straight line)

Warm-up task: turn around Let the robot turn for a full round (360°). Since we can't work with sensors yet the time needed for a full turn must be estimated and tried.

The mav-function allows also negative numbers for the velocity-parameter resulting in reversing the movement direction. So to let the robot drive backwards you simply have to inverse the velocity number (figure 5.19). Which direction is considered forward and which backwards is actually dependent on the way the motors are plugged into the controller. A small LED-light at the controller indicates which direction the motor is turning for the controller: green if the controller thinks the motor turns forward (+) and red if the controller thinks the motor turns backwards (-).

Warm-up task: forward-backward Let the robot go forward for a little while and then return to your starting point without turning the robot around.

5.4 Introduction of the first challenges

Didactical considerations

All movement eventualities are introduced, so it's time for presenting the challenges to the students. It is beneficial if the students can think about the possible solutions of a challenge, but it is not intended that they start coding them immediately. Instead the challenges should be coded at the end of the next chapter (writing your own functions) since the usage of the knowledge there will be helpful and there is no need to invent new challenges for the next chapter.

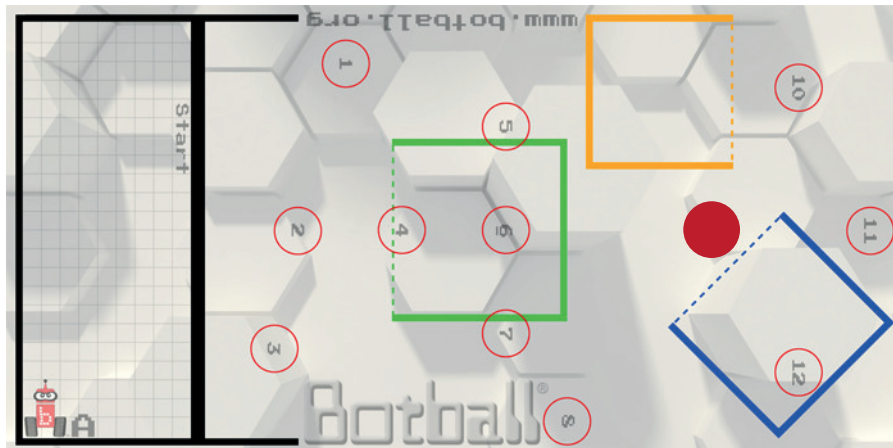


Figure 5.20: The scenario for challenge 1

The first three challenges require exact driving. The fourth and the fifth challenges are appealing on a more creative level and result in a more exhibition-like presentation. It is also possible to split these challenges and make the first three in form of a competition (without functions) and using the fourth and fifth challenge to practice the usage of functions.

For the challenges I used mats from the Junior Botball Challenge program ¹ because they are easy to use and the challenges can be easily adapted with them. But it is also a good possibility to draw the shapes on a flip chart paper or to use tape to line out the challenge area on the floor.

Challenge 1: Moving an object

Prerequisites (figure 5.20):

- A light object which can be moved with a slight push is placed on circle number 9

Start with your robot in the starting area. Then the robot drives to the object and moves visibly. The object must not leave the circle: one part of the object must still touch the circle. Afterwards the robot has to go back to the starting area. The challenge is only solved if the robot is standing inside the starting area after it has returned.

Challenge 2: Going around obstacles

Prerequisites (figure 5.21):

- An object is placed on circle number 2
- An object is placed on circle number 9
- An object is placed between the circles 5 and 6

¹<https://www.juniorbotballchallenge.org>

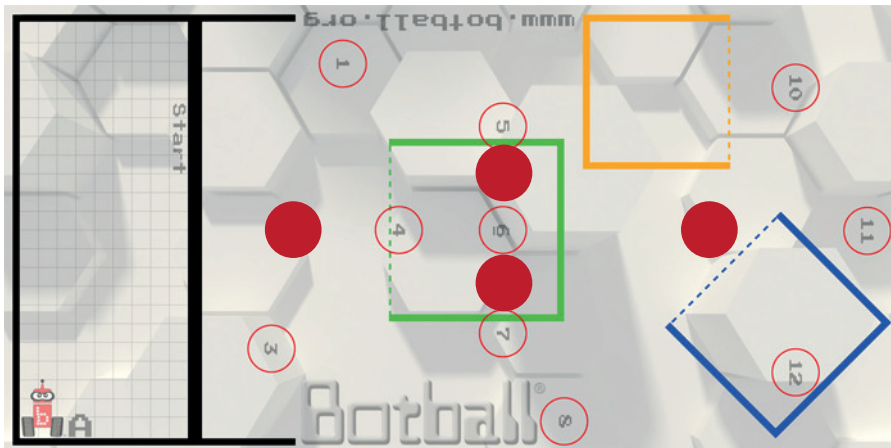


Figure 5.21: The scenario for challenge 2

- An object is placed between the circles 6 and 7

Start with your robot in the middle of your starting area. Then the robot drives around all objects and returns to the center of the starting area. The robot must not leave the mat completely. At least one wheel has to stay on the mat (if you like to enhance the challenge: both wheels have to stay on the mat). Furthermore the robot must not move the objects (grazing is ok, as long as the objects are not moved). The challenge is only solved if the robot is standing at the center of the starting area after it has returned.

Challenge 3: Parking

Prerequisites (figure 5.22):

- Only the mat is necessary (if you do not have a mat, you need three areas similar to the green, blue and yellow outlined areas on the mat)

Start with your robot in your starting area. Then the robot needs to drive to his parking place. There are three available parking places:

- an easy green area
- a blue area of medium difficulty
- a difficult yellow area

The robot needs to park in one of the areas. When trying to park in the green area it must not touch green solid lines. Accordingly when trying to park in the blue area, blue solid lines are a no-no. And when parking in the yellow area, yellow solid lines are off-limits. The challenge is solved when the robot stands in its parking place behind the correspondent dotted line (without touching solid lines, sensors and the servo arm do not count).

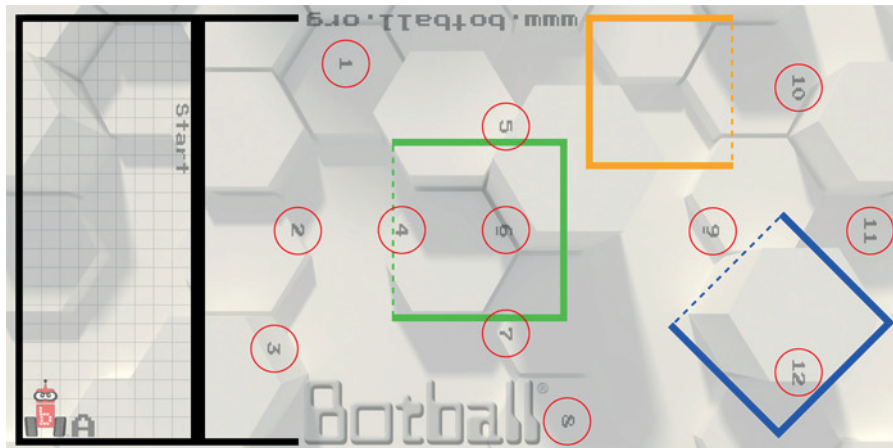


Figure 5.22: The scenario for challenge 3

Challenge 4: Dancing on the spot

Find a music which is suitable for the dance you imagine. Then let the robot dance on the spot (e.g. put the left wheel in front, and back again, put the right wheel in front, and back again, move forward, move backwards, turn around, or whatever you imagine). The challenge is only solved if the robot stands on the same spot where it has started.

Challenge 5: Dancing in a wider range

Find a music which is suitable for the dance you imagine. Then let the robot dance over an area like with ice dancing (e.g. while driving a wide circle, the robot turns around and does some pirouettes or whatever you imagine). The challenge is only solved if the robot stands roughly on the same spot where it has started.

5.5 Defining functions

Didactical considerations

I introduce function definition as a mean to break down a big task in smaller more comprehensible tasks and thereby emphasizing thinking about program design. The goals of this section for the students are:

- comprehending function definition as a mean to enhance the robots abilities
- understanding the structure of a function definition without parameters and return values
- being able to write a prototype
- using self-written functions

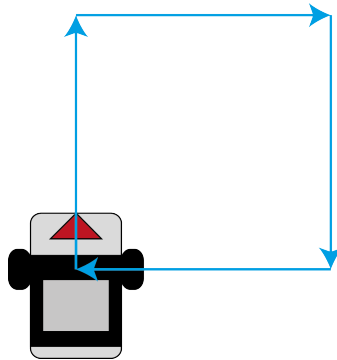


Figure 5.23: Driving a square

As experience shows motivating the students to use functions is often difficult. No program needs functions. every program can all be written with “spaghetti-code” and several copy and pastes. But when the usage of function is introduced as “teaching SaSbot (or your robot) new instructions” the appreciation of it is easily understood.

To emphasize this point and as a repetition I use the square assignment as a starting point. The resulting program is quite long with lots of repetitions. And tinkering with the mav-parameters for driving straight or turning is tedious. But it can be shortened easily with functions (and even more with loops later on) and this shows the advantage of it quite clearly.

When introducing function definitions it is important to relate to the known structure of the main-function. To ease the already complicated process, I only offer one approach: working with prototypes. With this approach no other considerations need to be made and it can easily lead to writing your own libraries.

When covering function calls again the sameness to the already known function calls need to be stressed.

After this section the students should work on their challenges, but it may be necessary to discuss possible functions with them and encourage the usage of functions, since the concept is still quite new to them.

Content for students

Warm-up task: Driving a square Program your robot for driving a square. Try to adjust the mav-parameters so that the robot is standing in the same position as at the beginning after it finished driving the square.

Before starting to code think about your design. Maybe the following questions can help:

- What different kind of movements are involved (driving forward, driving backward, turns - in which direction etc.)
- What steps are necessary to complete the task
- What instructions do you have for implementing these steps

A first design proposal could be:

1. Drive straight for 3 seconds
2. Turn right
3. Drive straight for 3 seconds
4. Turn right
5. Drive straight for 3 seconds
6. Turn right
7. Drive straight for 3 seconds
8. Turn right
9. Turn all motors off

A more refined design could be:

1. Turn both motors on with 700 ticks per seconds
2. Pause the program for 3000 ms
3. Turn the first motor on with 1000 ticks per seconds
4. Turn the second motor off
5. Pause the program for 1450 ms
6. Turn both motors on with 700 ticks per seconds
7. Pause the program for 3000 ms
8. Turn the first motor on with 1000 ticks per seconds
9. Turn the second motor off
10. Pause the program for 1450 ms
11. Turn both motors on with 700 ticks per seconds
12. Pause the program for 3000 ms
13. Turn the first motor on with 1000 ticks per seconds
14. Turn the second motor off
15. Pause the program for 1450 ms
16. Turn both motors on with 700 ticks per seconds

17. Pause the program for 3000 ms
18. Turn the first motor on with 1000 ticks per seconds
19. Turn the second motor off
20. Pause the program for 1450 ms
21. Turn all motors off

As you can already see, the program will be quite long. Keep in mind that the time you need to wait until your robot finishes its 90° turn may vary. The resulting program is listed below:

```
1 // Driving a square - first try
2 // Created on Sat, March 7 2015
3
4 int main()
5 {
6     // Turn both motors on with 700 ticks per seconds
7     mav(1, 700);
8     mav(2, 700);
9     // Pause the program for 3000 ms
10    wait_for_milliseconds(3000);
11    // Turn the first motor on with 1000 ticks per seconds
12    mav(1, 1000);
13    // Turn the second motor off
14    mav(2, 0);
15    // Pause the program for 1450 ms
16    wait_for_milliseconds(1450);
17    // Turn both motors on with 700 ticks per seconds
18    mav(1, 700);
19    mav(2, 700);
20    // Pause the program for 3000 ms
21    wait_for_milliseconds(3000);
22    // Turn the first motor on with 1000 ticks per seconds
23    mav(1, 1000);
24    // Turn the second motor off
25    mav(2, 0);
26    // Pause the program for 1450 ms
27    wait_for_milliseconds(1450);
28    // Turn both motors on with 700 ticks per seconds
29    mav(1, 700);
30    mav(2, 700);
31    // Pause the program for 3000 ms
32    wait_for_milliseconds(3000);
33    // Turn the first motor on with 1000 ticks per seconds
```

```

34     mav(1, 1000);
35     // Turn the second motor off
36     mav(2, 0);
37     // Pause the program for 1450 ms
38     wait_for_milliseconds(1450);
39     // Turn both motors on with 700 ticks per seconds
40     mav(1, 700);
41     mav(2, 700);
42     // Pause the program for 3000 ms
43     wait_for_milliseconds(3000);
44     // Turn the first motor on with 1000 ticks per seconds
45     mav(1, 1000);
46     // Turn the second motor off
47     mav(2, 0);
48     // Pause the program for 1450 ms
49     wait_for_milliseconds(1450);
50     // Turn all motors off
51     ao();
52     return 0;
53 }

```

Listing 5.7: Driving a square - first try

The simple task results in a quite long program. Think about the following questions:

- What happens if you want to make your square bigger? How often do you need to change a value?
- What happens if you need to swap your robot, so that you need to adjust the parameters for the mav-functions or the wait_for_milliseconds functions to keep your robot going straight or turning the right angle? How often do you need to change a value?

It would be much easier if we can work with the first design proposal. Fortunately this is possible by teaching the robot a new vocabulary. This is called *defining a new function*. To work with a new function you need to do three steps:

1. Define what the new command is and what the robot needs to do *after* the main-function – this is the actual *function definition*
2. Make the new function known at the beginning of the program *before* the main function starts – this is called writing the *prototype*
3. Now you can use the new function in the same way as already predefined functions (like mav or ao) – this is the *calling of the function*

```

// Driving a square - second try
// Created on Sat, March 7 2015

int main()
{

    return 0;
}

void drive_straight()
{
    // Turn both motors on
    mav(1, 700);
    mav(2, 700);
    // Pause the program for 3s
    wait_for_milliseconds(3000);
}

```

Figure 5.24: The definition of the function drive_straight

First step: Defining the function after the main-function

To define a function you need to come up with a name for the new instruction. Since we want to simplify the driving program we need a function which enables the robot to drive straight forward. So a good name would be `drive_straight`. Note that spaces and other special characters than underscore “_” are not allowed in a name.

Now the robot needs to know, what it has to do, when this instruction is called. So after defining the name a set of instructions need to be specified. The overall structure of the new function is similar to the structure of the *main-function* aside from starting with `void`. The complete definition is shown in figure 5.24.

Second step: Writing the prototype

The job of the prototype is introducing the new function to the robot, so that the robot knows that there is a new instruction it needs to adhere. It is written above the main-function and has the same content as the header of the newly defined function. Other than the header of the function the prototype ends with a semicolon (;). The process is illustrated in figure 5.25.

Third step: calling the new function

The new function can be called in the same way as already defined functions. It has no parameter (we did not define any) so the structure of the resulting call is shown in figure 5.26. Without calling the function the robot only has learned a new vocabulary which it does not use. To make it go straight the new function needs to be used in the main-function. The main-function is always the starting-point for any program. Within the main-function every function can be

```

// Driving a square - second try
// Created on Sat, March 7 2015
void drive_straight();

int main()
{
    return 0;
}

void drive_straight()
{
    // Turn both motors on
    mav(1, 700);
    mav(2, 700);
    // Pause the program for 3s
    wait_for_milliseconds(3000);
}

```

copy the header of the function to a line above the main-function and add a ;

Figure 5.25: Adding the prototype to the definition of the function drive_straight

opening parenthesis closing parenthesis

drive_straight(); ← terminating semicolon

name of the function no parameter (the parentheses remain)

Figure 5.26: The structure of calling of the function drive_straight

called as often as it is needed. Since we want to drive a square, we need to drive straightforward four times (figure 5.27).

For completing the square a second function needs to be defined. A good name would be turn_right. You can practice writing the function on your own and compare your results with the listing below:

```

1 // Driving a square - first try
2 // Created on Sat, March 7 2015
3
4 // prototypes for the functions
5 void drive_straight();
6 void turn_right();

```

```

// Driving a square - second try
// Created on Sat, March 7 2015
void drive_straight();

int main()
{
    drive_straight();
    drive_straight();
    drive_straight();
    drive_straight();
    return 0;
}

void drive_straight()
{
    // Turn both motors on
    mav(1, 700);
    mav(2, 700);
    // Pause the program for 3s
    wait_for_milliseconds(3000);
}

```

Figure 5.27: The calling of the function `drive_straight` 4 times within the `main`-function

```

7
8 // starting point of the program - main-function
9 int main()
10 {
11     drive_straight(); // 1st call of drive_straight
12     turn_right(); // 1st call of turn_right
13     drive_straight(); // 2nd call of drive_straight
14     turn_right(); // 2nd call of turn_right
15     drive_straight(); // 3rd call of drive_straight
16     turn_right(); // 3rd call of turn_right
17     drive_straight(); // 4th call of drive_straight
18     turn_right(); // 4th call of turn_right
19     // Turn all motors off
20     ao();
21     return 0;
22 }
23
24 //function definitions
25 void drive_straight()
26 {

```

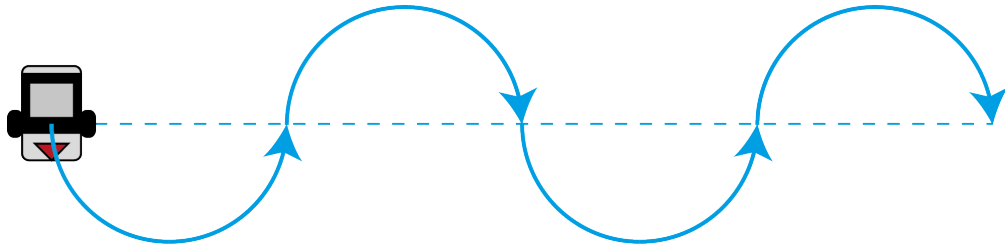


Figure 5.28: The robot drives a wiggly line

```

27 // Turn both motors on with 700 ticks per seconds
28 mav(1, 700);
29 mav(2, 700);
30 // Pause the program for 3000 ms
31 wait_for_milliseconds(3000);
32 }
33
34 void turn_right()
35 {
36 // Turn the first motor on with 1000 ticks per seconds
37 mav(1, 1000);
38 // Turn the second motor off
39 mav(2, 0);
40 // Pause the program for 1450 ms
41 wait_for_milliseconds(1450);
42 }

```

Listing 5.8: Driving a square - second try

The resulting program is not much shorter, but it is a lot more structured. Think again about the following questions:

- What happens if you want to make your square bigger? How often do you need to change a value?
- What happens if you need to swap your robot, so that you need to adjust the parameters for the mav-functions or the wait_for_milliseconds functions to keep your robot going straight or turning the right angle? How often do you need to change a value?

Warm-up task: going wiggly Let the robot follow a wiggly line (figure 5.28). What functions make sense? Think about the principal movements necessary for this driving style. Implement the program without using mav or wait_for_milliseconds in the main-function.

5.6 Functions with parameters

Didactical considerations

Parameters make the functions configurable. the goals of this section for the students are:

- understanding the concept of variables
- comprehending the term declarations and parameter
- understanding how parameters make functions configurable

For the usage of parameters the introduction of variables is necessary. Variables are often explained with boxes which lead to several misconceptions [67]:

- a variable can hold more than one value
- the value is deleted when read
- confusing initial value with actual value

Especially the first two misconceptions are easily comprehensible when thinking about the box metaphor. So I invented another concept - the magical glass jar. It needs to be a glass container with lid to enable looking at the content without opening the lid and taking something out (to counter the second misconception). The container needs to be magical because you cannot add another value respectively another thing to the content already there and you cannot empty the container. When calling the jar magical I remind the students that not everything is the same as with real jars.

It is also advisable to give the students a technical background about how variables are stored in RAM and how retrieving and storing information works to give the students additional information which leads to the rules of the “magical jars”.

Content for students

SaSbot – searching the edges of the room When I want SaSbot to find something it needs to cover a whole room for searching. For the beginning it is easier to think about searching along the room walls. But nearly no room has a square form. If we think about an easy form, we can assume a rectangle. But most rooms will be more complicated. So how can we drive a rectangle form or even a more complicated form in a structured way?

Think about the square-exercise. We used two functions to solve the problem with functions: one for driving straight and one for the turn. How many functions do we need, when we want to drive a rectangle pattern? How many principal movements do we have here?

There are still only two principal movements:

1. drive straight, but with different lengths
2. turn right

The drive_straight function only needs to be configurable, to enable it to go different lengths. To make a function configurable, we need special types of variables (parameters).

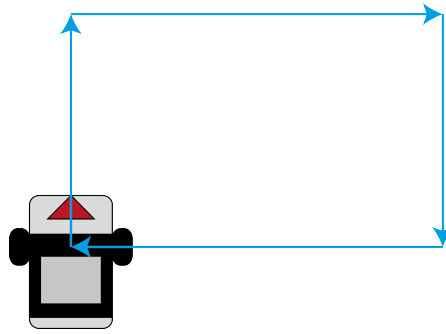


Figure 5.29: The robot drives a rectangle

Variables

Variables can be imagined as glass jars. They have similar characteristics:

- both must exist before you can put something into it
- both can store content (closing the lid on the jar)
- both are available in different sizes and forms
- both allow viewing and checking the content just by looking

But variables have more characteristics they are like *magical glass jars*:

- you must know their name to find them
- you cannot empty them
- if you put something new into the variable resp. the magical jar, the previous content disappears
- the size determines what you can store in it
- when the program ends, all variables/magical jars disappear

If you want to create a new variable (magical jar) you need to know two facts:

1. a *name*, which you can invent on your own (some rules need to be adhered to). It should reflect the thing you want to store in your variable.
2. a *datatype*, which determines the size and the kind of values you can store in it.

So we want to vary the driving length of our robot, when it drives straight. One possibility is to vary the time we pause the program before issuing another motor command. The time is set in milliseconds – a number with no decimal point allowed: an integer. So for the variable we have the two parts:


```


// Driving a rectangle
// Created on Sat, March 7 2015

int main()
{

    return 0;
}

void drive_straight(int time)
{
    // Turn both motors on
    mav(1, 700);
    mav(2, 700);
    // Pause the program for time
    wait_for_milliseconds(time);
}

```



a new variable named time is created here

looking at the content of the variable time and using it for determining the length of the pause

Figure 5.30: Declaring a parameter and using it within the function

1. name: `time` – since it should store a time, this name is easily understandable
2. datatype: `int` – `int` is the abbreviation of integer and it is the type of value we want to store.

To create the variable you need to write the datatype and the name in that order:

datatype name

That means for our variable we need to write:

```
int time
```

This line creates a new variable (a new magical jar). After the creation the content is undefined, it is simply not known what is stored in it.

Using variables as parameters

Parameters are used in functions and specified, when the function is defined. Remember the three steps necessary to get a new function to work.

1. Define what the new command is and what the robot needs to do *after* the main-function – this is the actual *function definition*
2. Make the new function known at the beginning of the program *before* the main function starts – this is called writing the *prototype*
3. Now you can use the new function in the same way as already predefined functions (like `mav` or `ao`) – this is the *calling of the function*


```

// Driving a rectangle
// Created on Sat, March 7 2015
void drive_straight(int time);

int main()
{
    return 0;
}

void drive_straight(int time)
{
    // Turn both motors on
    mav(1, 700);
    mav(2, 700);
    // Pause the program for time s
    wait_for_milliseconds(time);
}

```



copy the header of the function to a line above the main-function and add a ;

Figure 5.31: The prototype must contain the parameter too

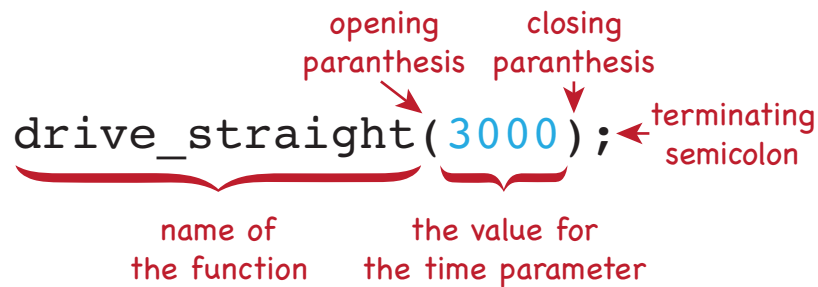


Figure 5.32: The structure of calling drive_straight with parameter

To use a variable as parameter it needs to be created between the parentheses of the function definition (first step). This causes the function to provide a new variable (= magical jar) every time it is called. This variable must be filled when the function is called. Within the function the variable can be read with just writing the name (figure 5.30).

The next step is to provide the program with the prototype of the new function. This is achieved in the same way as with functions without parameters (figure 5.31).

Before moving on think about the following question:

- Look at figure 5.31: for how long will the robot drive straight?

Remember: without calling the function it just is a piece of unused vocabulary and the robot does not drive a little bit. Even more in this state it is impossible to say how long the robot will drive if the function is called, because the parameter-variable is still undefined and therefore we

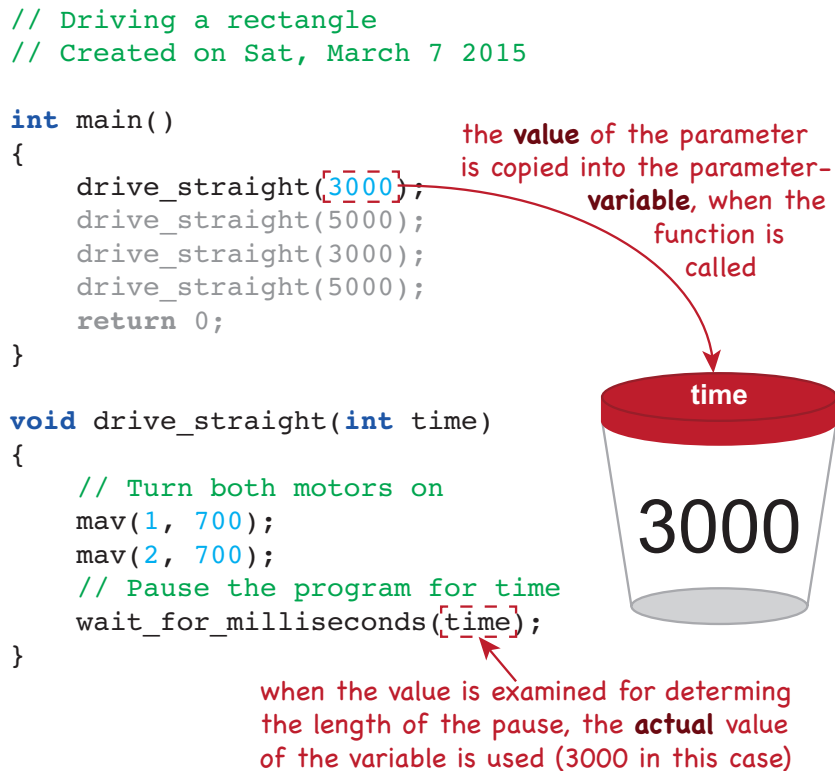


Figure 5.33: The structure of calling drive_straight with parameter

cannot determine how long the program must pause. For using this function we must provide a value for our variable. This is achieved the same way as with every other function having parameters – it must be written between the parentheses when calling it (figure 5.32)

When the function is called, the value between the parentheses will be copied into the new variable. The value of the variable is no longer undefined. When the parameter is used to determine the length of the pause, the actual value, which is currently stored in the variable, is examined and used as value for the wait_for_milliseconds function (figure 5.33).

Think about the following questions :

- How often is the function drive_straight called?
- Is it always called with the same value for the parameter? Which value(s) are used?
- What does the usage of different values for the parameters mean for the driving pattern? How does the robot drive?
- For how many seconds will the robot drive all in all?

Only the first call of the drive_straight-function is visualized in figure 5.33. Try to visualize the following function calls in the same way for answering the questions.

The complete rectangle program including the turn_right-function is listed below:

```
1 // Driving a rectangle
2 // Created on Sat, March 7 2015
3 // prototypes for the functions
4 void drive_straight(int time);
5 void turn_right();
6
7 // starting point of the program - main-function
8 int main()
9 {
10     drive_straight(3000);
11     turn_right();
12     drive_straight(5000);
13     turn_right();
14     drive_straight(3000);
15     turn_right();
16     drive_straight(5000);
17     turn_right();
18     // Turn all motors off
19     ao();
20     return 0;
21 }
22
23 //function definitions
24 void drive_straight(int time)
25 {
26     // Turn both motors on with 700 ticks per seconds
27     mav(1, 700);
28     mav(2, 700);
29     // Pause the program for time ms
30     wait_for_milliseconds(time);
31 }
32
33 void turn_right()
34 {
35     // Turn the first motor on with 1000 ticks per seconds
36     mav(1, 1000);
37     // Turn the second motor off
38     mav(2, 0);
39     // Pause the program for 1450 ms
40     wait_for_milliseconds(1450);
41 }
```

Listing 5.9: Driving a rectangle

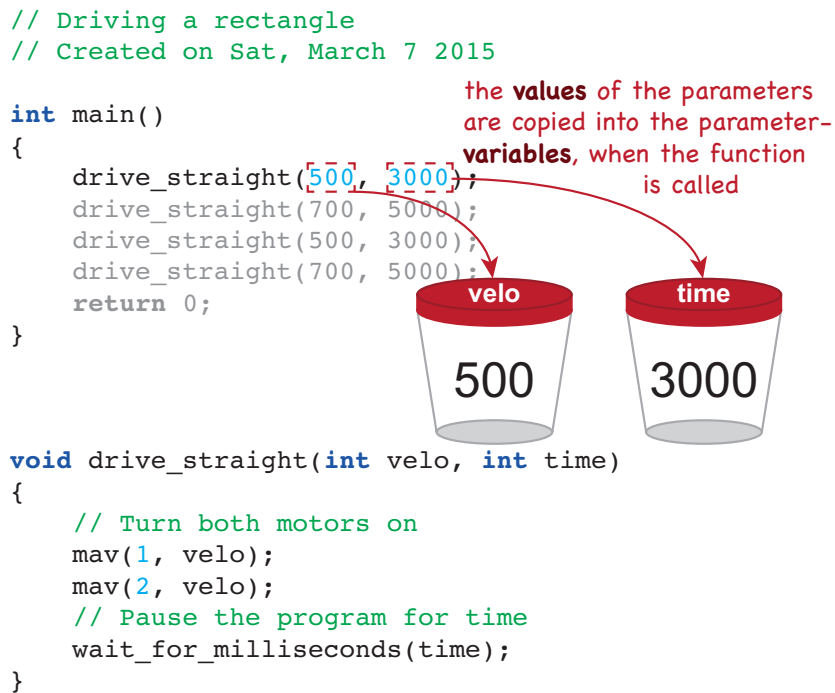


Figure 5.34: The structure of calling drive_straight with parameter

Warm-up task: Another way to determine the length Is there another way of determining how far the robot is going? Hint: to go farther than before, one can either move with the same velocity and increase the time, or ... Modify the rectangle program, so that the other way to modify the robot's driving distance is used.

It is also possible that a function has more than one parameter. The parameters are separated by a comma. The order of the calling parameters determine which value is stored in which parameter-variable (figure 5.34).

Warm-up task: Wiggly line with only one function Change the wiggly line program in a way that only uses one function instead of two. Hint: Examine the differences between the two functions used before. These are the starting-points for having parameters instead of fixed values.

5.7 The clearance competition

Didactical considerations

The clearance competition is designed to practice functions with parameters in a fun way. The goal is to remove small objects placed on the mat to clear the whole field. It is necessary to have

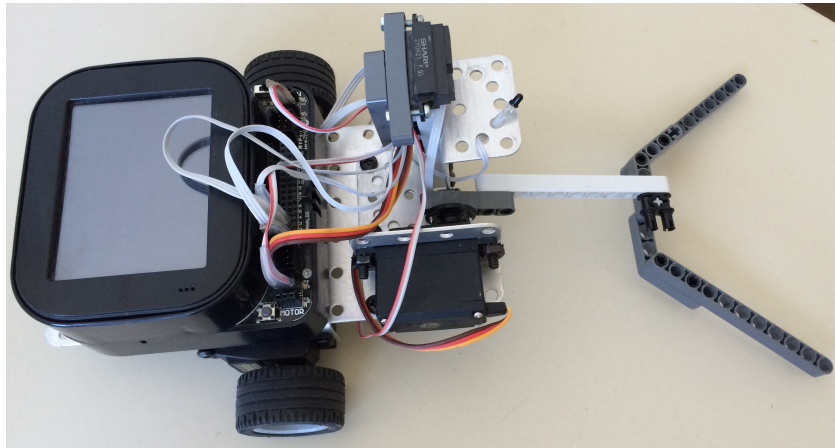


Figure 5.35: A simple tool for clearing the mat

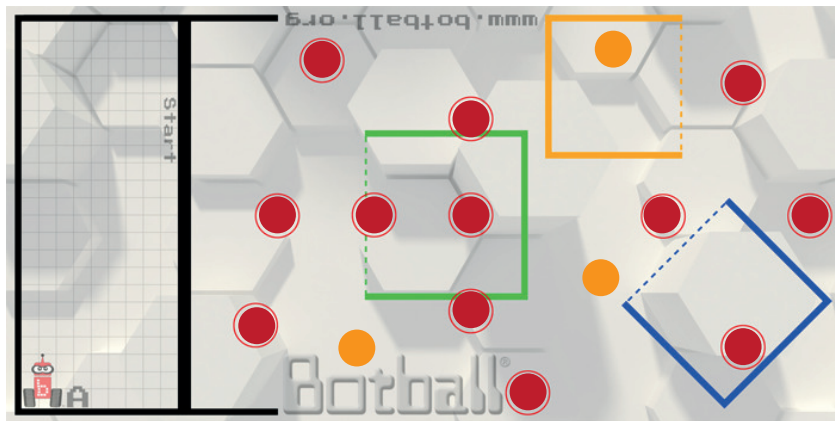


Figure 5.36: The field of the clearing competition

objects which are easily moveable, but not as easily as ping pong balls. I used fluffy balls for this competition.

Aside from using functions this competition has also a mechanical part, as the students need to develop a tool for the robot to move the objects. A simple first draft is shown in figure 5.35.

It is also possible to introduce the servo-motor beforehand to add an extra challenge and consolidate function calling.

Since students tend to write “spaghetti-code” when having time restrictions it is crucial to have the design as a part of the winning condition and to provide enough time for preparation.

The competition

Prerequisites (figure 5.36):

- A small moveable object is placed on every number on the field

- Three additional objects are placed randomly wherever there is room (examples shown as orange dots)

The robot needs to start in the starting box where no part is breaking the boundary. Clear the field within 5 minutes. It is necessary to reach the starting box at the end of the 5 minute limit to make every cleared object count otherwise only half of the objects count (round up). An object counts as cleared when at least part of it touches the floor.

Winning conditions:

1. The team who cleared most objects wins
2. If two teams have cleared the same amount of objects, the team with the more efficient code wins (referee/teacher decides – as a rule of thumb the fewer line of codes the better, lines containing only comments, parentheses or white spaces do not count)
3. If the winner is still not determinable, the team needing less time wins

5.8 Using sensors

Didactical considerations

Using sensors requires the usage of functions with return values. So the goals for the students are:

- being able to declare variables outside of the parameter area
- being able to store return values in a variable
- comprehending the differences between digital sensors and analog sensors
- understanding the format string and additional parameters for printf

The `mov` function is currently not usable in loops. Since loops are necessary when working with sensor conditions this is a good point to introduce a new motor-function `motor`. It can be used in a there-and-back-program, which is also a good repetition for functions. This assignment can be extended in the next section to show sensor driven movement using while loops.

When working with variables I write the declaration in a separate line, to emphasize the actual creation of the variable taking place there.

If there is time I recommend to consolidate the variable and sensor topic with doing some calculations with the results of the analog sensors especially the distance sensor. To do this it is necessary to explain characteristic curves and to actually measure them using the `printf`-function at different distances and so obtaining the necessary values for the curve.

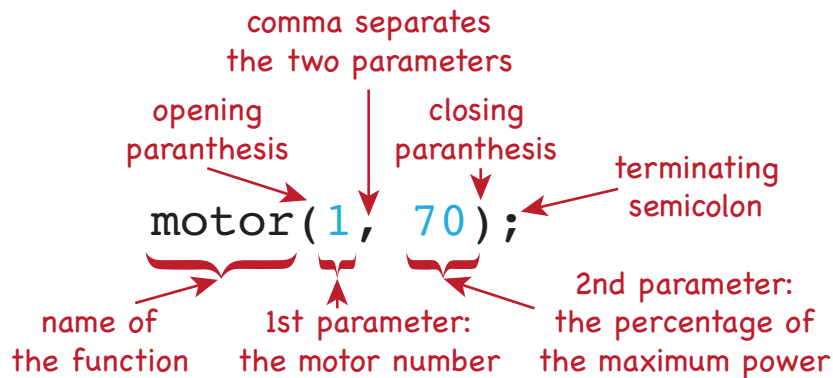


Figure 5.37: The motor-function for use with sensors

Content for students

When working with sensors it is advised to use another function for controlling the motors. This function is called `motor`. The main difference is that instead of a velocity you specify the percentage of power to be used. As such it is dependent on the energy level of the battery. The structure of the function call is shown in figure 5.37

Warm-up task: there and back again Make your robot drive to the end of the mat, turn around and go back to his starting point using the new `motor` function. What functions make sense? Do the functions have parameters?

SaSbot – Sensing the environment SaSbot is designed to seek things for me. To search the environment it needs to be aware of the environment. The sensors of a robot bring this awareness into being. What kind of sensors does SaSbot need to fulfill its designated role?

The robot we are working with has sensors belonging to one of two categories:

1. digital sensors: return either the value 0 or the value 1
2. analog sensors: return any value within the range within 0 and 1023

Working with a touch sensor

The touch sensor is a digital sensor. This means its value can be 1 (touch sensor is pressed) oder 0 (touch sensor is released). To ask the sensor which value it currently has you need to do two things:

1. Create a variable for storing the answer from the sensor
2. Call a function which asks the sensor for its value and store its answer in the variable

To implement these two steps in your program you have to use the structure shown in figure 5.38. When running these lines of codes the following things happen (see also figure 5.39):

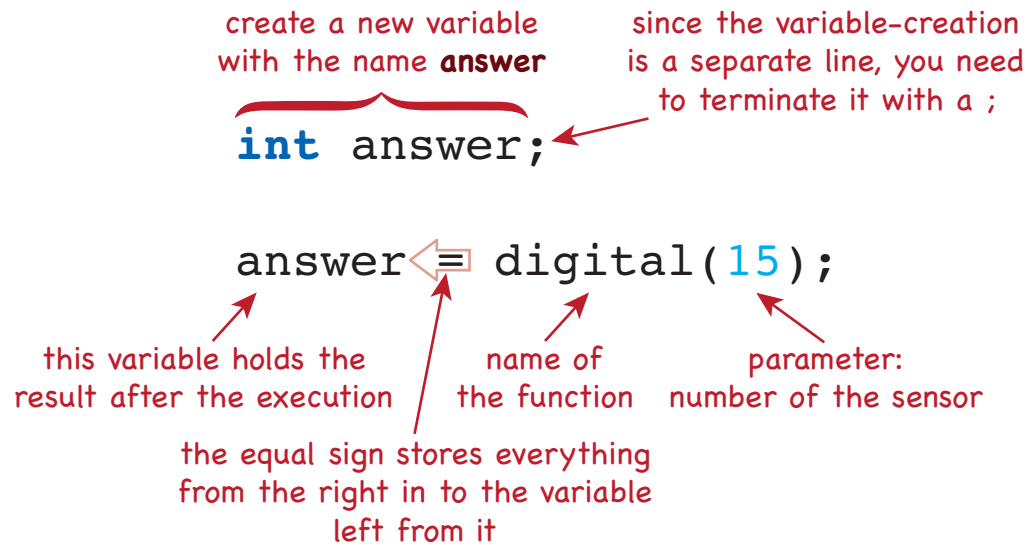


Figure 5.38: How to call a function, which asks a sensor of its value

1. The variable (magical jar) is created. It has the name `answer` and can store integer numbers (0 and 1 are integer numbers)
2. The sensor with the number 15 is asked what its current value is
3. The sensor answers with the current value. Let's assume it is 0 (it could be 1 as well, but it is easier to imagine a concrete value and we need to choose one, hence 0).
4. The equal sign takes the 0 and stores it in whatever variable (magical jar) is left of it.
5. We wrote the variable (magical jar) with the name `answer` left of the equal sign, so the value 0 is stored there for further use.

The robot can tell us this value via the `printf`-function. For this purpose the `printf`-function needs to be extended:

- A placeholder representing the value must be inserted into the text. For integer numbers you can use `%d`.
- After the text the variable holding the value needs to be written as second parameter (separate by a comma from the text)

The correspondent listing is shown in listing 5.10

```

1 // Printing a sensor value
2 // Created on Tue, March 8 2015
3
4 int main()
```

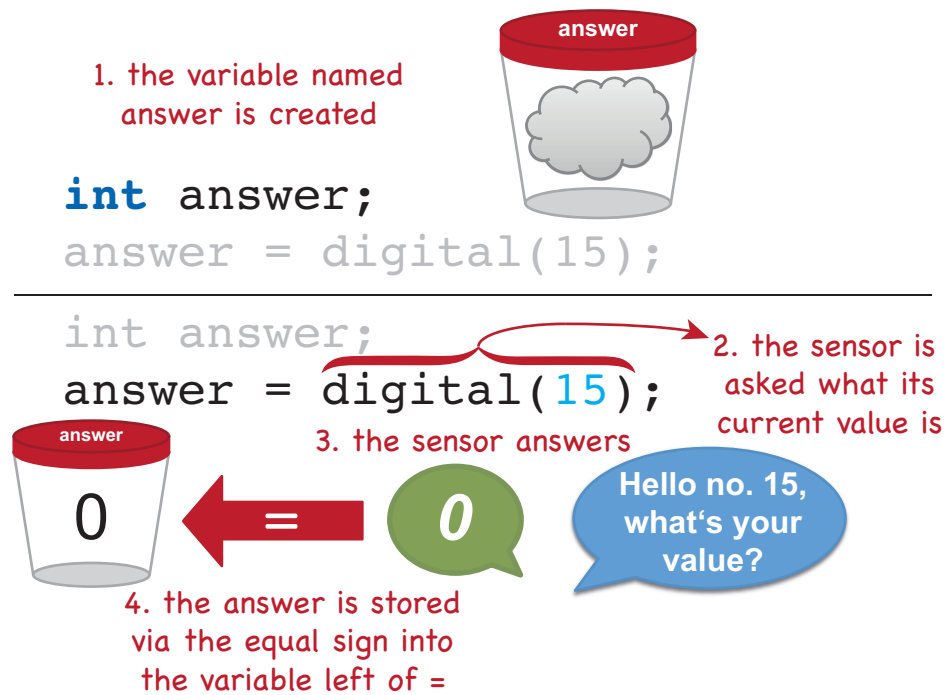


Figure 5.39: The internal process when asking a sensor of its value

```
5 {
6     // Create the variable for the sensor value
7     int answer;
8     // ask sensor no. 15 for its value
9     answer = digital(15);
10    // print the value in the answer variable
11    printf("Sensor no. 15 has the value: %d\n", answer);
12    return 0;
13 }
```

Listing 5.10: Asking a sensor for its value

Warm-up task: try the sensor Try the program on your robot? What value does the robot tell you its sensor has? How can you get the robot to show you another value from the sensor (remember the touch sensor is a digital sensor and can have either 0 or 1 as its value)?

Working with other sensors

The principle of working with the other sensors is similar working with the touch sensor with the difference that the other sensors are all analog sensors. For asking an analog sensor of its value

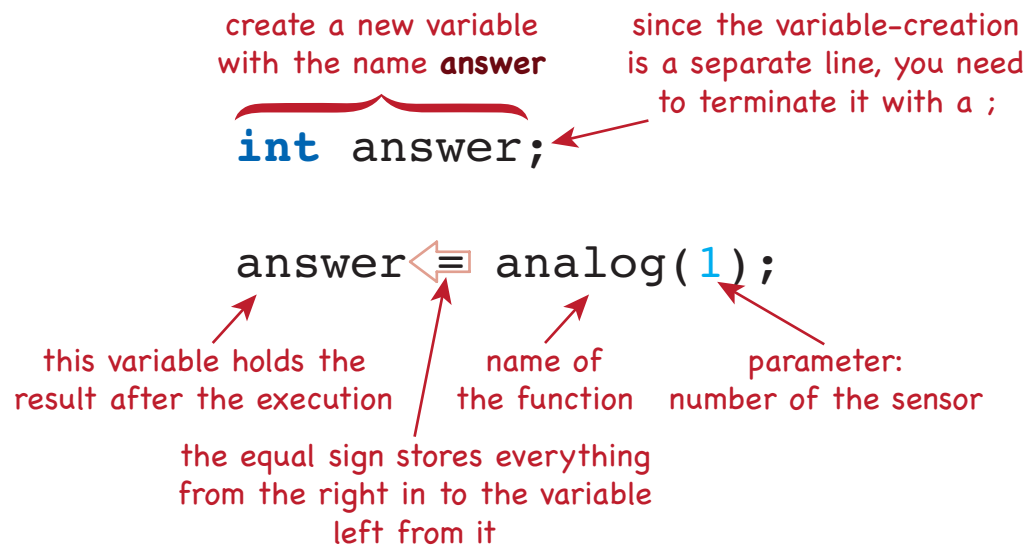


Figure 5.40: How to call a function, which asks a analog sensor of its value

you need the function with the name `analog`, which works the same way as the `digital`-function (figure 5.40)

Only when working with the distance sensor the pull-up resistor needs to be set. This can be achieved on the sensor screen directly on the controller (cf. appendix section A.2).

Warm-up task: other sensors Ask every sensor which is currently mounted on the robot of its value and let the robot print the results on its screen. Observe how different circumstances affect the results (nearer or farther away from an obstacle, brighter or darker light, standing on brighter or darker surface, ...). The sensors are mounted on the following ports:

- Distance sensor: analog sensor on port 0 (toggle pull-up)
- Reflectance sensor: analog sensor on port 1
- Light sensor: analog sensor on port 2
- Touch sensor: digital sensor on port 15

5.9 Loops

Didactical considerations

To respond to sensor inputs it is necessary to introduce loops. To keep the API simple and comprehensible the Botball API does not use an interrupt driven sensor framework. Instead it is necessary to implement a busy-waiting approach. The goals for the students are

- comprehending the basic principle and necessity of loops

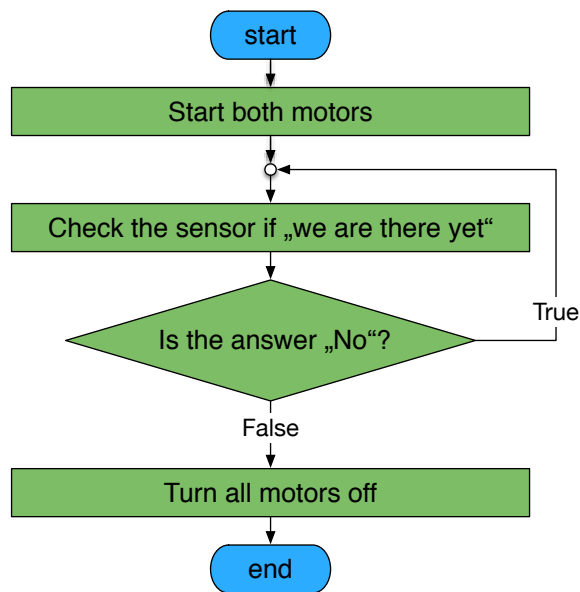


Figure 5.41: Driving until “we are there”

- understand the process of busy waiting
- gaining the ability to use looping-conditions

To explain loops with busy waiting I use the scene in Shrek 2, where the donkey always asks “Are we there yet?”. This kind of construct is best solved with a post-test loop. Usually introductory programming starts with explaining the while loop because of its similarity with the if-statement. Although it is easy to switch between the different loop-types for an experienced programmer using a pre-test loop would only complicate matters for a novice.

Content for students

We drive with the robot on a time controlled basis. Since we usually want to reach a specific location, we need to estimate the time on a try-and-error basis. Now we can use sensors to determine the point where we want to go. For example, we can use the touch sensor to determine if the robot has reached a wall. The principle behind this approach is similar to the “Are we there yet?”-scene in the movie Shrek 2². The robot starts the motors (with no waiting for a time) and then asks the sensor “Are we there yet?”. While the answer is “No” the robot needs to keep going. This principle is shown in figure 5.41.

The instruction to have such a repetition is called *loop* and here we can use the *do-while*-loop shown in figure 5.42. The complete listing with explanations is shown in figure 5.43. A step-by-step sequence when executing the program is shown in the slides in appendix B.

²<https://youtu.be/basofea2UEs>

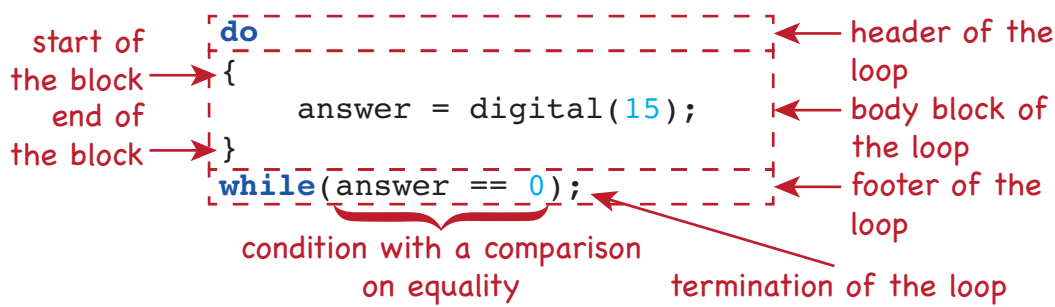


Figure 5.42: Structure of the do-while-loop

Warm-up task: There – ouch – and back again The robot drives until it touches the wall and then turns to go back to its starting point.

When working with analog sensors and loops, comparison on equality is often not meaningful, since it is difficult to check on a particular value in the range of 0 to 1023. Instead comparison using greater-than or less-than-signs are more sensible. The signs which can be used for comparison are shown in table 5.6

Operator	Example	Meaning
==	answer == 0	answer is equal to 0
!=	answer != 0	answer is <i>not</i> equal to 0
<	answer < 100	answer is less than 100
<=	answer <= 100	answer is less than or equal to 100
>	answer > 100	answer is greater than 100
>=	answer >= 100	answer is greater than or equal to 100

Table 5.6: Operators usable for comparison

Warm-up task: There and back again with distance sensor Modify the “There and back again” program to use the distance sensor instead of the touch sensor.

Warm-up task: There and back again with reflectance sensor Modify the “There and back again” program to use the reflectance sensor instead of the touch sensor. The robot shall move between two black lines on the floor.

Warm-up task: And there was light The robot waits until the light is turned on. Then it begins to dance and when the light goes out, it stops.

```

// Driving a square - second try
// Created on Sat, March 7 2015
void start_straight(int power);

int main()
{
    // Create a variable Create variables always at the
    int answer; ← start of the block
    // Start driving
    start_straight(50); ← The motors are started without
                        specifying how long they will go
do the
following ... → do
{
    answer = digital(15); ← ask the sensor of its
                        value and store it in
... while the
answer is 0 → while(answer == 0); ← the variable answer
    // Turn all motors off
    ao(); ← when the answer reaches a value unequal
    return 0; ← to 0 the program resumes here
}

void start_straight(int power)
{
    // Turn both motors on
    motor(1, power);
    motor(2, power);
} ← No duration specified

```

Figure 5.43: Structure of the do-while-loop

Warm-up task: point parcour Write a program to make your robot follow the points to the last one. There are 5 points in total and the next point can be found always at a 90° angle right from the last point (e.g. figure 5.44).

5.10 The find the spot competition

Didactical considerations

The goal is to find a spot within a walled area representing a room. To give the robot enough room to maneuver and making the search challenging it is necessary to make the area big enough.

- Place a spot within the field, but outside of the starting area. I recommend using tape (black or white) so that the spot cannot get out of place when a wheel passes over it.

The robot needs to start in the starting box where no part is breaking the boundary facing either the opposite short side or the opposite long side at a 90° angle. The goal is to find a spot three times within a five minute time window for each round. The spot is the same for all teams in each round but it is moved between the rounds. When the spot is found the robot needs to print the text “Here is the spot” and stop.

Winning conditions:

1. The team who found the most spots wins
2. If two teams have found the same amount of spots, the team with the more efficient code wins (referee/teacher decides – as a rule of thumb the fewer line of codes the better, lines containing only comments, parentheses or white spaces don’t count)
3. If the winner is still not determinable, the team needing less time (the sum counts) wins

5.11 Summary and outlook

My curriculum provides an entry point for teaching introductory programming with robots based on the concept in chapter 4. Thereby the development of SaSbot from a robot which can use only basic movement commands to a robot using custom functions and sensors to follow a defined search pattern is used as a guideline for introducing new programming techniques. The following programming concepts are introduced:

- basic program structure
- function calls
- function definition
- variables as parameters
- variables for storing return-values
- post-test loops

The concepts are structured in a way, that each topic can build upon the knowledge of previous topics leading to an enhanced sustainability of the knowledge. To complete the curriculum for a whole semester course the following topics should be introduced next.

- decisions
- other loop-types

This enables SaSbot to follow more sophisticated search-patterns or to include line following as a way to navigate.

This curriculum was evaluated with small adaptations at workshops held by PRIA which is described in the next chapter.

Evaluation

6.1 Method

Due to time restrictions I used a case study to evaluate my concept.

Study setup

In this case study I used workshops from schools at ISCED level 2¹. These schools had the opportunity to choose a group of students who were able to attend a basic robotics workshop in the context of the project STEMoFuture organized by the Practical robotics institute Austria (PRIA)² with the maximum timeframe of 15 hours. From the schools asking for a workshop I chose two schools meeting the following requirements:

- the schools were scheduled for a Botball workshop
- the students participating were at least 13 years old

Two types of workshops were offered: a Lego Mindstorms workshop, where programming was done with the Lego Mindstorms graphical programming environment; a Botball workshop, where the students programmed the robots in C. The schools could choose which workshop they wanted within the limits of availability of the robot kits and trainers. Since my concept is based on Botball, only Botball workshops were relevant for the case study.

In Austria ISCED level 2 starts after 4 years of ISCED level 1, meaning students we had participating in all STEMoFuture workshops were from 10 to 15 years old. My concept focuses

¹according to International Standard Classification of Education (ISCED) 2011 <http://www.uis.unesco.org/Library/Pages/DocumentMorePage.aspx?docIdValue=702&docIdFld=ID&SPSLanguage=EN>

²a non-profit association with the aim to promote scientific and technical excellence in schools using robotics; <http://pria.at>

on students age 13 and older, so only workshop groups with students who were at least 13 years old could be used for the case study.

Only two schools remained who met the basic criteria mentioned above, so these two workshop groups were chosen from all schools participating in the STEMofFuture project. The group I will refer to as group A was from a school only teaching ISCED level 2 with no classes in further education levels. Several students were from families who immigrated from countries where the main language was not German and this is one of the reasons students from that type of school usually do not pursue making the general qualification exam for university entrance [1].

The second group of students (group B) attended a school offering classes for ISCED level 2 and ISCED level 3 including the general qualification exam for university entrance at the end. Most students in this school came from families with German as first language and considered attending school at ISCED level 3 and making their general qualification exam for university entrance.

Study questions

There were two questions I wanted to evaluate:

1. Do the students maintain their initial motivation level?
2. Do the students perform better in exams than students of “traditional” introductory programming courses based on approaches using only computers and examples based on texts and numbers?

Therefore I developed the following evaluation strategy (cf. figure 6.1):

1. To analyze motivations two sources of evidence are be used:
 - Questions (for quantitative analysis): At three times during the workshop I ask students questions regarding their interest in the workshop itself to get an indicator of their motivation. Since I consider the term motivation too abstract for students at this age, I use the term interest instead of motivation. The second question I use as an indicator will be asked at the beginning and the end of the workshop and regards the interest for work in an technology or engineering fields.
 - Direct Observation (for qualitative analysis): I observe the behavior of the students while giving them instructions and during the times when they are required to solve tasks. These observations are noted down in protocols after the workshops
2. To analyze the level of knowledge I use a quantitative analysis. I ask exam-like questions to compare them with results obtained from the first exams during previous introductory computer science courses which used Java and a “traditional” approach with computers (not robots). To avoid having the students writing a full exam at the end I split the questions and issue one set of questions in the middle of the workshop and one set of questions at the end of the workshop. As with real exams the questions fall in two categories:

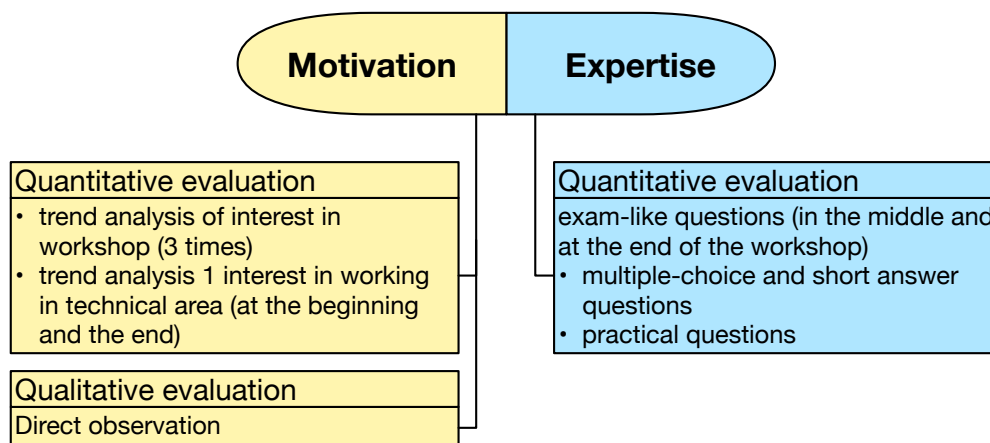


Figure 6.1: Evaluation strategy to cover motivation and expertise of the students

- Multiple choice and short answer questions: to show basic understanding and knowledges
- Practical tasks: here the students need to draft a possible solution to a given task with real programming instructions on the paper. This shows if the student is able to solve a computing problem on his own.

Since the concept is designed for first year introductory programming in technical high schools, thus for students who are most probably interested in programming, I also included a question regarding specific programming interest. The answer to this question is used as a filtering criteria. Only students who answer this question positively are included in the final evaluation regarding the knowledge gained. The resulting evaluation structure in terms of the quantitative analysis is shown in figure 6.2

The questions are issued three times during the workshop: one directly after the introduction of the 1st workshop block before anything is taught, one evaluation sheet after the 3rd workshop block for intermediate results and one evaluation sheet at the end of the 5th workshop block.

The questions are phrased as statements where the students can choose, if these statements are applicable for themselves and if they can identify with them. I use four levels for answers to avoid a neutral answer to encourage an explicit decision in one direction by the students (cf [22]). They need to choose between one of the following levels with + or – symbols most students are used to from the grading of homework (table 6.1)

The two statements regarding interest in the workshop and interest in working the field of technology are shown in table 6.2 together with the statement regarding interest in programming which was used only at the beginning of the workshop.

To evaluate the knowledge and the level of expertise gained during the workshop I used several multiple choice and short answer questions covering different topics taught. Since the students are doing these exams voluntarily, I try to limit the question to mostly one maximum two per topic, even if a complete coverage of the knowledge taught cannot be achieved this way.

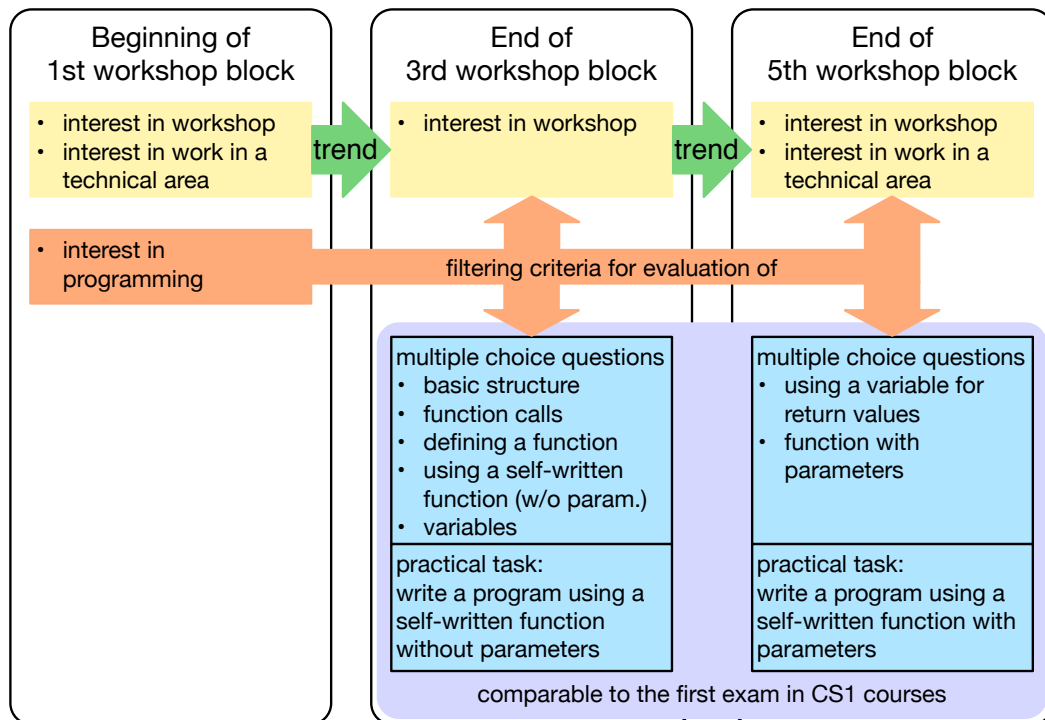


Figure 6.2: The structure of the quantitative part of the evaluation

Symbol	Meaning
--	not at all applicable for yourself
-	rather not applicable for yourself
+	for a large part applicable for yourself
++	totally applicable for yourself

Table 6.1: Symbols used for evaluating interest statements

I also avoided asking questions whose topics are taught just on the same day without having the possibility to consolidate.

The questions are shown in table 6.3 (The listing used for the last question is shown below).

```

1 void drive_straight(int time);
2
3 int main()
4 {
5     drive_straight(3000);
6     drive_straight(4000);
7     ao();
8     return 0;

```

I am interested in attending the workshop
When I finished my education I'd like to work in the field of technology
I am interested in programming (how I can issue commands, making the device do what I want)

Table 6.2: Statements used for assessing interest levels

topic / concept	question
basic program structure	Is the following statement true: the block of the main function is closed with a ;
function calling	How is the call of a function which uses configuration parameters structured?
function definition	Is the following statement true: The prototype of a function is used to introduce a self written function to the program. It is written before the main-function is written.
calling of a self-written function	Is the following statement true: A user defined function can be used several times with in the main function.
variables	Is the following statement true: A variable can never be empty, after she is first initialized.
	What does the datatype of a variable determine (name, size, type of value, nothing - it is only used for creating the variable, ...)?
using variables for return values	Write the instructions for following task: Ask the analog sensor with number 3 for his actual value and store the result of this function in a variable.
functions with parameters	Evaluate the following program and determine how long the robot drives (from the beginning of the movement to the time it stands still) or if the program is erroneous (listing 6.1)

Table 6.3: Questions used in multiple-choice and short answer parts of the exams

```

9 }
10
11 void drive_straight(int time)
12 {
13     mav(1, 700);
14     mav(2, 700);
15     msleep(time);
16 }

```

Listing 6.1: Program used for evaluation of understanding of functions with parameters

For the practical questions I chose tasks which required the defining of a function. The first task requires a function without parameter taught at the previous workshop day, and the second task required a function with parameters. This topic was covered the two workshop days before the final question was issued.

- Write a program where the robot will drive a turn two times for 3 seconds each time. Define a new command “turn” and use this command in your main function to accomplish the task.
- Define a new command “pirouette” where the robot rotates. The time of the rotation should be configurable with an argument of the function pirouette. Write the new function and a program where the new command is used for spinning the robot for 3 seconds.

Realization

Participation

Both workshops took place during school time and the students were supposed to attend. However especially in group B there was a high fluctuation in attendance level of the students resulting in few students being present when the questions were issued. The students in group A were present in a more regular way but still many students only filled out two of the three evaluation sheets resulting in incomplete data. Since all students who were present at the moment the evaluation sheet was issued returned their surveys the number of evaluation sheets returned is a good indicator for the attendance level. So less than half of the students who were initially present filled out all 3 evaluation sheets (table 6.4).

student groups	total number of students	students with 3 evaluation sheets
group A	23	13
group B	22	8
total	45	21

Table 6.4: Attendance at the workshop

In group B there were also two evaluation sheets from the third evaluation date where I was not able to match them to an intermediate or initial evaluation sheet because a student wrote two identification codes on the evaluation sheet and the other used a pseudonym or was not present at any of the other evaluation times. These evaluation sheets were not included in the study.

Regarding the attendance of the students the mentoring teachers told me after the workshop blocks, that the workshop time was after school. The teachers swapped another lesson time for this workshop which therefore took place on another time as the regular school lessons. But some students had already other appointments (sports or private lessons) which they were allowed to attend instead.

According to the premise that the workshop is targeted at students who show a basic interest in computer science, only those were chosen for evaluating the knowledge gained. So from all

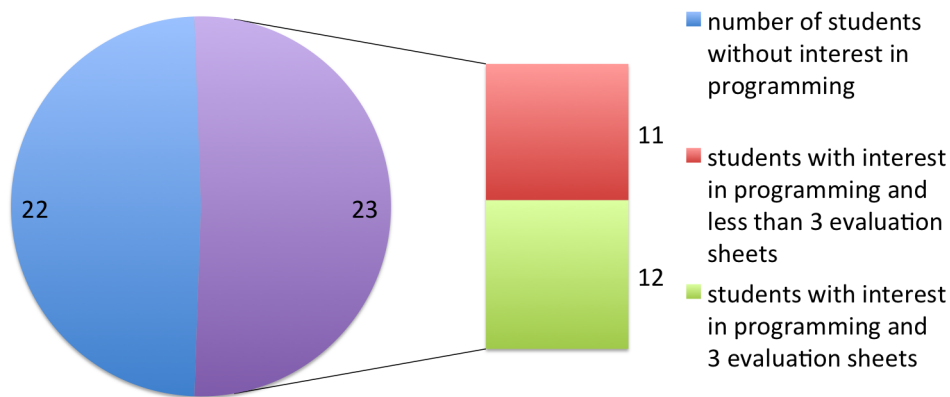


Figure 6.3: Number of students used in evaluating the expertise (students with interest in programming and 3 evaluation sheets)

attendees at the workshops only the evaluation sheets with a rating of + or ++ in the interest field of computer science were chosen. Since not all of the students filled out all three evaluation sheets I had to further limit the results, so that only students who took part in all 3 evaluations provide the basis of all further evaluations (figure 6.3). Thus, 12 students were used for the evaluation regarding the knowledge.

So the available data base for the evaluation was severely limited. Only 27% of the students had filled out all 3 evaluation sheets and had interest in programming according to the first evaluation sheet, and it is the data of these students which can be used for the quantitative evaluation of the expertise.

To enhance the informative value of the evaluation I supplemented the results gained from the exam-like questions with a quantitative analysis of the results from the programs the students with programming interest did for the challenges and competitions. Additionally I did a qualitative evaluation of all of the programs the students created during the workshop. So the resulting evaluation structure is shown in figure (6.4)

6.2 Results

Results regarding the motivational level

Quantitative results from the questions issued

To evaluate the results from the questions regarding interest, I counted how applicable to themselves the students rated the specific statements.

I asked the students about their motivation and interest in the workshop at three times during the workshop. The numbers of each answer category the students gave are shown in figure 6.5.

As seen in the figure the interest level in the workshop was generally very high. Furthermore the level of interest was preserved during the workshop with minor variations of one or two

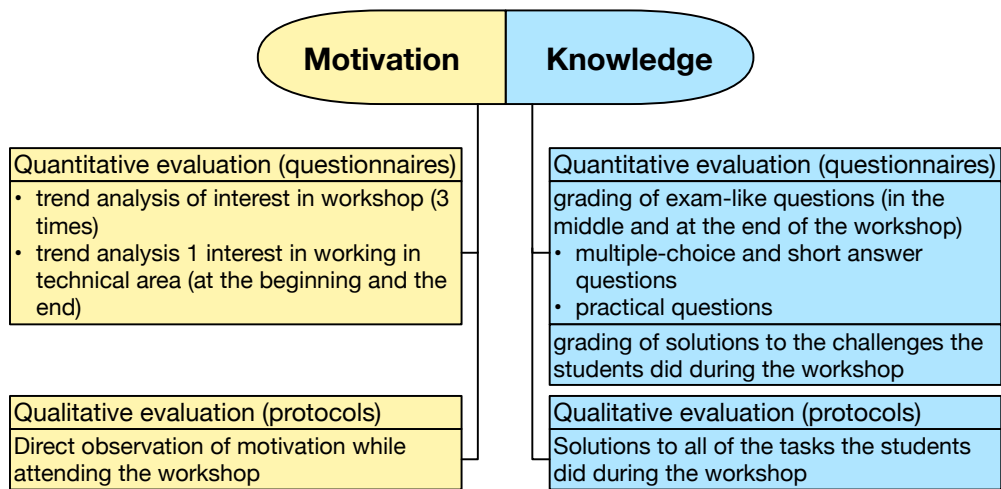


Figure 6.4: Modified evaluation strategy to enhance significance

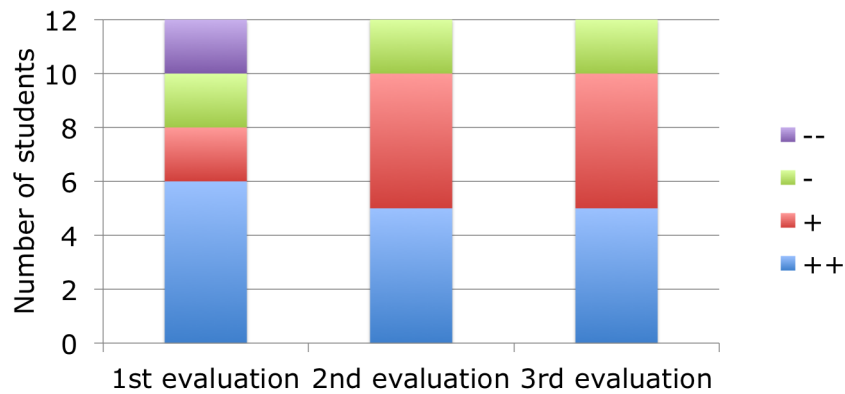


Figure 6.5: Student interest during the workshop

students who switched their interest by one level which can also be attributed to the participants general mood.

As a second indicator for their motivation level I asked the students if they would consider working in the field of technology after their education. This question was asked at the beginning of the workshop and at the end of the workshop. When analyzing the answers of the students, a trend towards an increase in interest in pursuing a technical career can be seen (figure 6.6):

Qualitative results from the observation

At the beginning of the workshop most students of group A were highly motivated to attend the workshop, even if technology and computer science was not their main field of interest. In contrary a larger group of students of group B were more or less openly opposing the necessity to attend the workshop. Due to organizational restrictions the workshop for group B was scheduled

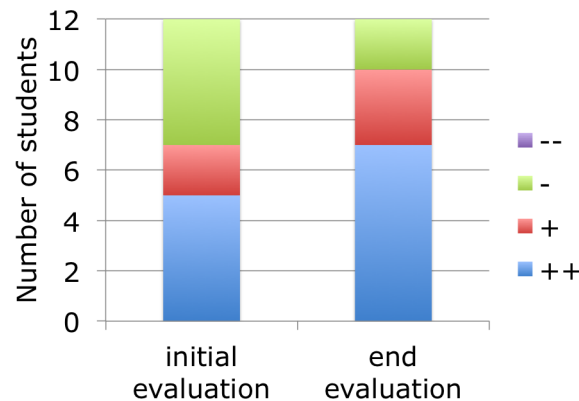


Figure 6.6: Interest in pursuing a technical career at the begin and the end of the workshop

from 2 p.m. to 5 p.m. once a week and therefore it did not fit into the regular timetable of these students.

The extensive explanations necessary for getting a robot to do something seemed to intimidate several students and they were a bit overwhelmed of the flood of new terms and concepts. When the robot first started to show a text which was programmed before, interest and motivation started to rise, but some students still were not impressed.

With the start of the first driving exercise nearly all students became engaged. Even more they started to try out different movements, questioned the fact, that the robot could not move in a straight line, and discussed possible solutions with me, their teachers or their teammates. Some students who had a soccer ball with them started to use it and teach their robot dribbling. This is a good example for the potential constructionism has to increase the motivation of students because it enables the students to connect the lessons to their own interests. Others started asking, how they can let the robot drive backwards, a lesson planned for the next workshop unit.

The second workshop unit started with repetition of the driving lessons from the unit before with the addition of how they can make the robot turn or go backwards. I introduced the different team roles also on the second workshop day which was received with interest. The switching of the roles took some additional efforts of persuasion because some students wanted to stay in control of the programming part. But after the role switch was established it allowed students, who were not very active during the last workshop day, to participate more intensively at the tasks.

Driving was still fun for nearly all students, but as soon as the workshop turned to the explanation of how to write functions on your own, some students lost interest again. In group B students tended to stay focused during the explanations and asked questions regarding the understanding of the subject at hand. In group A most students just wrote down what to do without issuing questions on their own. Both groups managed the guided task with writing a function for turning the robot for 90 degrees and completed the task of driving a square.

The individual challenges were introduced to both groups, but the students of group A ran out of time so the actual implementation of the challenges was done on day 3. In group B the

individual challenges were received with interest and most students started to work immediately on a driving challenge. Only the girl group wanted to do the dance challenge. But two student groups did not show interest in the challenges at first, but as the deadline drew nearer they started working feverishly because they did not want to be the ones without something to show.

On the third workshop day some students were discouraged by the fact that the robots tend to turn slightly left or right although both of their motors were given the same velocity while driving. Some of these students managed to see the newly explained parameters as a possible solution to their difficulties, with the ability to tweak the used parameters according to their currently used robot (mostly of group B). The group roles and the switch after each task was established, so no extra persuasion was needed on this part. For the challenges and competitions I let the students choose their own role. This led to blurred boundaries between the different roles. Instead of one student taking responsibility for the design, the design was discussed among two or even all three team members. But all students of a team were already used to contribute to the solution, so even students who were not very interested in the workshop, accepted tasks like testing the program written by their teammates.

Since the students of group B were able to complete their individual challenges on day 2, the ball collection competition was introduced to these students. They began the discussion of the necessities of the new competition after the explanations immediately. Time ran short so the actual competition was postponed to day 4.

The students of group A on the other hand were not very excited about the new possibilities and they needed more time to do the warm-up tasks with the new technology (like the day before). But when they got the possibility to work on the individual challenges their motivation level rose again. Most groups tried the dance challenge and with the addition that they can choose a suitable piece of music from their mobiles nearly everybody got involved and started to work with the robots. The show at the end of the workshop was accompanied with enthusiastic comments from the students.

Workshop day 4 went not as planned, because both groups had their time halved due to external circumstances. So group B continued with their ball moving competition. Due to the shortened time the introduction of a new topic after the competition was postponed and the students were given more time to polish their programs before the competition started. All groups used their time well and started to work on their robots. The competition itself was a great fun and the group which managed to get all balls but one off the map was the winner. During the competition it was remarkable that all students cheered for the other teams and were sympathetic when the robot missed a ball.

As group A just did their individual challenges at the last workshop day, the students were introduced to the new ball moving competition. As the days before the opportunity to work individually raised the spirits and motivational level. Especially the chance to work on a mechanical part (the “bulldozer blade”) appealed to some students who were not that much interested in programming.

In this group around three to four student teams had great difficulties with the task of the ball moving competition and did not know where to start. So I presented a light version of the ball moving competition as a warm-up task and all teams chose to work on this task as a first step. Having a task which seemed to be solvable by the students raised the spirits immediately.

On workshop day 5 the students of group A were intensively working on their programs for the ball moving competition. There were some students who did not want to involve themselves at first but with the splitting of the roles established everybody had something to do. And soon nearly all students were actively working on the problems at hand. The competition in this group had more the spirit of a show case like with the individual challenges, because not all student groups managed to get a working solution for the problem. But all student teams had a robot which moved some of the balls from the field. And this was enough, so no actual ranking was done in this group although some students did some bragging about their robot and how it managed to push the most balls away.

Group B was introduced to using sensors at the beginning of workshop day 5. The usage of the scene from Shrek 2 where the donkey always asked “Are we there yet?” was well remembered and got the interest even from the students who were not interested in explanations. Then the students had the opportunity to program some simple tasks on their own which they did with a professional students attitude – “we have to do it, so we are doing it”. Like on the other workshop days participation and motivation rose at the final challenge where I laid out a course with black dots they needed to follow. And it was exciting for the teams to look how far the robot will follow the course and several teams managed to follow the course through to the end.

So in a summary it was very often observable that motivation levels dropped during explanations of the theoretical background. But most students were highly motivated when there were challenges and competitions where they could work on their own. This applied to both groups even when group A needed more time to understand the basic principles and concepts.

Results regarding the expertise gained

Results from the exams

The exam-like questions were planned similar to the first exam held in computer science courses in the years 2011 and 2012 with half of the examination points awarded given to the examples in step with actual practice and the other half attained from the multiple choice and short answer questions.

But the participation in answering the exam questions could not be enforced, some students simply did not answer the longer questions in step with actual practice because they did not feel like doing them or they considered doing them too strenuous according to their remarks. This was especially true in group B. So at 25% of the practical questions of the evaluation group no answering attempt could be seen on the evaluation sheet. Additionally the answering of the questions in step with actual practice is the part where the most studying at home is needed according to student remarks during introductory programming courses at the technical high school. The students attending the workshop were not supposed to study anything at home.

Because of the reasons presented above I did a grading without taking the answers for the practical questions into account. So the grading from the exam-like questions is based only on the multiple-choice and short-answer questions.

To get additionally results for the students performance in programming practical challenges I graded the programs the students created for the first challenge round and the competition.

I chose these two programs, because no solution was presented for these programs and the students were encouraged to work and find the solution on their own. Help was only given, if the students had a syntax error they could not resolve on their own. To be in line with the target group of the concept I only used the programs created by students who were interested in programming although other students participated in the teams as well.

To get a base line to compare the results to I put a third column into the evaluation table with results achieved at the first computer science exam during my teaching at classes at the technical highschool TGM³ in the years 2011 and 2012 (65 students).

So in summary I did a qualitative evaluation of three factors for comparison:

1. exam-like questions: the questions were comparable to actual held programming exams and were graded according to school system grades (1 – 5). Only students with interest in programming and all questions answered were taken into account (12 students).
2. solutions to programming challenges: the programs were graded according to school system grades (1 – 5) and students with interest in programming and all challenges done were taken for this evaluation (23 students)
3. baseline for comparing the results: Grades from the first exams in introductory programming courses held in the years 2011 and 2012 (65 students)

	exam-like questions	programming challenges	school courses
grade 1	33%	26%	11%
grade 2	17%	35%	29%
grade 3	33%	22%	28%
grade 4	17%	13%	12%
grade 5	0%	4%	20%

Table 6.5: Grading results

In Austria only grade 5 is considered failed. The other grades (1 – 4) are all considered passed with grade 1 as best grade. So no student failed the exam according to real exam rules and many students were above average. Only one student failed to deliver solutions who would achieve a passing grade and again the majority students got one of the best two grades.

With the small group of students the results are not statistically significant. Therefore the results from other sources need to be taken into account.

Qualitative results from the practical tasks and challenges

Many students used the technologies introduced in their challenges and tasks. For example nearly all students in group B used the functions in the individual challenges issued after day 2. But two groups of students needed special encouragement to accomplish their challenge with

³Vienna Institute of Technology, Wexstraße 19-23, A-1200 Vienna, Austria; <http://www.tgm.ac.at/>

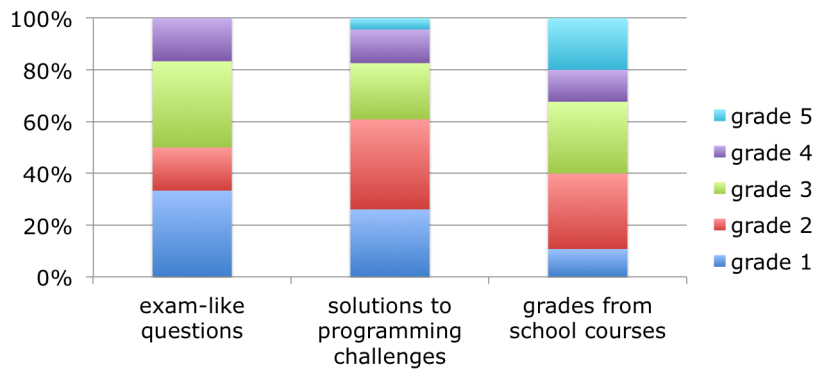


Figure 6.7: Overall grade results with different grading methods compared to exam results from 2011 and 2012

writing functions and not just issuing one command after the other in the main function. One of these two groups saw the benefit right before the showing of the challenges, because they needed to switch their robot with another one (because of low energy) and therefore they had to tweak their motor commands to the new mechanics of the robot. With their functions in place it was much easier done than without functions. On the other hand one group of students in group B had a completely different approach, because they started working on a kind of function library for the robot for different forms of movement (going forward, going backwards, turning left, turning right ...) instead of picking a specific challenge and working on it. They later just added them together for a dance performance of their robot.

Since group A needed more time, they tackled their challenges at the end of workshop day 3. So they had already the knowledge of how to write functions with arguments, but most of them stayed with the one function with arguments which was introduced on the slides (driving straight with time as argument). Most of the other movement options were programmed with functions without arguments.

In the follow-up challenge (ball moving race) functions with arguments were used widely as one of the win conditions was code efficiency and all students also thought about this part. In the end code efficiency was not needed to determine the winner, since there was a clear ranking according to the quantity of balls moved but it was very useful in bringing the students to think about it.

The usage of variables for storing return values of functions and while loops could not be assessed because these topics were only touched slightly. Therefore the students could not work on the challenges on their own and there was no possibility to observe their knowledge regarding these subjects.

6.3 Discussion

The case study in applying the curriculum in robotics workshops with students from showed convincing results in the area of motivational problems. The questions issued show that even if

the concept consisted of large theoretical parts the interest in the workshop did not fade. Even more the interest in pursuing a technical career was increased for the students who showed interest in programming. Both results indicate that the motivational effect of using robots in teaching programming is indeed countering the motivation loss when teaching the complicated correlations of writing a program.

These results are reinforced by the observations made. Although some of the students lost their interest during the explanation of the theoretical background, they could be caught with the practical challenges with the robots. These challenges and competitions were continuously the reason for the raise in the motivation level in both groups. In my opinion the opportunity to develop such challenges with relative ease is the main advantage of the usage of robotics in combination with introductory computer science. It is even easily possible to tweak the tasks and individualize the challenge levels to the needs of the students.

Another great advantage is the possibility for students to choose their own bottom-up or top-down approach, like the group who started with developing a movement library for the robot before they worked on an actual challenge. The curriculum favors working from bottom up but this example shows, that there are other approaches as well.

These facts lead to the conclusion that robots are well suited for countering motivational problems. Nevertheless the workshop time did not cover a whole semester and so the results show only a tendency which needs to be examined closer.

The case study showed that the outcome regarding knowledge gained and failure rates in exams could not be examined in the given evaluation setting. Because the students in the workshop had not the same incentives in doing the exam questions as students attending an introductory programming course at school where passing the exams is mandatory for continuing the education, the results are in fact not comparable.

When designing the exam questions I aimed to be comparable with actual exams held after the first few weeks in an introductory computer science course, but the questions in step with actual practice are the questions where most learning and preparation from students is needed especially for covering the syntax and reviewing the tasks and practical examples of the course so far. The students in the workshops did nothing of these and so it is not surprising that they had a lot of errors affecting the final grade. And some students, even of those who were interested in the topic itself, simply did not bother to fill out the practical exam questions, because it was too tedious for them to do an exam-like exercise or they were not in the mood for it.

So the data which can be derived from the exam-like questions can only give some hints. The normal grading of the exam questions shows a large quantity of grade 4 marks, more than usually obtained in exams covering the first half of a semester but only one student actually failed the exam completely. But grading the two parts of the exam (multiple choice and practical) separately shows, that the bad results are due to bad results in the practical part of the exam. In contrast the results of the grading of the multiple choice questions were really promising and there many students achieved the best mark and no student failed the exam.

The grading of the challenge and competition tasks indicate an increase in expertise gained as well. One has to bear in mind that in contrast to the exam questions the students worked in teams and they could use examples and documentation from the previous lessons and tasks. So better results than with exams could be expected. However one team consisting of two students

had very low results and did not use functions at the challenge and competition and achieved therefore only grade 5. Since one of the team members was not present at the intermediate evaluation time and the second one filled only the first evaluation sheet, this result could be explained with the low attendance level. But more than half of the students achieved the two best grades, a result showing good prospects for the knowledge gained.

Again the qualitative analysis emphasizes the results from the quantitative analysis. The usage of functions and arguments still posed difficulties at the beginning but with increasing familiarity functions and arguments were used in a very natural and intuitive way. Designing a program to solve certain tasks and dividing functionality into procedures are one of the most difficult subjects in a computer science course but the robots make the solutions to these problems more natural for the students. The students are more inclined to think of the concept which leads to the solution for themselves because they can imagine what the robot needs to do which is not the case if they need to think of a solution for a mathematical problem.

And breaking down the problem into procedures is more intuitive too as the student team which started to write a library shows. In my 10 years of experience with teaching computer science courses I never had a student attempting this approach with classical teaching concepts. But it is not clear if the knowledge gained is truly understood or if the students simply follow a kind of a recipe.

An additional example which was not in the scope of the original evaluation was encouraging too: a girl who stated that she is not interested in programming and pursuing a technical career at the first evaluation point, stated at the last evaluation point, that she really is interested in working in the field of technology. She also was rather good at the exams and would have achieved grade 2.

In summary I think the approach of teaching introductory programming with robots was promising because of the following points:

- Working on challenges with the robots was motivating for the students
- It was easy to individualize problems according to the capability of the students and to avoid too frustrating or too easy challenges
- The students could imagine the steps necessary to solve a challenge and thus working on the design of a program was natural for the students.
- The breaking down of problem into functions was more intuitive than with more classical tasks.
- The students exam results were promising without having the students learn at home for the exams.

But the results are not clear due to difficulties in the evaluation setup. The following problems were encountered:

- High fluctuation of student attendance.
- Taking the exam-like questions too lightly.

To counter these problems I propose the following changes for evaluating introductory programming workshops:

- Preparation of the accompanying teachers: The teachers need to understand the importance of the student attendance before the workshop starts. With this information they can prepare their students and take preventive measures. Maybe it is even possible to have the teachers take the results of the exam-questions into account for their grading, so the exam situation is more similar to the real exam environment.
- Additional incentives for students for taking the exams: The students can be given a certificate of attendance of the robotic course. For students who did well in the exams the certificate can state that they completed the robotic workshop with honor or something similar. Maybe a little giveaway can be arranged.

Conclusion

The problems of introductory programming are researched quite well but there are still several unanswered problems with concepts answering the needs of specific subdomains. The area of educational robotics is especially well suited to answer the motivational problems, which are affecting beginners in computer science. Additionally with robots the use of constructionism, still one of the most prominent didactic methods, is natural and as such robots are a suitable tool to enhance students' learning.

In this work I developed a concept combining several methods to enhance motivation and to allow students a better understanding of the key concepts in programming. This concept was tested in a case study together with school teachers who volunteered for attending workshops. The workshops covered a large part of the developed curriculum even though it was shortened due to time restrictions.

The results of the evaluation showed that the use of robotics especially with small challenges and competitions was very motivating for the students. An improvement of the exam-results could not really be observed because of the differences of the evaluation set-up and the exam setting in regular school classes. But the observations and the questions the student answered show a promising level of knowledge gained during the workshop and are as such auspicious.

Some unanswered questions remain:

- Does the motivational effect of the robots persists over a whole semester or even longer ?
- How do students fare when undertaking the exams under real course conditions?
- Is the knowledge gained also understood on a technical level?

The answering of these questions would be an ideal starting point for future research.

Additional material

A.1 Running a program on the controller

After a double click on the Botball Logo starts the KISS IDE for programming the robot you need to create a new project for your program (figure A.1). After choosing a meaningful name



Figure A.1: Selecting a new project to start

for identifying your project (figure A.2) you need to select a template and a name for your program in the project (figure A.3). The “Hello, World!” template provides a basic structure which is a good starting point. When clicking the OK button a very basic example program is loaded in the IDE (figure A.4). In the next step you need to connect the controller with the computer to compile the instructions for it. But first the controller must be powered on with the small switch on his right side (figure A.5). Like every computer the controller needs some time to start up. When it is finished a screen for selecting the different controller features is shown

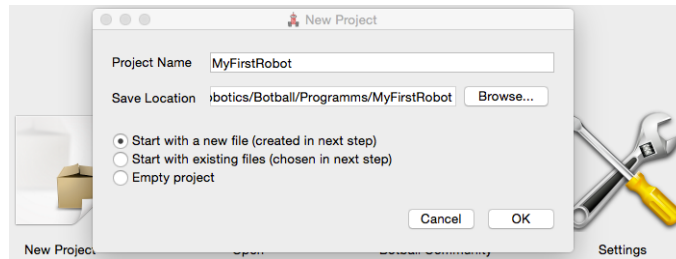


Figure A.2: Naming your project

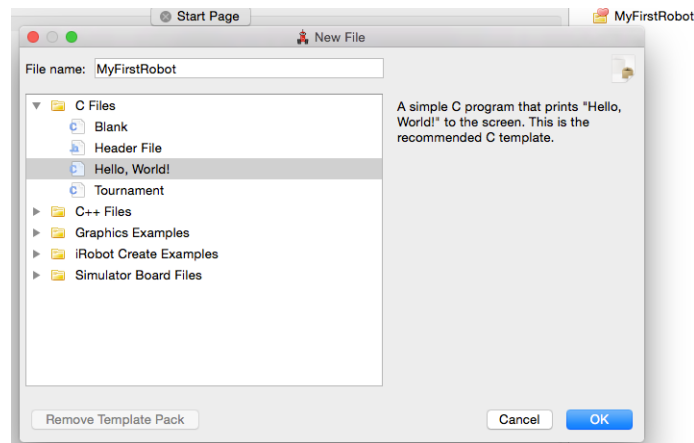


Figure A.3: Choosing a template and naming your program

(figure A.6). Then you need to connect the Micro USB port from the back side of the controller (figure A.7) with an USB port of the computer. If this is the first time the controller is connected with this computer it may take a bit before the computer recognizes the controller.

After the controller is connected with the computer you can start the translation process respectively the *compiling* of the program with a click on the “Compile” button in the icon bar of the IDE (figure A.8). When you first compile a new program you need to select a target environment for which the compilation should take place. Our program needs to be compiled for the controller connected via USB. So our target environment is listed with the computer equivalent of USB port which is a term like `/dev/tty.usbmodem...` (figure A.9).

When no errors are encountered a success message will appear briefly in the lower left corner of the screen. Now you can see your program on the controller in the list of programs (figure A.10). The name shown here is the name you have given at the beginning of the process.

With a touch you can select the program you want to run and press the “Run” button on the screen of the controller. If you have done everything correctly the example program shows the message “Hello, world!” is shown on the controller screen (figure A.11).

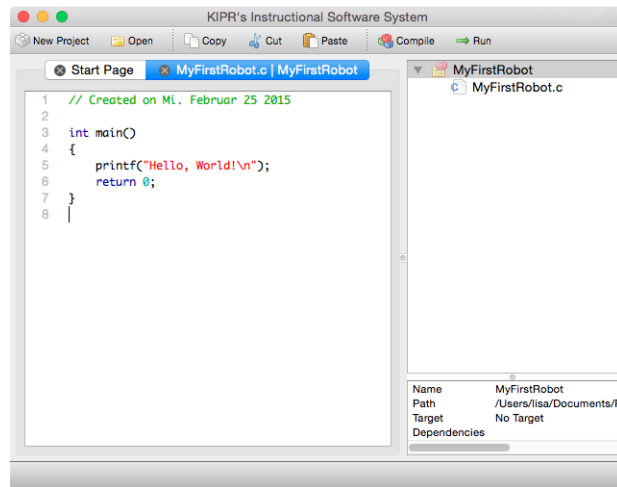


Figure A.4: The KISS IDE with the “Hello, World!” example program

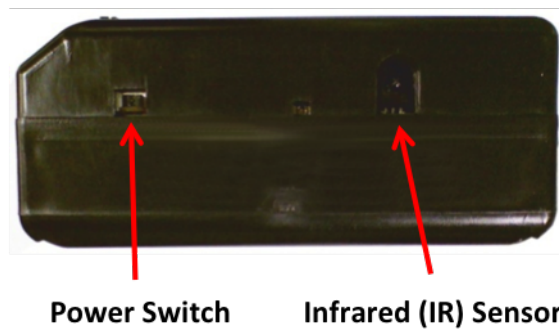


Figure A.5: The KIPR link controller from the right (picture courtesy of KIPR)

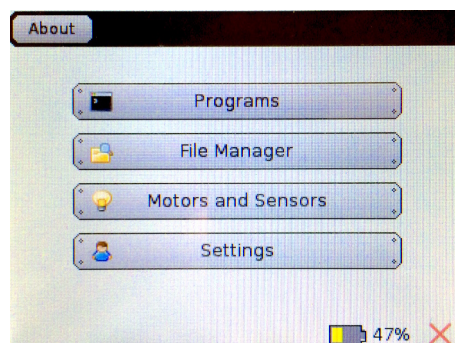


Figure A.6: Start screen of the KIPR link controller

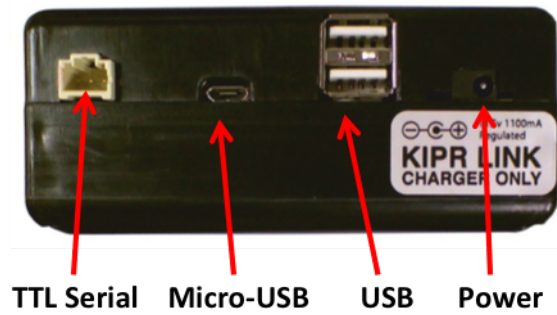


Figure A.7: The back side of the KIPR link controller (picture courtesy of KIPR)

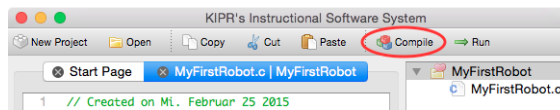


Figure A.8: Starting the compiling with the “Compile” button

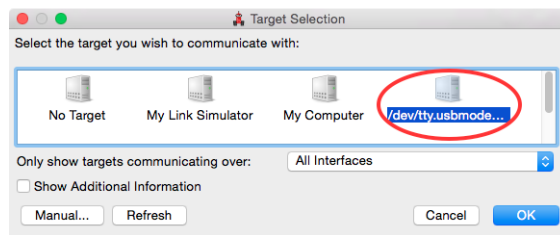


Figure A.9: Selecting the via USB connected controller

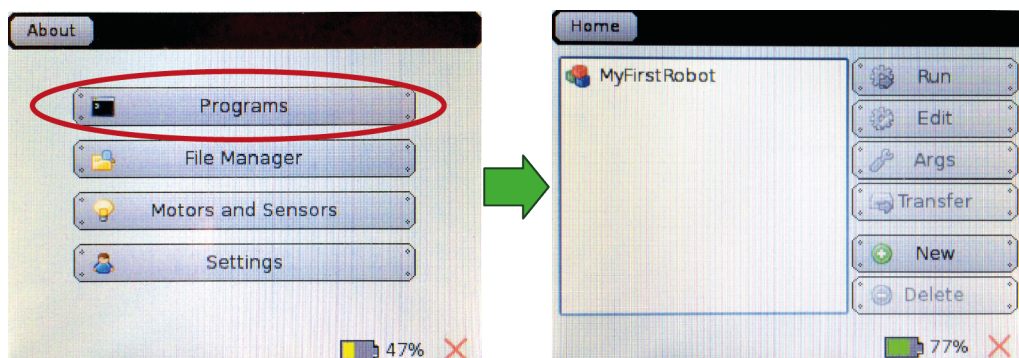


Figure A.10: Showing all available programs on the controller

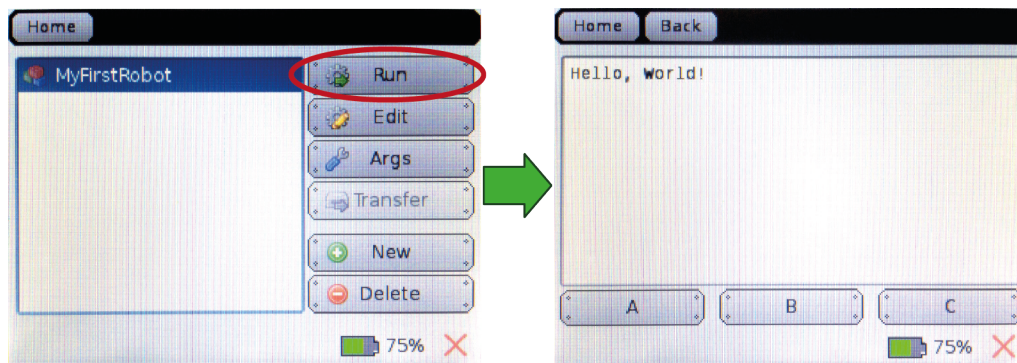


Figure A.11: Running the first example program

A.2 Sensor screen

On the sensor screen you can see the current values of all sensors. To reach it touch the sensor button at the home screen (figure A.12). The sensor screen is also used for setting the status of the pull-up resistor. Some sensors like the distance sensor need the pull-up status deactivated. To achieve this simply touch the “Toggle Pull-up” button on the screen when the sensor in question is selected, so that the corresponding up arrow vanishes (e.g pull-up deactivated at sensor 0 – figure A.13).

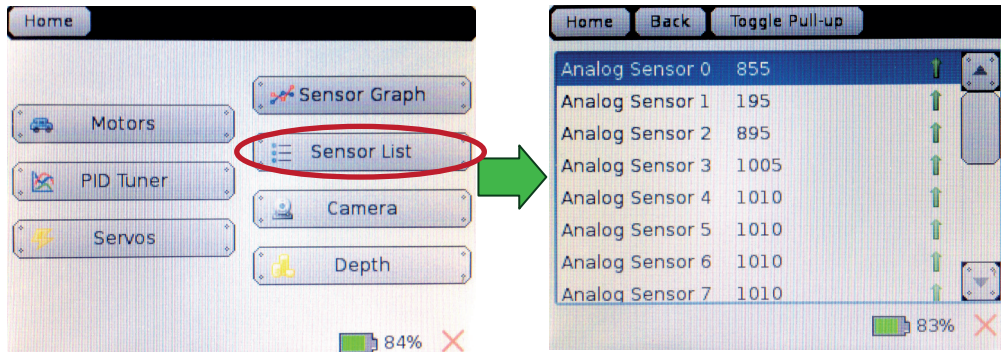


Figure A.12: Selecting the Sensor List

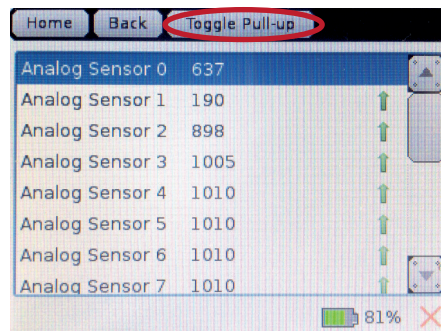


Figure A.13: Deactivating the Pull-up

APPENDIX **B**

workshop slides

The following slides were used in the evaluation workshops held for PRIA. As such they represent an adapted version of the presented curriculum. The workshops were held for Austrian students age 13+ and therefore the slides are written in german.

PRIA Practical Robotics Institute Austria

Den Roboter programmieren

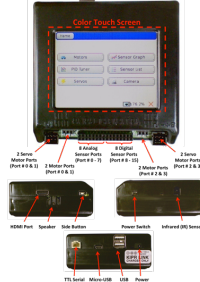
Umgang mit dem Botball-System



PRIA Practical Robotics Institute Austria

Was bedeutet programmieren?

- Programmieren = Befehle geben
- Wem geben wir Befehle?
 - Dem **Controller**
 - Entspricht dem Gehirn des Roboters

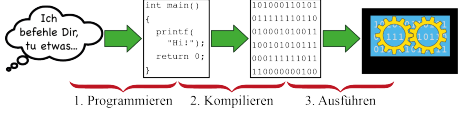


www.pria.at

PRIA Practical Robotics Institute Austria

Übersetzer = Compiler

- Übersetzungsprogramme = Compiler
- NICHT von Umgangssprache in Binärcode
- Nur von Programmiersprachen (C, C++, Java, php, Python) in Binärcode



1. Programmieren 2. Kompilieren 3. Ausführen

www.pria.at

PRIA Practical Robotics Institute Austria

Review

- Gab es Probleme
 - beim Programmieren?
 - beim Übersetzen?
 - beim Ausführen?

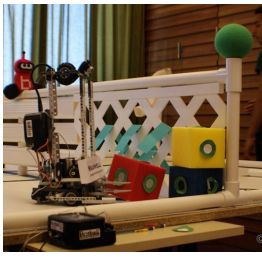
Aktivität

www.pria.at

PRIA Practical Robotics Institute Austria

Das Botball-System

- Spaß haben
- Lernen
- am Wettkampf teilnehmen
- Von KIPR




KISS INSTITUTE FOR PRACTICAL ROBOTICS

www.pria.at

PRIA Practical Robotics Institute Austria

Was bedeutet programmieren?

- Problem: der Controller versteht keine Umgangssprache
- Der Controller versteht nur **Binärcode** (bzw. Maschinensprache)



www.pria.at

PRIA Practical Robotics Institute Austria

Programmiere einen Menschen

- Teamarbeit: 1 Programmierer, 1 Compiler und 1 Roboter mit Controller
- Der Programmierer überlegt sich, was sein Roboter tun soll
- Dann schreibt er ein Programm für den Roboter
 - Die erlaubten Programmier-Befehle sind auf dem Arbeitsblatt aufgelistet (= engl. Befehle)
- Der Compiler übersetzt die engl. Programmier-Befehle in Robotersprache (= dt. Befehle)
- Der Roboter führt die deutschen Befehle aus


Aktivität

www.pria.at

PRIA Practical Robotics Institute Austria

Entwicklungsumgebung starten

- Zum Programmieren gibt es Entwicklungsumgebungen
 - engl. Integrated Development Environment = IDE
- Für das Botball-System gibt es die KISS-IDE
- Die IDE kann mit einem Doppelklick gestartet werden.



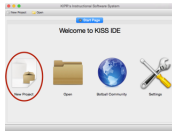
Mitmachen

www.pria.at

PRIA Practical Robotics Institute Austria Neues Projekt

- Programme werden in Projekten organisiert
- Für ein neues Programm muss zuerst ein neues Projekt angelegt werden

Mitmachen

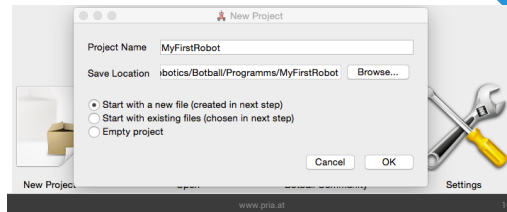


www.pria.at

PRIA Practical Robotics Institute Austria Projektname vergeben

- Für das Projekt muss ein sinnvoller Name vergeben werden

Mitmachen

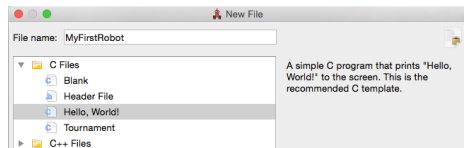


www.pria.at

PRIA Practical Robotics Institute Austria Programmvorlage wählen

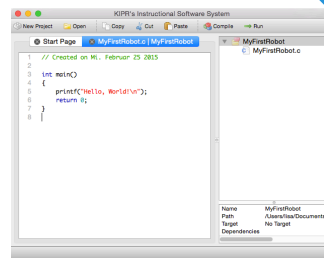
- Für das Programm muss eine Vorlage gewählt werden und ein weiterer Name vergeben werden
 - Das kann der gleiche wie vorhin sein

Mitmachen



www.pria.at

PRIA Practical Robotics Institute Austria Das erste Programm

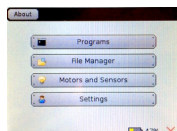


www.pria.at

PRIA Practical Robotics Institute Austria Den Roboter aktivieren

- Den Controller starten
 - kleiner schwarzer Schiebeschalter rechts
- Warten bis der Controller gestartet ist.

Mitmachen

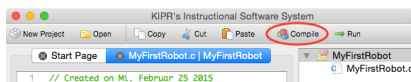
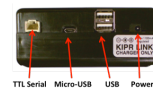


www.pria.at

PRIA Practical Robotics Institute Austria Das Programm kompilieren

- Verbinde den Controller mittels USB-Cabel mit dem Computer
 - Am Controller Micro-USB verwenden
- Starte das Kompilieren mit dem Compile-Button

Mitmachen

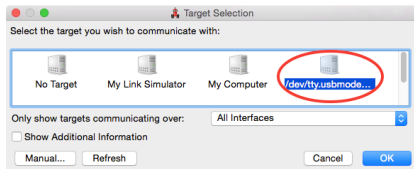


www.pria.at

PRIA Practical Robotics Institute Austria Ziel wählen

- Auswählen, für welche Maschine der Binärcode erzeugt wird.

Mitmachen

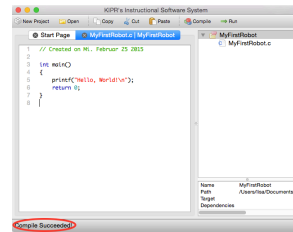


www.pria.at

PRIA Practical Robotics Institute Austria Erfolgreich?

- Übersetzungsvorgang erfolgreich?

Mitmachen

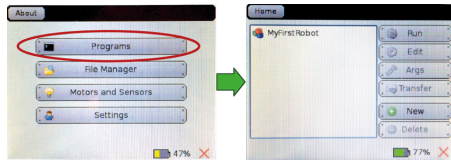


www.pria.at

Auf dem Roboter ausführen

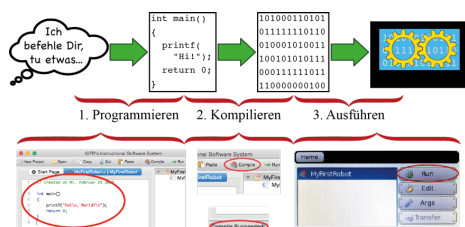
- Die Programme, die am Controller zum Ausführen bereit sind anzeigen

Mitmachen



www.pri.at 17

Zusammenfassung



www.pri.at 19

Das Beispiel-Programm analysieren

```

1 // Created on Mi. Februar 25 2015
2
3 int main()
4 {
5     printf("Hello, World!\n");
6     return 0;
7 }
    
```

1 → Kommentar
 3 → Hauptbereich des Programms – main-Funktion
 5 → Anweisung einen Text anzuzeigen
 6 → Anweisung das Programm zu beenden

www.pri.at 21

Das Beispiel-Programm analysieren

```

1 // Created on Mi. Februar 25 2015
2
3 int main()
4 {
5     printf("Hello, World!\n");
6     return 0;
7 }
    
```

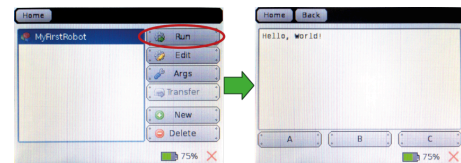
3 → Befehle = Anweisungen
 5 → Anweisung einen Text anzuzeigen
 6 → Anweisung das Programm zu beenden
 Alle Anweisungen werden mit einem ; abgeschlossen

www.pri.at 23

Auf dem Roboter ausführen

- Das richtige Programm aus der Liste wählen (antippen)
- Auf „Run“ tippen

Mitmachen

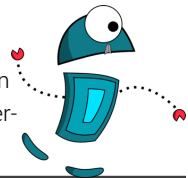


www.pri.at 18

Suchen & Entdecken

Roboter SuE-bot

Lernen zu Kommunizieren Mensch-Roboter-Interaktion



Das Beispiel-Programm analysieren

```

1 // Created on Mi. Februar 25 2015
2
3 int main()
4 {
5     printf("Hello, World!\n");
6     return 0;
7 }
    
```

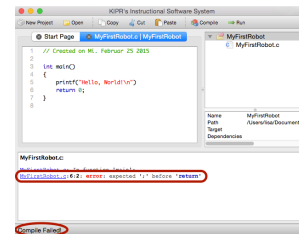
3 → Kopfzeile der main-Funktion
 4 → Körper-Block der main-Funktion

Die geschwungen Klammern {} bilden die Begrenzung des Blocks

www.pri.at 22

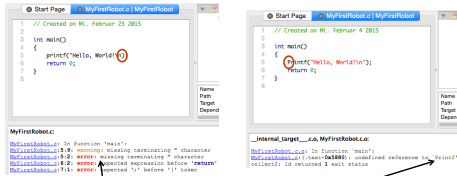
Fehler

- Es kann leicht passieren einen Fehler zu machen



www.pri.at 24

PRIA Practical Robotics Institute Austria Noch mehr Fehler



Bei mehreren Fehlern zunächst nur den ersten ausbessern

Groß- und Kleinschreibung sind wichtig!

www.pria.at

25

PRIA Practical Robotics Institute Austria Einen eigenen Gruß anzeigen

Aktivität

- Was hat das Beispiel-Programm bewirkt?
 - Wenn du nicht sicher bist, probiere es noch einmal aus.
- Was musst du ändern, damit der Roboter dich begrüßt?
 - Probiere es aus!
- Was musst du ändern, damit der Roboter jedes Team-Mitglied in einer eigenen Zeile begrüßt?
 - Ausprobieren!

www.pria.at

26

PRIA Practical Robotics Institute Austria Einen einzelnen Gruß anzeigen

Lösung

```
1 // Created on Mi. Februar 4 2015
2
3 int main()
4 {
5     printf("Guten Morgen, Lisa!\n");
6     return 0;
7 }
```

www.pria.at

27

PRIA Practical Robotics Institute Austria Mehrere Grüße anzeigen

Lösung

```
1 // Created on Mi. Februar 4 2015
2
3 int main()
4 {
5     printf("Guten Morgen, Lisa!\n");
6     printf("Hallo, Timon!\n");
7     printf("Servus, Andrej!\n");
8     return 0;
9 }
```

www.pria.at

28

PRIA Practical Robotics Institute Austria Review

Review

- Was könnte das `\n` am Ende bewirken?
- Der Computer führt einen Befehl nach dem anderen aus. Ist das auch erkennbar?

www.pria.at

29

PRIA Practical Robotics Institute Austria Funktionen, Funktionen, ...

Funktionen, Funktionen, ...

Jetzt kommt Bewegung ins Spiel



PRIA Practical Robotics Institute Austria Funktionen

- Funktion**
 - eine Fähigkeit, die der **Roboter** hat
 - kann über **Parameter** gesteuert werden

```
1 // Created on Mi. Februar 25 2015
2
3 int main()
4 {
5     printf("Hello, World!\n"); // Eine Funktion einen Text anzuzeigen
6     return 0;
7 }
```

Keine Funktion – keine Fähigkeit des Roboters, sondern etwas, das den Programm-Ablauf steuert

www.pria.at

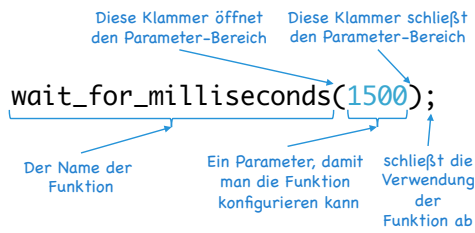
31

PRIA Practical Robotics Institute Austria Die printf-Funktion



www.pria.at

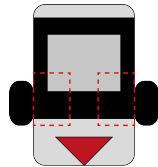
32



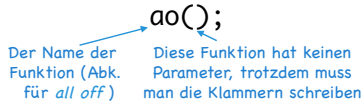
- ⚙ Was macht das Programm nun anders?
- ⚙ Was könnte daher die Aufgabe der Funktion `wait_for_milliseconds` sein?
- ⚙ Was passiert, wenn man die Parameter-Zahl verändert, z.B. auf 100 oder 5000?
 - ⚙ Ausprobieren!

Aktivität

- ⚙ Der Roboter hat 2 Motoren
 - ⚙ Ein Motor für das linke Rad
 - ⚙ Ein Motor für das rechte Rad



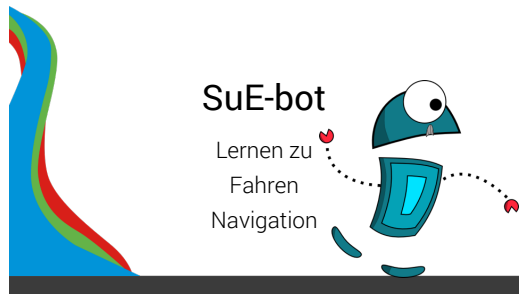
- ⚙ Funktion für das Ausschalten eines Motors:



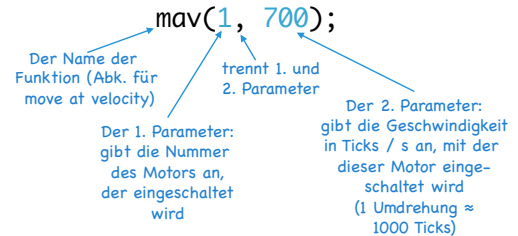
```

1 // Created on Mi. Februar 4 2015
2
3 int main()
4 {
5     printf("Guten Morgen, Lisa!\n");
6     wait_for_milliseconds(1500);
7     printf("Hallo, Timon!\n");
8     wait_for_milliseconds(1500);
9     printf("Grüß Dich, Andrej!\n");
10    return 0;
11 }
    
```

Mitmachen



- ⚙ Funktion für das Einschalten eines Motors:



- ⚙ Welche Schritte sind nun notwendig, um den Roboter für eine Sekunde vorwärts fahren zu lassen, und dann stehen zu bleiben.
- ⚙ Zur Erinnerung: es gibt folgende Funktionen:
 - ⚙ Wartepause (für eine gewisse Zeit)
 - ⚙ Einen Motor starten
 - ⚙ Alle Motoren ausschalten

Aktivität

- ✦ Wie viele Motoren hat der Roboter?
- ✦ Was passiert, wenn man nur einen Motor einschaltet?
 - ✦ Wer mag kann es ausprobieren, in dem er versucht mit nur einem Fuß zu gehen, und den anderen nicht zu bewegen
- ✦ Wie kann man 2 Motoren einschalten?
- ✦ Was passiert mit einem eingeschalteten Motor, wenn das Programm auf Wartepause geht?

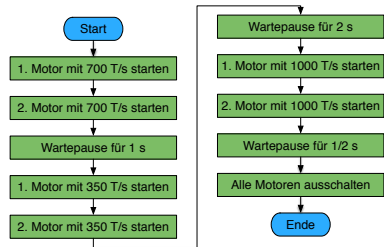
Aktivität

```
// Die erste Bewegung
// Created on Fr, März 6 2015
int main()
{
    // Ersten Motor starten
    // Zweiten Motor starten
    // Wartepause für 1s
    // Alle Motoren ausschalten
    // Programm beenden
    return 0;
}

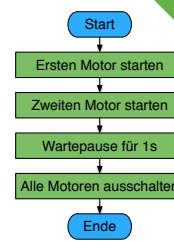
// Die erste Bewegung
// Created on Fr, März 6 2015
int main()
{
    // Ersten Motor starten
    mav(1, 700);
    // Zweiten Motor starten
    mav(2, 700);
    // Wartepause für 1s
    wait_for_milliseconds(1000);
    // Alle Motoren ausschalten
    aa();
    // Programm beenden
    return 0;
}
```

- ✦ Ändere nun dein Programm so um, dass der Roboter
 - ✦ zuerst 1 s eher schneller fährt
 - ✦ dann 2 s eher langsamer (nur halb so schnell)
 - ✦ und dann wieder ½ s ganz schnell

Aktivität



1. Beide Motoren starten
 1. Ersten Motor starten
 2. Zweiten Motor starten
2. Wartepause für 1s
3. Alle Motoren ausschalten



Lösung

- ✦ Achtung!
- ✦ Den Roboter zuerst abstecken
- ✦ Dann auf den Boden oder den Game-Table setzen
- ✦ Erst dann „Run“ antippen

Aktivität

1. Ersten Motor mit 700 T/s starten
2. Zweiten Motor mit 700 T/s starten
3. Wartepause für 1 s
4. Ersten Motor mit 350 T/s starten
5. Zweiten Motor mit 350 T/s starten
6. Wartepause für 2 s
7. Ersten Motor mit 1000 T/s starten
8. Zweiten Motor mit 1000 T/s starten
9. Wartepause für ½ s
10. Alle Motoren ausschalten

```
// 3 Geschwindigkeiten
// Created on Fr. März 6 2015
int main()
{
    // 1. Motor mit 700 T/s starten
    // 2. Motor mit 700 T/s starten
    // Wartepause für 1 s
    // 1. Motor mit 350 T/s starten
    // 2. Motor mit 350 T/s starten
    // Wartepause für 2 s
    // 1. Motor mit 1000 T/s starten
    // 2. Motor mit 1000 T/s starten
    // Wartepause für 0,5 s
    // Alle Motoren
    // Programm beenden
    return 0;
}
```

```
// 3 Geschwindigkeiten
// Created on Fr. März 6 2015

int main()
{
    // 1. Motor mit 700 T/s starten
    mav(1, 700);
    // 2. Motor mit 700 T/s starten
    mav(2, 700);
    // Wartezeit für 1 s
    wait_for_milliseconds(1000);
    // 1. Motor mit 350 T/s starten
    mav(1, 350);
    // 2. Motor mit 350 T/s starten
    mav(2, 350);
}


// Wartezeit für 2 s
wait_for_milliseconds(2000);
// 1. Motor mit 1000 T/s starten
mav(1, 1000);
// 2. Motor mit 1000 T/s starten
mav(2, 1000);
// Wartezeit für 0,5 s
wait_for_milliseconds(500);
// Alle Motoren ausschalten
ao();
// Programm beenden
return 0;
}
```

- ⚙ Das Verwenden von Funktionen funktioniert immer nach dem gleichen Aufbau:
Funktionsname(Parameter);
- ⚙ Funktionen ohne Parameter, z.B:
ao();
- ⚙ Funktionen mit einem Parameter, z.B:
wait_for_milliseconds(1000);
- ⚙ Funktionen mit mehreren Parametern, z.B:
mav(1, 700);

PRIA Practical Robotics Institute Austria

Noch mehr Bewegung

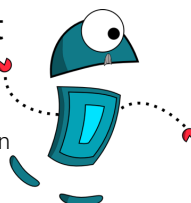
Mechanische Überlegungen



PRIA Practical Robotics Institute Austria

SuE-bot

Wer suchen will muss besser fahren



PRIA Practical Robotics Institute Austria

Hin und wieder zurück

Aktivität

- Lass den Roboter 3 s nach vor fahren und dann ohne umzudrehen wieder in seine ursprüngliche Position zurück kehren.

PRIA Practical Robotics Institute Austria

Umsetzung im Programm

```

// Hin und wieder zurück
// Created on Fr. März 6 2015
int main() {
  // 1. Motor mit 700 T/s starten
  // 2. Motor mit 700 T/s starten
  // Wartezeit für 3 s
  // 1. Motor mit -700 T/s starten
  // 2. Motor mit -700 T/s starten
  // Wartezeit für 3 s
  // Alle Motoren ausschalten
  // Programm beenden
  return 0;
}

// Hin und wieder zurück
// Created on Fr. März 6 2015
int main() {
  // 1. Motor mit 700 T/s starten
  mav(1, 700);
  // 2. Motor mit 700 T/s starten
  mav(2, 700);
  // Wartezeit für 3 s
  wait_for_milliseconds(3000);
  // 1. Motor mit -700 T/s starten
  mav(1, -700);
  // 2. Motor mit -700 T/s starten
  mav(2, -700);
  // Wartezeit für 3 s
  wait_for_milliseconds(3000);
  // Alle Motoren ausschalten
  aa();
  // Programm beenden
  return 0;
}

```

PRIA Practical Robotics Institute Austria

Wiederholung

- Mit Funktionen kann man Fähigkeiten des Roboters verwenden
- Starten eines Motors:
`mav(1, 700);`
- Das Programm für eine Anzahl an ms pausieren:
`wait_for_milliseconds(1000);`
- Alle Motoren ausschalten:
`aa();`

PRIA Practical Robotics Institute Austria

Rückwärts fahren

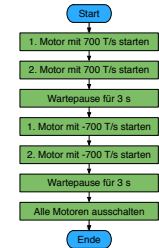
- Bis jetzt: positive Geschwindigkeit → vorwärts fahren
- Für andere Richtung → Vorzeichen umdrehen
- Das bedeutet: negative Geschwindigkeit → rückwärts fahren
- z.B.: `mav(1, -500);`



PRIA Practical Robotics Institute Austria

Entwurf

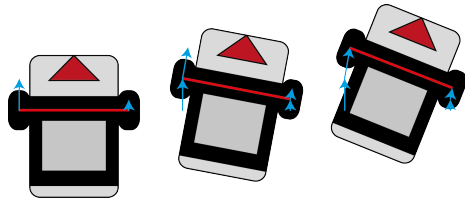
- Ersten Motor mit 700 T/s starten
- Zweiten Motor mit 700 T/s starten
- Wartezeit 3 s
- Ersten Motor mit -700 T/s starten
- Zweiten Motor mit -700 T/s starten
- Wartezeit 3 s
- Alle Motoren ausschalten



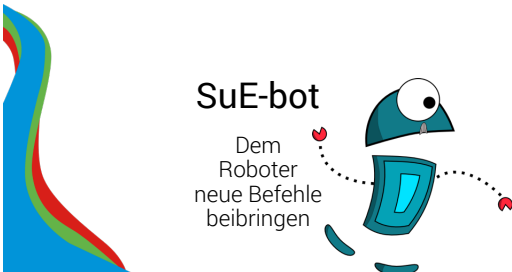
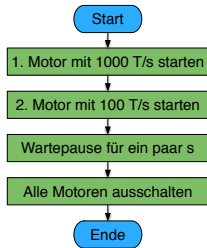
PRIA Practical Robotics Institute Austria

Bewegung genauer betrachtet

- Geradeaus fahren kann schwierig sein:
 - Die Motoren sind nicht genau gleich
 - Die Räder sind nicht genau gleich ausgerichtet
 - Ein Rad könnte fester angeschraubt sein, als das andere
- Was passiert, wenn ein Rad schneller fährt, als das andere?



1. Ersten Motor mit 1000 T/s starten
2. Zweiten Motor mit 100 T/s starten
3. Wartepause ? s (ausprobieren)
4. Alle Motoren ausschalten



- | | |
|--------------------------------------|---------------------------------------|
| 1. Beide Motoren mit 700 T/s starten | 11. Beide Motoren mit 700 T/s starten |
| 2. Wartepause für 3 s | 12. Wartepause für 3 s |
| 3. Ersten Motor für 700 T/s starten | 13. Ersten Motor für 700 T/s starten |
| 4. Zweiten Motor für 0 T/s starten | 14. Zweiten Motor für 0 T/s starten |
| 5. Wartepause für 2s | 15. Wartepause für 2s |
| 6. Beide Motoren mit 700 T/s starten | 16. Beide Motoren mit 700 T/s starten |
| 7. Wartepause für 3 s | 17. Wartepause für 3 s |
| 8. Ersten Motor für 700 T/s starten | 18. Ersten Motor für 700 T/s starten |
| 9. Zweiten Motor für 0 T/s starten | 19. Zweiten Motor für 0 T/s starten |
| 10. Wartepause für 2s | 20. Wartepause für 2s |
| | 21. Alle Motoren ausschalten |

- ⚙️ Erstelle ein neues Programm, das den Roboter eine Drehung vollführen lässt.
- ⚙️ Wenn der Roboter wieder an der gleichen Stelle angekommen ist, dann soll er stehen bleiben
 - ⚙️ Hinweis: derzeit kann der Roboter noch nicht erkennen, wann er stehen bleiben soll. D.h. ihr müsst einfach ausprobieren, wie lange er braucht.

Aktivität

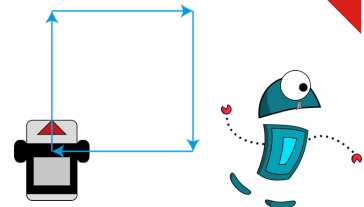
```

// Eine Umdrehung
// Created on Fr. März 6 2015
int main() {
  // 1. Motor mit 1000 T/s starten
  mov(1, 1000);
  // 2. Motor mit 100 T/s starten
  mov(2, 100);
  // Wartepause für ein paar s
  wait_for_milliseconds(6000);
  // Alle Motoren ausschalten
  do();
  // Programm beenden
  return 0;
}
    
```

Die genaue Zahl kann für jeden ein bisschen anders sein

- ⚙️ Versuche, den Roboter so zu programmieren, dass er ein Quadrat fährt.

Aktivität



```

// Führt ein Quadrat
// Created on So. März 7 2015
int main() {
  // Beide Motoren mit 700 T/s starten
  mov(1, 700);
  mov(2, 700);
  // Wartepause für 3s
  wait_for_milliseconds(3000);
  // 1. Motor mit 700 T/s starten
  mov(1, 700);
  // 2. Motor mit 0 T/s starten
  mov(2, 0);
  // Wartepause für 2s
  wait_for_milliseconds(2000);
  // Beide Motoren mit 700 T/s starten
  mov(1, 700);
  mov(2, 700);
  // Wartepause für 3s
  wait_for_milliseconds(3000);
  // 1. Motor mit 700 T/s starten
  mov(1, 700);
  // 2. Motor mit 0 T/s starten
  mov(2, 0);
  // Wartepause für 2s
  wait_for_milliseconds(2000);
  // Alle Motoren ausschalten
  do();
  // Programm beenden
  return 0;
}
    
```

PRIA Eigene Funktionen

- Man kann auch eigene Funktionen definieren
- Ähnlich zu Makros
- Vorgehensweise:
 - Nach** der main-Funktion: Funktionsdefinition = Funktion schreiben und mit Inhalt füllen
 - Vor** der main-Funktion: Prototypen schreiben = Funktion bekannt machen
 - Innerhalb** der main-Funktion: Funktionsaufruf = Funktion verwenden

PRIA Funktion gerade_fahren

- 2. Schritt: Prototyp definieren
- Vor** der main-Funktion
- Macht die Funktion im ganzen Programm bekannt
- Besteht nur aus der Kopfzeile + ;

```
// Created on So. März 7 2015
void gerade_fahren();

int main()
{
    return 0;
}

void gerade_fahren()
{
    // Motoren mit 700 T/s starten
    mav(1, 700);
    mav(2, 700);
    // Wartezeit für 3s
    wait_for_milliseconds(3000);
}
```

Kopieren und hinzufügen

PRIA Funktion gerade_fahren

- 3. Schritt: Funktionsaufruf
- Verwenden der Funktion **innerhalb** der main-Funktion
- Genauso, wie bei den anderen Funktionen

```
// Created on So. März 7 2015
void gerade_fahren();

int main()
{
    gerade_fahren();
    gerade_fahren();
    gerade_fahren();
    ao();
    return 0;
}
```

```
void gerade_fahren()
{
    // Motoren mit 700 T/s starten
    mav(1, 700);
    mav(2, 700);
    // Wartezeit für 3s
    wait_for_milliseconds(3000);
}
```

Mitmachen

PRIA Umsetzung im Programm

```
// Führt ein Quadrat
// Created on So. März 7 2015
void gerade_fahren();
void nach_rechts();

int main()
{
    gerade_fahren();
    nach_rechts();
    gerade_fahren();
    nach_rechts();
    gerade_fahren();
    nach_rechts();
    gerade_fahren();
    nach_rechts();
    // Alle Motoren ausschalten
    ao();
    // Programm beenden
    return 0;
}
```

```
void gerade_fahren()
{
    // Motoren m. 700 T/s starten
    mav(1, 700);
    mav(2, 700);
    // Wartezeit für 3s
    wait_for_milliseconds(3000);
}

void nach_rechts()
{
    // 1. Motor m. 700 T/s starten
    mav(1, 700);
    // 2. Motor m. 0 T/s starten
    mav(2, 0);
    // Wartezeit für 2s
    wait_for_milliseconds(2000);
}
```

PRIA Funktion gerade_fahren

- 1. Schritt: Funktion definieren
- Nach** der main-Funktion (ganz am Ende der Datei)
- Name vergeben und Inhalt schreiben

```
void gerade_fahren()
{
    // Beide Motoren mit 700 T/s starten
    mav(1, 700);
    mav(2, 700);
    // Wartezeit für 3s
    wait_for_milliseconds(3000);
}
```

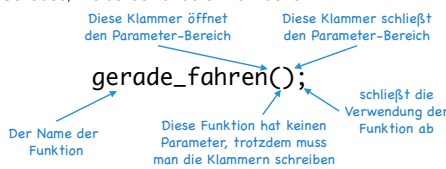
Mitmachen

Kopfzeile der eigenen Funktion
Körper-Block der eigenen Funktion

Die geschwungen Klammern {} bilden die Begrenzung des Blocks

PRIA Funktion gerade_fahren

- 3. Schritt: Funktionsaufruf
- Verwenden der Funktion **innerhalb** der main-Funktion
- Genauso, wie bei den anderen Funktionen



Mitmachen

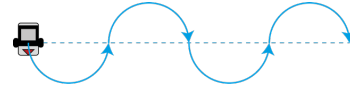
PRIA Funktion nach_rechts

- Versuche nach dem gleichen Prinzip eine Funktion mit dem Namen nach_rechts zu schreiben
- Verwende diese Funktion dann im Roboter-Programm für das Quadrat, so dass in der main-Funktion nur noch die Funktionen gerade_fahren, nach_rechts und ao verwendet werden.

Aktivität

PRIA Schlangenlinien

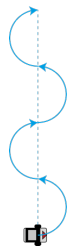
- Lass den Roboter eine Schlangenlinie fahren
- In der main-Funktion sollen bis auf ao() keine Motor- oder Warte-Funktionen vorkommen, sondern nur selbst geschriebene Funktionen
- Welche Funktionen wären sinnvoll?



Aktivität

PRIA Practical Robotics Institute Austria **Überlegungen**

- Prinzipieller Ablauf:
 - Bogen nach links
 - Bogen nach rechts
 - Bogen nach links
 - Bogen nach rechts
 - ... so oft man will
- Damit sind 2 Funktionen sinnvoll:
 - bogen_nach_links
 - bogen_nach_rechts



www.pria.at 26

PRIA Practical Robotics Institute Austria **Umsetzung im Programm**

```

// Führt eine Schlangenlinie
// Created on Sa, März 7, 2015
void bogen_nach_links();
void bogen_nach_rechts();

int main()
{
  bogen_nach_rechts();
  bogen_nach_links();
  bogen_nach_rechts();
  bogen_nach_links();
  bogen_nach_rechts();
  // Alle Motoren ausschalten
  aa();
  // Programm beenden
  return 0;
}

void bogen_nach_links()
{
  // 1. Motor m. 700 T/s starten
  mav(1, 700);
  // 2. Motor m. 300 T/s starten
  mav(2, 300);
  // Wartezeit für 3s
  wait_for_milliseconds(3000);
}

void bogen_nach_rechts()
{
  // 1. Motor m. 300 T/s starten
  mav(1, 300);
  // 2. Motor m. 700 T/s starten
  mav(2, 700);
  // Wartezeit für 3s
  wait_for_milliseconds(3000);
}

```

www.pria.at 28

PRIA Practical Robotics Institute Austria **Herausforderungen**

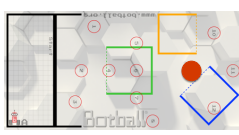
- Jedes Team sucht sich aus den Herausforderungen eine aus
- Diese Herausforderung versucht ihr dann möglichst selbständig zu meistern.
- Wenn ihr eine Herausforderung gemeistert habt, könnt ihr euch noch eine weitere Herausforderung aussuchen
- Die Herausforderungen sind unterschiedlich komplex. Sucht euch die aus, die euch gefällt.

Aktivität

www.pria.at 27

PRIA Practical Robotics Institute Austria **1 – Verschiebe das Hindernis**

- Fahre bis zum Hindernis vor und verschiebe es ein bisschen.
- Das Hindernis darf den Kreis nicht ganz verlassen
- Kehe anschließend wieder zur Startzone zurück



www.pria.at 29

PRIA Practical Robotics Institute Austria **2 – Einparken**

- Lass den Roboter auf der aufgelegten Matte in jede der 3 Parkzonen einparken (grün, blau, gelb).
- Wenn der Roboter in die blaue Parkzone einparkt, dann darf er keine blauen Linien überfahren, bei der grünen Parkzone keine grünen Linien und bei der gelben Parkzone keine gelben Linien

Herausforderung



www.pria.at 29

PRIA Practical Robotics Institute Austria **3 – Drumherum**

- Umfahre die aufgestellten Hindernisse außen ohne sie zu berühren und umzustoßen. Bleibe aber trotzdem mit zumindest einem Rad auf der Matte.
- Kehe anschließend wieder zur Startzone zurück

Herausforderung



www.pria.at 30

PRIA Practical Robotics Institute Austria **4 – Tanz auf der Stelle**

- Lass deinen Roboter nach einer selbst gewählten Choreografie tanzen
 - Z.B. links vor, rechts vor, beide vor uuuund zurück, ...
- Der Roboter soll am Schluss wieder in der Startposition stehen

Herausforderung

www.pria.at 31

PRIA Practical Robotics Institute Austria **5 – Tanz auf der Fläche**

- Lass deinen Roboter ähnlich wie beim Eistanzen mit einer selbst gewählten Choreografie auf einer Fläche tanzen
 - z.B. in dem er einen Kreis fährt und dabei sich dazwischen immer wieder umdreht, ein Stück rückwärts fährt und dann wieder eine Pirouette dreht und wieder vorwärts fährt
- Der Roboter soll am Schluss wieder ca. in der Startposition stehen

Herausforderung

www.pria.at 32

- ⚙ Wie ist es euch bei den Herausforderungen gegangen?
- ⚙ Wo habt ihr die größten Schwierigkeiten gehabt?

Aktivität

- ⚙ Überlege dir folgendes für deinen Traumroboter:
 - ⚙ Muss sich dein Traumroboter bewegen? Wenn ja welche Bewegungen müsste er machen können? Geradeaus fahren, umdrehen, eine Kurve, ...?
 - ⚙ Wenn nein, fällt dir vielleicht ein anderer Roboter ein, den du gerne hättest, der sich bewegen müsste. Welche Bewegungen müsste dieser Roboter machen können?
 - ⚙ Beschreibe einen Bewegungsablauf für einen deiner Traumroboter.

Hausübung

- ⚙ Negative Geschwindigkeit → rückwärts fahren
`mav(1, -700);`
`mav(2, -700);`
- ⚙ Ein Motor schneller als der andere → Kurve
`mav(1, 700);`
`mav(2, 0);`
- ⚙ Weitere Funktionen für das Fahren:
`wait_for_milliseconds(1000);`
`ao();`

- ⚙ Wie müsste Sue-Bot fahren, damit sie ein ganzes Zimmer absuchen kann?

Sue-Bot



PRIA Practical Robotics Institute Austria

Struktur in das Chaos

Besser Sprache verstehen

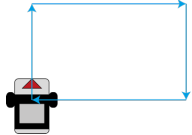


PRIA Practical Robotics Institute Austria

Rechteck

Review


- Was müsste am Quadrat-Programm geändert werden, damit ein Rechteck abgefahren wird?
- Wie viele Funktionen sind da sinnvoll?



PRIA Practical Robotics Institute Austria

Was sind Variablen


- Variablen sind Behälter, in denen Sachen gespeichert werden können, wie z.B. Marmeladengläser:
 - muss existieren, bevor man etwas hinein gibt
 - kann Inhalte aufnehmen und speichern (Deckel zu)
 - gibt es in verschiedenen Größen und Formen
 - erlauben es den Inhalt anzusehen und zu überprüfen



PRIA Practical Robotics Institute Austria

Erzeugen von Variablen

- Um eine Variable (= ein magisches Marmeladenglas) zu erzeugen braucht man
 - Einen Namen (selbst aussuchen, Sonderzeichen und Leerzeichen verboten!)
 - Die Größe (und damit die Art des Inhalts) z.B: `int` (ist die passende Größe für ganze Zahlen)
 - `int zeit` erzeugt eine Variable für ganze Zahlen mit dem Namen `zeit`



PRIA Practical Robotics Institute Austria

Wiederholung

- Negative Geschwindigkeit → rückwärts fahren
`mav(1, -700);`
`mav(2, -700);`
- Ein Motor schneller als der andere → Kurve
`mav(1, 700);`
`mav(2, 0);`
- Weitere Funktionen für das Fahren:
`wait_for_milliseconds(1000);`
`ao();`

PRIA Practical Robotics Institute Austria


Konfigurierbare Funktionen

- Funktionen können durch Parameter konfigurierbar gemacht werden
- Parameter werden zwischen die Klammern geschrieben.
- Damit man für eigene Funktionen Parameter einsetzen kann braucht man **Variablen**.

PRIA Practical Robotics Institute Austria

Magische Marmeladengläser

- Variablen sind allerdings nicht so wie normale Marmeladengläser: sie sind magisch
 - Sie können nur über ihren Namen gefunden werden
 - Inhalt kann nicht hinausgenommen werden
 - Wenn etwas neues hineingegeben wird, verschwindet das alte
 - Die Größe bestimmt die Art des Inhalts
 - Wenn das Programm endet, dann verschwinden alle Marmeladengläser (manche auch früher)



PRIA Practical Robotics Institute Austria

Funktion mit Parameter definieren


Mitmachen

- Eine Variable kann als Parameter beim Definieren (Schreiben) einer Funktion verwendet werden
- Dann muss die Variable innerhalb der Klammern () erzeugt werden

```
void gerade_fahren(int zeit)
{
  // Beide Motoren mit 700 T/s starten
  mav(1, 700);
  mav(2, 700);
  // Wartezeit für zeit ms
  wait_for_milliseconds(zeit);
}
```

Die Variable `zeit` wird erzeugt

Nachschauen, was in der Variablen drin ist und als Zeitdauer verwenden



PRIA Prototyp für Funktion mit Parameter

- Der Prototyp muss auch den Parameter enthalten
- Er muss immer gleich wie die Kopfzeile der Funktion sein +;

```
// Created on Sa. März 7 2015
void gerade_fahren(int zeit);

int main()
{
    return 0;
}

void gerade_fahren(int zeit);
{
    // Motoren mit 700 T/s starten
    mav(1, 700);
    mav(2, 700);
    // Warte pause für zeit ms
    wait_for_milliseconds(zeit);
}
```

Kopieren und
hinzu fügen

www.pria.at

9

PRIA Funktion mit Parameter aufrufen

- Die Variable (das Marmeladenglas) wird gefüllt, wenn die Funktion verwendet wird.
- D.h. man **muss** beim Verwenden der Funktion einen passenden Inhalt angeben
- Genauso wie bei anderen Funktionen mit Parameter

Diese Klammer öffnet den Parameter-Bereich

Diese Klammer schließt den Parameter-Bereich

gerade_fahren(3000);

Der Name der Funktion

Diese Funktion hat eine ganze Zahl als Parameter, die für die Zeitdauer steht

schließt die Verwendung der Funktion ab

www.pria.at

10

PRIA Funktion gerade_fahren

```
// Created on Sa. März 7 2015
void gerade_fahren(int zeit);

int main()
{
    gerade_fahren(3000);
    gerade_fahren(5000);
    gerade_fahren(3000);
    gerade_fahren(5000);
    ao();
    return 0;
}
```

```
void gerade_fahren(int zeit)
{
    // Motoren mit 700 T/s starten
    mav(1, 700);
    mav(2, 700);
    // Warte pause für zeit ms
    wait_for_milliseconds(zeit);
}
```

Mitmachen

www.pria.at

11

PRIA Parameterwerte genauer betrachtet

Der Parameterwert beim Aufruf wird in die Parametervariable gefüllt

```
int main()
{
    gerade_fahren(3000);
    gerade_fahren(5000);
    gerade_fahren(3000);
    gerade_fahren(5000);
    ao();
    return 0;
}
```

```
void gerade_fahren(int zeit)
{
    // Motoren mit 700 T/s starten
    mav(1, 700);
    mav(2, 700);
    // Warte pause für zeit ms
    wait_for_milliseconds(zeit);
}
```



Nachsehen, was in der Variablen drinnen steht und als Zeitdauer verwenden

www.pria.at

12

PRIA Parameterwerte genauer betrachtet

Der Parameterwert beim Aufruf wird in die Parametervariable gefüllt

```
int main()
{
    gerade_fahren(3000);
    gerade_fahren(5000);
    gerade_fahren(3000);
    gerade_fahren(5000);
    ao();
    return 0;
}
```

```
void gerade_fahren(int zeit)
{
    // Motoren mit 700 T/s starten
    mav(1, 700);
    mav(2, 700);
    // Warte pause für zeit ms
    wait_for_milliseconds(zeit);
}
```

Nachsehen, was in der Variablen drinnen steht und als Zeitdauer verwenden

www.pria.at

13

PRIA Parameterwerte genauer betrachtet

Der Parameterwert beim Aufruf wird in die Parametervariable gefüllt

```
int main()
{
    gerade_fahren(3000);
    gerade_fahren(5000);
    gerade_fahren(3000);
    gerade_fahren(5000);
    ao();
    return 0;
}
```



Nachsehen, was in der Variablen drinnen steht und als Zeitdauer verwenden

www.pria.at

14

PRIA Parameterwerte genauer betrachtet

Der Parameterwert beim Aufruf wird in die Parametervariable gefüllt

```
int main()
{
    gerade_fahren(3000);
    gerade_fahren(5000);
    gerade_fahren(3000);
    gerade_fahren(5000);
    ao();
    return 0;
}
```

```
void gerade_fahren(int zeit)
{
    // Motoren mit 700 T/s starten
    mav(1, 700);
    mav(2, 700);
    // Warte pause für zeit ms
    wait_for_milliseconds(zeit);
}
```

Nachsehen, was in der Variablen drinnen steht und als Zeitdauer verwenden

www.pria.at

15

PRIA Vollständiges Rechteck-Programm

```
// Fährt ein Quadrat
// Created on Sa. März 7 2015
void gerade_fahren(int zeit);
void nach_rechts();

int main()
{
    gerade_fahren(3000);
    nach_rechts();
    gerade_fahren(5000);
    nach_rechts();
    gerade_fahren(3000);
    nach_rechts();
    gerade_fahren(5000);
    nach_rechts();
    // Alle Motoren ausschalten
    ao();
    // Programm beenden
    return 0;
}
```

```
void gerade_fahren(int zeit)
{
    // Motoren m. 700 T/s starten
    mav(1, 700);
    mav(2, 700);
    // Warte pause für zeit ms
    wait_for_milliseconds(zeit);
}

void nach_rechts()
{
    // 1. Motor m. 700 T/s starten
    mav(1, 700);
    // 2. Motor m. 0 T/s starten
    mav(2, 0);
    // Warte pause für 2s
    wait_for_milliseconds(2000);
}
```

www.pria.at

16

PRIA Practical Robotics Institute Austria

Varianten?

Aktivität

- Gibt es noch ein andere Möglichkeit, eine konfigurierbare Funktion für unterschiedlich lange Strecken zu schreiben?
 - Hinweis: um einen längeren Weg als zuvor zu gehen, kann ich mir entweder mit gleicher Geschwindigkeit mehr Zeit lassen, oder ...

PRIA Practical Robotics Institute Austria

Anderer Parameter

```
// Fährt ein Quadrat
// Created on Sa. März 7 2015
void gerade_fahren(int geschwindigkeit);
void nach_rechts();

int main()
{
  gerade_fahren(500);
  nach_rechts();
  gerade_fahren(800);
  nach_rechts();
  gerade_fahren(500);
  nach_rechts();
  gerade_fahren(800);
  nach_rechts();
  // Alle Motoren ausschalten
  ao();
  // Programm beenden
  return 0;
}

void gerade_fahren(int geschwindigkeit)
{
  // Motoren m. geschwindigkeit starten
  mav(1, geschwindigkeit);
  mav(2, geschwindigkeit);
  // Wartezeit für 3 ms
  wait_for_milliseconds(3000);
}

void nach_rechts()
{
  // 1. Motor m. 700 T/s starten
  mav(1, 700);
  // 2. Motor m. 0 T/s starten
  mav(2, 0);
  // Wartezeit für 2s
  wait_for_milliseconds(2000);
}
```

PRIA Practical Robotics Institute Austria

Mehrere Parameter

- Eine Funktion kann auch mehrere Parameter haben
- Dafür müssen auch mehrere Variablen erzeugt werden
- Die Parameter werden durch einen Beistrich getrennt

```
void gerade_fahren(int geschwindigkeit, int zeit)
{
  // Beide Motoren mit geschwindigkeit T/s starten
  mav(1, geschwindigkeit);
  mav(2, geschwindigkeit);
  // Wartezeit für zeit ms
  wait_for_milliseconds(zeit);
}
```

PRIA Practical Robotics Institute Austria

Mehrere Parameterwerte

Die Reihenfolge bestimmt, welche Variable welchen Wert erhält

```
int main()
{
  gerade_fahren(500, 2000);
  gerade_fahren(800, 3000);
  gerade_fahren(500, 2000);
  gerade_fahren(800, 3000);
  ao();
  return 0;
}

void gerade_fahren(int geschwindigkeit, int zeit)
{
  // Motoren mit geschwindigkeit T/s starten
  mav(1, geschwindigkeit);
  mav(2, geschwindigkeit);
  // Wartezeit für zeit ms
  wait_for_milliseconds(zeit);
}
```

Nachschauen, was in der Variablen drinnen steht und verwenden

Mitmachen

PRIA Practical Robotics Institute Austria

Zwei Parameter

```
// Fährt ein Quadrat
// Created on Sa. März 7 2015
void gerade_fahren(int geschwindigkeit, int zeit);
void nach_rechts();

int main()
{
  gerade_fahren(500, 2000);
  nach_rechts();
  gerade_fahren(800, 3000);
  nach_rechts();
  gerade_fahren(500, 2000);
  nach_rechts();
  gerade_fahren(800, 3000);
  nach_rechts();
  // Alle Motoren ausschalten
  ao();
  // Programm beenden
  return 0;
}

void gerade_fahren(int geschwindigkeit, int zeit)
{
  // Motoren m. geschwindigkeit starten
  mav(1, geschwindigkeit);
  mav(2, geschwindigkeit);
  // Wartezeit für 3 ms
  wait_for_milliseconds(zeit);
}

void nach_rechts()
{
  // 1. Motor m. 700 T/s starten
  mav(1, 700);
  // 2. Motor m. 0 T/s starten
  mav(2, 0);
  // Wartezeit für 2s
  wait_for_milliseconds(2000);
}
```

PRIA Practical Robotics Institute Austria

Schlangenlinien

Aktivität

- Versuche dein Schlangenlinien-Programm so zu ändern, dass nur noch eine einzige Funktion bogen_fahren vorkommt, die je nach Parameter entweder nach links oder nach rechts fährt.
 - Hinweis: Schau nach in was sich die beiden Funktionen unterscheiden. Das sind dann gute Ausgangspunkte für Parameter

PRIA Practical Robotics Institute Austria

Umsetzung im Programm

```
// Fährt eine Schlangenlinie
// Created on Sa. März 7 2015
void bogen_fahren(int geschwindigkeit1, int geschwindigkeit2);

int main()
{
  bogen_fahren(700, 300);
  bogen_fahren(300, 700);
  bogen_fahren(700, 300);
  bogen_fahren(300, 700);
  bogen_fahren(700, 300);
  bogen_fahren(300, 700);
  bogen_fahren(700, 300);
  bogen_fahren(300, 700);
  // Alle Motoren ausschalten
  ao();
  // Programm beenden
  return 0;
}

void bogen_fahren(int geschwindigkeit1, int geschwindigkeit2)
{
  // 1. Motor m. geschwindigkeit1
  mav(1, geschwindigkeit1);
  // 2. Motor m. geschwindigkeit2
  mav(2, geschwindigkeit2);
  // Wartezeit für 3s
  wait_for_milliseconds(3000);
}
```

PRIA Practical Robotics Institute Austria

Herausforderungen mit Funktionen

Aktivität

- Ändere dein Herausforderungsprogramm so, dass es Funktionen verwendet. (Wenn du möchtest kannst du auch eine neue Herausforderung machen)
 - Besser als mein Code
 - Schneller und besser (kürzer) als ein anderes Team
 - Choreographie erweitern, so dass der Roboter Tanzbegriffe versteht, pirouette, ...

- ✿ Schreibe ein Programm, mit Hilfe von Funktionen, so dass SuE-Bot die ganze Fläche abfahren kann, um etwas zu suchen

Aktivität



- ✿ Variablen sind Gefäße zum Speichern von Inhalt
- ✿ Sie werden mit ihrer Größe (int) und dem Namen erzeugt
- ✿ Variablen können verwendet, um Funktionen konfigurierbar zu machen. Sie werden dann Parameter genannt
- ✿ Parameter werden zwischen die Klammern in den Parameterbereich einer Funktion geschrieben
- ✿ Eine Funktion kann mehrere Parameter haben
- ✿ Wenn ein oder mehrere Parameter bei einer Funktion definiert wurden, dann muss man sie auch bei der Verwendung angeben

- ✿ Mit Hilfe von 3 Schritten kann man Funktionen selber schreiben:
 - ✿ Funktion nach der main-Funktion definieren mit `void Funktionsname()`

```
{
    Inhalt
}
```
 - ✿ Prototypen vor die main-Funktion kopieren (; nicht vergessen)
 - ✿ Funktion innerhalb der main-Funktion verwenden (genauso wie bereits vorhandene Funktionen)

- ✿ Überlege dir folgendes für deinen Traumroboter:
 - ✿ Welche Funktionen könntest du für deinen Roboter schreiben?
 - ✿ Wären das nur Bewegungsfunktionen oder sind auch andere Funktionen sinnvoll?

Hausübung

PRIA Practical Robotics Institute Austria

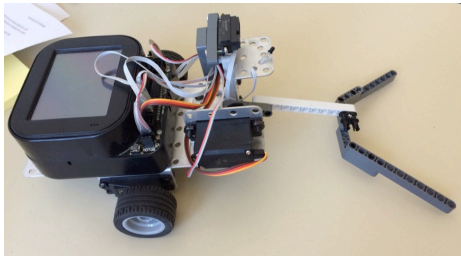
Arbeiten mit dem Arm

Bewegen des Servo-Motors



PRIA Practical Robotics Institute Austria

Eine Schaufel bauen



www.pria.at 3

PRIA Practical Robotics Institute Austria

Servo-Funktionen

- ⚙ Einstellen der Winkelposition (zw. 900 und 1360):

```
set_servo_position(1, 1000);
```

Der Name der Funktion)

Der 1. Parameter: gibt die Nummer des Motors an, der eingeschaltet wird

Der 2. Parameter: gibt die Position an, auf die der Servo-Motor eingestellt wird

www.pria.at 5

PRIA Practical Robotics Institute Austria

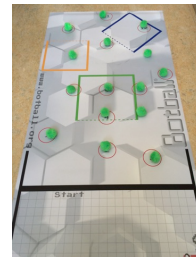
Nicht in einem Schritt

- ⚙ Achtung!
- ⚙ Nicht auf einmal von ganz oben nach ganz unten!
 - ⚙ Bei schnellen Bewegungen schießen sie gerne über das Ziel hinaus
 - ⚙ Schrittweise annähern

www.pria.at 7

PRIA Practical Robotics Institute Austria

Ziel: Aufräumen

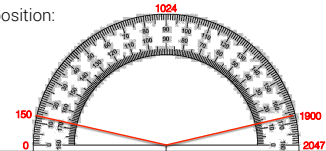


www.pria.at 2

PRIA Practical Robotics Institute Austria

Servo-Motor steuern

- ⚙ Servo-Motor
 - ⚙ Kann keine volle Umdrehung machen
 - ⚙ Wird auf eine Winkelposition eingestellt
 - ⚙ Nur Werte zwischen **900 und 1360** verwenden!
 - ⚙ Standardposition: 1024



www.pria.at 4

PRIA Practical Robotics Institute Austria

Servo-Funktionen

- ⚙ Servo-Motoren aktivieren
`enable_servos();`
- ⚙ Servo-Motoren deaktivieren
`disable_servos();`

www.pria.at 6

PRIA Practical Robotics Institute Austria

Beispiel

```
// Created on Do Apr 16 2015
int main()
{
  set_servo_position(1, 900);
  enable_servos();
  wait_for_milliseconds(1000);
  set_servo_position(1, 1100);
  wait_for_milliseconds(1000);
  set_servo_position(1, 1300);
  wait_for_milliseconds(1000);
  set_servo_position(1, 1350);
  wait_for_milliseconds(1000);
  return 0;
}
```

Mitmachen

www.pria.at 8

- ✿ Überlege dir ein Programm, das den Roboter winken lässt (den Arm hinauf und hinunter bewegt)
 - ✿ Welche Funktionen sind für dieses Programm sinnvoll
 - ✿ Was müsste man tun, damit das Winken nicht so ruckartig erfolgt

Aufgabe

www.pria.at

9

```

// Created on Mo Apr 20 2015
void hinauf();
void hinunter();
int main()
{
    set_servo_position(1, 1000);
    enable_servos();
    hinauf();
    hinunter();
    hinauf();
    hinunter();
    disable_servos();
    return 0;
}

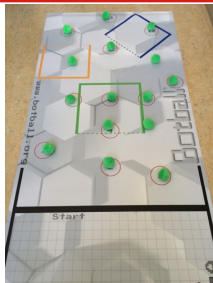
void hinauf()
{
    set_servo_position(1, 1300);
    wait_for_milliseconds(300);
    set_servo_position(1, 1200);
    wait_for_milliseconds(300);
    set_servo_position(1, 1100);
    wait_for_milliseconds(300);
    set_servo_position(1, 1000);
    wait_for_milliseconds(300);
    set_servo_position(1, 900);
}

void hinunter()
{
    set_servo_position(1, 900);
    wait_for_milliseconds(300);
    set_servo_position(1, 1000);
    wait_for_milliseconds(300);
    set_servo_position(1, 1100);
    wait_for_milliseconds(300);
    set_servo_position(1, 1200);
    wait_for_milliseconds(300);
    set_servo_position(1, 1300);
}
    
```

www.pria.at

11

- ✿ Auf jedem Zahlenfeld ein Pom
- ✿ 3 Poms werden in den freien Bereichen fallen gelassen



Wettbewerb

www.pria.at

13

- ✿ Damit sich der Arm auf und ab bewegt, sind 2 Funktionen sinnvoll
 - ✿ hinauf ... Für das Hinauf-Bewegen des Armes
 - ✿ hinunter ... Für das Hinunter-Bewegen des Armes
- ✿ Damit er sich möglichst ruhig bewegt, müsste man den Servo in ganz kleinen Schritten bewegen.

Entwurf

www.pria.at

10

- ✿ Starte so in der Startbox, dass kein Teil des Roboters über die Begrenzungslinie hinausragt.
- ✿ Räume innerhalb von 5 Minuten alle Poms vom Feld herunter, so dass zumindest ein Teil des Poms den Boden berührt
- ✿ Kehre innerhalb der 5 Minuten zur Startbox zurück.
- ✿ Ist der Roboter am Ende nicht in der Startbox, so zählt nur die Hälfte der Poms

Wettbewerb

www.pria.at

12

1. Jedes Pom, das zumindest zum Teil den Boden berührt, bringt einen Siegpunkt, Das Team mit den meisten Siegpunkten gewinnt.
2. Bei Gleichstand nach 1. wird der Programmcode gewertet. Je weniger Zeilen desto besser (Leerzeilen, return 0; Zeilen die nur Kommentare enthalten bzw. Zeilen die nur { oder } enthalten zählen nicht).
3. Bei Gleichstand nach 1. + 2. gewinnt das Team, dessen Roboter die kürzeste Zeit gebraucht hat.

Wettbewerb

www.pria.at

14

PRIA Practical Robotics Institute Austria

Die Umwelt wahrnehmen

Arbeiten mit Sensoren

PRIA Practical Robotics Institute Austria

Wiederholung

- * Variablen sind Gefäße zum Speichern von Inhalt
 - * Sie werden mit ihrer Größe (**int**) und dem Namen erzeugt
- * Variablen können verwendet, um Funktionen konfigurierbar zu machen. Sie werden dann Parameter genannt
 - * Parameter werden zwischen die Klammern in den Parameterbereich einer Funktion geschrieben
 - * Eine Funktion kann mehrere Parameter haben
 - * Wenn ein oder mehrere Parameter bei einer Funktion definiert wurden, dann muss man sie auch bei der Verwendung angeben

PRIA Practical Robotics Institute Austria

Hin und wieder zurück

- * Schreibe ein Programm, das bis mit der neuen motor-Funktion zum Ende der Matte fährt und dann umdrehen und wieder zurück
 - * Welche Funktionen sind sinnvoll?
 - * Haben die Funktionen Parameter?

Aktivität

PRIA Practical Robotics Institute Austria

Umsetzung im Programm

```

// Hin und wieder zurück
// Created on Mo. März 9 2015
void gerade_fahren(
int geschwind, int zeit);
void umdrehen();

int main()
{
  gerade_fahren(70, 6000);
  umdrehen();
  gerade_fahren(70, 6000);
  // Alle Motoren ausschalten
  ao();
  // Programm beenden
  return 0;
}

void gerade_fahren(
int geschwind, int zeit)
{
  // Motoren starten
  motor(1, geschwind);
  motor(2, geschwind);
  // Wartezeit für zeit ms
  wait_for_milliseconds(zeit);
}

void umdrehen()
{
  // 1. Motor m. 70% starten
  motor(1, 70);
  // 2. Motor m. -70% starten
  motor(2, -70);
  // Wartezeit für 1,4 s
  wait_for_milliseconds(1400);
}

```

PRIA Practical Robotics Institute Austria

Wiederholung

- * Mit Hilfe von 3 Schritten kann man Funktionen selber schreiben:
 - * Funktion nach der main-Funktion definieren mit `void Funktionsname()`

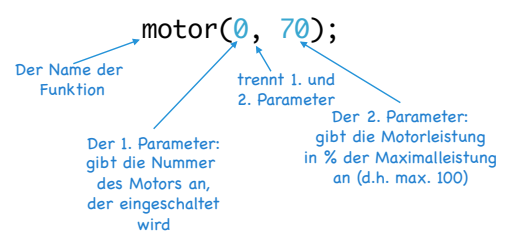
```

{
  Inhalt
}

```
 - * Prototypen vor die main-Funktion kopieren (; nicht vergessen)
 - * Funktion innerhalb der main-Funktion verwenden (genauso wie bereits vorhandene Funktionen)

PRIA Practical Robotics Institute Austria

Eine andere Motor-Funktion



PRIA Practical Robotics Institute Austria

Entwurf

- * Eine Funktion gerade_fahren mit 2 Parametern ist am flexibelsten
 1. Parameter: für die Geschwindigkeit geschwind
 2. Parameter: für die Zeitdauer zeit
- * Eine 2. Funktion für das Umdrehen

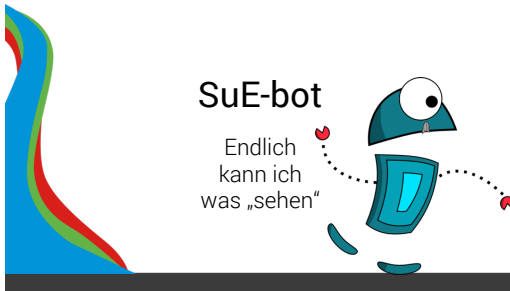
Entwurf

PRIA Practical Robotics Institute Austria

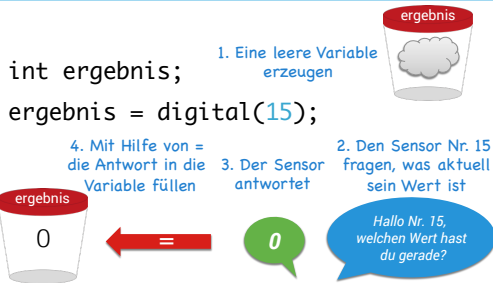
Review

- * Wie lange mussten die Motoren vor fahren, bis der Roboter das Ende der Matte erreicht hatten?
- * Wie bist du auf diesen Wert gekommen?

Review

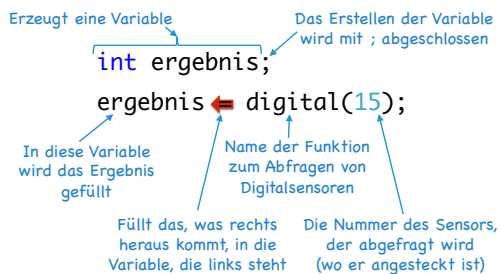


- Gehört zu den Digital-Sensoren
 - Entweder gedrückt: 1
 - Oder nicht gedrückt: 0
- Sensorwert abfragen:
 - Eine Variable zum Speichern des Ergebnis erzeugen
 - Die Funktion zum Abfragen des Sensors aufrufen und das Ergebnis in die Variable füllen



```
// Created on Mo. März 9 2015
int main()
{
    // Variable ergebnis erzeugen
    int ergebnis;
    // Digitalsensor Nr. 15 abfragen
    ergebnis = digital(15);
    // Inhalt von ergebnis ausgeben
    printf("Antwort: %d", ergebnis);
    // Programm beenden
    return 0;
}
```

- Der Roboter kann seine Umgebung wahrnehmen
- Dazu bedient er sich seiner Sensoren
- Die Sensoren sind mit dem Controller verbunden
- Auf Anfrage liefern sie einen Wert
 - Digitalsensoren: 0 oder 1
 - Analogsensoren: von 0 bis 1023
- Achtung! Die Sensoren arbeiten nicht mit der mav-Funktion zusammen



- Mit Hilfe der printf-Funktion kann das, was in einer Variablen steht ausgegeben werden.
- ```
int ergebnis;
ergebnis = digital(15);
printf("Antwort: %d", ergebnis);
```
- Im Ausgabertext kommt noch ein Platzhalter für den Inhalt der Variablen dazu
- Hier wird nachgesehen, was in der Variablen enthalten ist. Dieser Wert wird statt dem Platzhalter eingesetzt

- Was wird ausgegeben, wenn das Programm gestartet wird?
- Wie kann man einen 1er vom Programm anzeigen lassen?



**PRIA** Practical Robotics Institute Austria  
**Der Ablauf genauer betrachtet**

```
int antwort;
gerade_start(700);

do
{
 antwort = digital(15);
}
while(antwort == 0);

// Alle Motoren
ao();
```

Den Berührungssensor fragen, welchen Wert er gerade hat

Halo Nr. 15, welchen Wert hast du gerade?

0

**PRIA** Practical Robotics Institute Austria  
**Der Ablauf genauer betrachtet**

```
int antwort;
gerade_start(700);

do
{
 antwort = digital(15);
}
while(antwort == 0);

// Alle Motoren ausschalten
ao();
```

Bei der Variablen antwort nachschauen, welcher Wert gespeichert ist.

Ist die Zahl in antwort gleich 0?  
 „Ja, stimmt“

**PRIA** Practical Robotics Institute Austria  
**Der Ablauf genauer betrachtet**

```
int antwort;
gerade_start(700);

do
{
 antwort = digital(15);
}
while(antwort == 0);

// Alle Motoren ausschalten
ao();
```

Beginn der Schleifenstruktur (die Variable ist noch immer da)

**PRIA** Practical Robotics Institute Austria  
**Der Ablauf genauer betrachtet**

```
int antwort;
gerade_start(700);

do
{
 antwort = digital(15);
}
while(antwort == 0);

// Alle Motoren
ao();
```

Den Berührungssensor fragen, welchen Wert er gerade hat

Halo Nr. 15, welchen Wert hast du gerade?

0

**PRIA** Practical Robotics Institute Austria  
**Der Ablauf genauer betrachtet**

```
int antwort;
gerade_start(700);

do
{
 antwort = digital(15);
}
while(antwort == 0);

// Alle Motoren ausschalten
ao();
```

Bei der Variablen antwort nachschauen, welcher Wert gespeichert ist.

Ist die Zahl in antwort gleich 0?  
 „Ja, stimmt“

**PRIA** Practical Robotics Institute Austria  
**Der Ablauf genauer betrachtet**

```
int antwort;
gerade_start(700);

do
{
 antwort = digital(15);
}
while(antwort == 0);
```

Beginn der Schleifenstruktur (die Variable ist noch immer da)

**PRIA** Practical Robotics Institute Austria  
**Der Ablauf genauer betrachtet**

```
int antwort;
gerade_start(700);

do
{
 antwort = digital(15);
}
while(antwort == 0);

// Alle Motoren
ao();
```

Den Berührungssensor fragen, welchen Wert er gerade hat

Halo Nr. 15, welchen Wert hast du gerade?

1

**PRIA** Practical Robotics Institute Austria  
**Der Ablauf genauer betrachtet**

```
int antwort;
gerade_start(700);

do
{
 antwort = digital(15);
}
while(antwort == 0);

// Alle Motoren ausschalten
ao();
```

Bei der Variablen antwort nachschauen, welcher Wert gespeichert ist.

Ist die Zahl in antwort gleich 0?  
 „Nein, stimmt nicht“

## PRI4 Practical Robotics Institute Austria Der Ablauf genauer betrachtet

```
int antwort;
gerade_start(700);

do
{
 antwort = digital(15);
}
while(antwort == 0);

// Alle Motoren ausschalten
ao();
```

Die Schleifenstruktur ist zu Ende. Das Programm macht danach weiter



www.pri4.at

33

## PRI4 Practical Robotics Institute Austria Zwischen 2 Wänden

- Modifiziere dein Hin-Und-Wieder-Zurück-Programm so, dass dein Roboter zwischen 2 Hindernissen Hin und Her fährt.

- Wenn du an einer Wand angekommen bist, muss der Roboter noch ein Stück zurück fahren, bevor er sich umdreht. So hat er genug Platz zum Umdrehen

Aktivität

www.pri4.at

34

## PRI4 Practical Robotics Institute Austria Umsetzung im Programm

```
void kurz_zurueck();
void umdrehen();
void bis_zum_anschlag(int geschwindigkeit);

int main()
{
 bis_zum_anschlag(500);
 kurz_zurueck();
 umdrehen();
 bis_zum_anschlag(500);
 // Alle Motoren ausschalten
 ao();
 // Programm beenden
 return 0;
}

void kurz_zurueck()
{
 // Beide Motoren mit 50% starten
 motor(1, 50);
 motor(2, 50);
 // Wartezeit für 0,5 s
 wait_for_milliseconds(500);
}

void umdrehen()
{
 // 1. Motor m. 70% starten
 motor(1, 70);
 // 2. Motor m. -70% starten
 motor(2, -70);
 // Wartezeit für 4 s
 wait_for_milliseconds(1400);
}

void bis_zum_anschlag(int geschwindigkeit)
{
 // Variable antwort erzeugen
 int antwort;
 // Motoren m. geschwindigkeit starten
 motor(1, geschwindigkeit);
 motor(2, geschwindigkeit);
 do // mache das Folgende
 {
 // Digitalsensor Nr. 15 fragen
 antwort = digital(15);
 } // solange antwort gleich 0 ist
 while(antwort == 0);
}
```

www.pri4.at

35

## PRI4 Practical Robotics Institute Austria Parcours abfahren

- Lass deinen Roboter in die grüne Garage finden, wobei die Garage nun durch Wände begrenzt ist
- Lass deinen Roboter in die blaue Garage finden, wobei die Garage nun durch Wände begrenzt ist
- Lass deinen Roboter in die gelbe Garage finden, wobei die Garage nun durch Wände begrenzt ist

Herausforderung

www.pri4.at

36

## PRI4 Practical Robotics Institute Austria SuE-Bot - Suchen

- Ändere (oder schreibe) ein Sue-Bot-Programm, dass das ganze Zimmer absucht

Sue-Bot



www.pri4.at

37



# List of Figures

|      |                                                                                                                                   |    |
|------|-----------------------------------------------------------------------------------------------------------------------------------|----|
| 2.1  | didactic relation in the didactic triangle [33]                                                                                   | 3  |
| 3.1  | Lego Mindstorms EV3 IDE                                                                                                           | 12 |
| 3.2  | RobotC IDE                                                                                                                        | 13 |
| 3.3  | LeJOS with Eclipse                                                                                                                | 14 |
| 3.4  | KISS IDE                                                                                                                          | 15 |
| 3.5  | Simulator for the Link Controller                                                                                                 | 16 |
| 3.6  | Robotino View                                                                                                                     | 17 |
| 3.7  | Robotino SIM simulator                                                                                                            | 17 |
| 3.8  | Robotino C++ in Visual Studio [58]                                                                                                | 18 |
| 4.1  | The robot build for the workshop                                                                                                  | 22 |
| 4.2  | Basic concept                                                                                                                     | 23 |
| 4.3  | Team positions in a 3 person team                                                                                                 | 25 |
| 5.1  | The computer does not understand human language                                                                                   | 30 |
| 5.2  | The principle of giving a computer an instruction it understands                                                                  | 31 |
| 5.3  | The KIPR link controller (picture courtesy of KIPR)                                                                               | 32 |
| 5.4  | The principle of giving a computer an instruction it understands combined with the steps necessary on the computer and controller | 33 |
| 5.5  | The basic structure of a program                                                                                                  | 34 |
| 5.6  | The basic structure of the main function                                                                                          | 34 |
| 5.7  | The area where the statements are written                                                                                         | 35 |
| 5.8  | Syntax error in the KISS IDE (missing ; after the printf statement)                                                               | 36 |
| 5.9  | Syntax error in the KISS IDE (missing " after the Hello, World! – only one error but many error messages)                         | 37 |
| 5.10 | Syntax error in the KISS IDE (printf is written with a upper case P)                                                              | 37 |
| 5.11 | Using a function in contrast to a structural statement                                                                            | 39 |
| 5.12 | The structure of a function call                                                                                                  | 39 |
| 5.13 | The structure of the wait_for_milliseconds function call                                                                          | 40 |
| 5.14 | The motors of the robot (one attached to each wheel)                                                                              | 41 |
| 5.15 | The structure of the mav function call                                                                                            | 42 |
| 5.16 | The structure of the ao function call                                                                                             | 42 |
| 5.17 | The design of the first movement program as a flowchart                                                                           | 43 |

|      |                                                                                                                               |    |
|------|-------------------------------------------------------------------------------------------------------------------------------|----|
| 5.18 | Different velocities per wheel let the robot drive a curve . . . . .                                                          | 44 |
| 5.19 | A negative number for the velocity parameter results in driving backwards . . . . .                                           | 45 |
| 5.20 | The scenario for challenge 1 . . . . .                                                                                        | 46 |
| 5.21 | The scenario for challenge 2 . . . . .                                                                                        | 47 |
| 5.22 | The scenario for challenge 3 . . . . .                                                                                        | 48 |
| 5.23 | Driving a square . . . . .                                                                                                    | 49 |
| 5.24 | The definition of the function drive_straight . . . . .                                                                       | 53 |
| 5.25 | Adding the prototype to the definition of the function drive_straight . . . . .                                               | 54 |
| 5.26 | The structure of calling of the function drive_straight . . . . .                                                             | 54 |
| 5.27 | The calling of the function drive_straight 4 times within the main-function . . . . .                                         | 55 |
| 5.28 | The robot drives a wiggly line . . . . .                                                                                      | 56 |
| 5.29 | The robot drives a rectangle . . . . .                                                                                        | 58 |
| 5.30 | Declaring a parameter and using it within the function . . . . .                                                              | 59 |
| 5.31 | The prototype must contain the parameter too . . . . .                                                                        | 60 |
| 5.32 | The structure of calling drive_straight with parameter . . . . .                                                              | 60 |
| 5.33 | The structure of calling drive_straight with parameter . . . . .                                                              | 61 |
| 5.34 | The structure of calling drive_straight with parameter . . . . .                                                              | 63 |
| 5.35 | A simple tool for clearing the mat . . . . .                                                                                  | 64 |
| 5.36 | The field of the clearing competition . . . . .                                                                               | 64 |
| 5.37 | The motor-function for use with sensors . . . . .                                                                             | 66 |
| 5.38 | How to call a function, which asks a sensor of its value . . . . .                                                            | 67 |
| 5.39 | The internal process when asking a sensor of its value . . . . .                                                              | 68 |
| 5.40 | How to call a function, which asks a analog sensor of its value . . . . .                                                     | 69 |
| 5.41 | Driving until “we are there” . . . . .                                                                                        | 70 |
| 5.42 | Structure of the do-while-loop . . . . .                                                                                      | 71 |
| 5.43 | Structure of the do-while-loop . . . . .                                                                                      | 72 |
| 5.44 | Field for the find the spot competition . . . . .                                                                             | 73 |
| 5.45 | Field for the find the spot competition . . . . .                                                                             | 73 |
| 6.1  | Evaluation strategy to cover motivation and expertise of the students . . . . .                                               | 77 |
| 6.2  | The structure of the quantitative part of the evaluation . . . . .                                                            | 78 |
| 6.3  | Number of students used in evaluating the expertise (students with interest in programming and 3 evaluation sheets) . . . . . | 81 |
| 6.4  | Modified evaluation strategy to enhance significance . . . . .                                                                | 82 |
| 6.5  | Student interest during the workshop . . . . .                                                                                | 82 |
| 6.6  | Interest in pursuing a technical career at the begin and the end of the workshop . . . . .                                    | 83 |
| 6.7  | Overall grade results with different grading methods compared to exam results from 2011 and 2012 . . . . .                    | 87 |
| A.1  | Selecting a new project to start . . . . .                                                                                    | 93 |
| A.2  | Naming your project . . . . .                                                                                                 | 94 |
| A.3  | Choosing a template and naming your program . . . . .                                                                         | 94 |
| A.4  | The KISS IDE with the “Hello, World!” example program . . . . .                                                               | 95 |
| A.5  | The KIPR link controller from the right (picture courtesy of KIPR) . . . . .                                                  | 95 |

A.6 Start screen of the KIPR link controller . . . . . 95  
A.7 The back side of the KIPR link controller (picture courtesy of KIPR) . . . . . 96  
A.8 Starting the compiling with the “Compile” button . . . . . 96  
A.9 Selecting the via USB connected controller . . . . . 96  
A.10 Showing all available programs on the controller . . . . . 96  
A.11 Running the first example program . . . . . 97  
A.12 Selecting the Sensor List . . . . . 98  
A.13 Deactivating the Pull-up . . . . . 98



# List of Tables

|     |                                                                                 |    |
|-----|---------------------------------------------------------------------------------|----|
| 5.1 | Example for an 8-bit binary code . . . . .                                      | 30 |
| 5.2 | Example for an ASCII encoding of characters . . . . .                           | 30 |
| 5.3 | Binary code for instructions . . . . .                                          | 31 |
| 5.4 | Programming language instructions with binary code translation . . . . .        | 32 |
| 5.5 | Error scenarios the students should try and record . . . . .                    | 38 |
| 5.6 | Operators usable for comparison . . . . .                                       | 71 |
| 6.1 | Symbols used for evaluating interest statements . . . . .                       | 78 |
| 6.2 | Statements used for assessing interest levels . . . . .                         | 79 |
| 6.3 | Questions used in multiple-choice and short answer parts of the exams . . . . . | 79 |
| 6.4 | Attendance at the workshop . . . . .                                            | 80 |
| 6.5 | Grading results . . . . .                                                       | 86 |



# Listings

|      |                                                                                     |    |
|------|-------------------------------------------------------------------------------------|----|
| 5.1  | Comment inside the main-function . . . . .                                          | 35 |
| 5.2  | Two printf commands . . . . .                                                       | 36 |
| 5.3  | Greeting of three team members . . . . .                                            | 40 |
| 5.4  | Slow greeting of three team members . . . . .                                       | 41 |
| 5.5  | The design of the first move program written as comments . . . . .                  | 43 |
| 5.6  | The complete first move program . . . . .                                           | 44 |
| 5.7  | Driving a square - first try . . . . .                                              | 51 |
| 5.8  | Driving a square - second try . . . . .                                             | 54 |
| 5.9  | Driving a rectangle . . . . .                                                       | 62 |
| 5.10 | Asking a sensor for its value . . . . .                                             | 67 |
| 6.1  | Program used for evaluation of understanding of functions with parameters . . . . . | 78 |





# Acronyms

**API** application programming interface. 12, 20

**COM** component object model. 16

**EV3** evolution 3 - the current version of the LEGO Mindstorms controller. 11

**FAQ** frequently asked questions. 12

**IDE** integrated development environment. 12, 13, 15, 16, 19, 26, 29, 32, 123

**JVM** Java virtual machine. 13

**KIPR** KISS institute of practical robotic. 14, 15

**NXT** next - the second version of the LEGO Mindstorms controller. 11, 12, 19

**PC** personal computer. 16

**PRIA** Practical robotics institute Austria. 75

**RCX** robotic command explorer - the first version of the LEGO Mindstorms controller. 11, 19

**SE** standard edition. 13



# Bibliography

- [1] *Education in figures 2013/14 - key indicators and analyses*. Statistik Austria, 4 2015.
- [2] Edith Ackermann. Piaget’s constructivism, papert’s constructionism: What’s the difference. *Future of learning group publication*, 5(3):438, 2001.
- [3] Dimitris Alimisis. Educational robotics: Open questions and new challenges. *Themes in Science and Technology Education*, 6(1):pp–63, 2013.
- [4] Dimitris Alimisis and Chronis Kynigos. Constructionism and robotics in education. *Teacher Education on Robotic-Enhanced Constructivist Pedagogical Methods*, pages 11–26, 2009.
- [5] Heilo Altin and Margus Pedaste. Learning approaches to applying robotics in science education. *Journal of baltic science education*, 12(3):365–377, 2013.
- [6] James M Applefield, Richard Huber, and Mahnaz Moallem. Constructivism in theory and practice: Toward a better understanding. *The High School Journal*, pages 35–53, 2000.
- [7] Owen Astrachan, Kim Bruce, Elliot Koffman, Michael Kölling, and Stuart Reges. Resolved: Objects early has failed. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE ’05, pages 451–452, New York, NY, USA, 2005. ACM.
- [8] Frances Bailie, Mary Courtney, Keitha Murray, Robert Schiaffino, and Sylvester Tuohy. Objects first - does it work? *J. Comput. Sci. Coll.*, 19(2):303–305, December 2003.
- [9] David J. Barnes. Teaching introductory java through lego mindstorms models. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, SIGCSE ’02, pages 147–151, New York, NY, USA, 2002. ACM.
- [10] Mordechai Ben-Ari. Constructivism in computer science education. In *Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE ’98, pages 257–261, New York, NY, USA, 1998. ACM.
- [11] Fabiane Barreto Vavassori Benitti. Exploring the educational potential of robotics in schools: A systematic review. *Computers & Education*, 58(3):978 – 988, 2012.

- [12] Anders Berglund and Raymond Lister. Introductory programming and the didactic triangle. In *Proceedings of the Twelfth Australasian Conference on Computing Education - Volume 103, ACE '10*, pages 35–44, Darlinghurst, Australia, Australia, 2010. Australian Computer Society, Inc.
- [13] M. Beynon and C. Roe. Computer support for constructionism in context. In *Advanced Learning Technologies, 2004. Proceedings. IEEE International Conference on*, pages 216–220, Aug 2004.
- [14] John D. Bransford, Ann L. Brown, Rodney R. Cocking, M. Suzanne Donovan, and James W. Pellegrino, editors. *How People Learn: Brain, Mind, Experience, and School: Expanded Edition*. National Academies Press, 2000.
- [15] Charles M Brooks and Janice L Ammons. Free riding in group projects and the effects of timing, frequency, and specificity of criteria in peer assessments. *Journal of Education for Business*, 78(5):268–272, 2003.
- [16] Kim B. Bruce. Controversy on how to teach cs 1: A discussion on the sigcse-members mailing list. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '04, pages 29–34, New York, NY, USA, 2004. ACM.
- [17] Jim Bumgardner. The origins of mindstorms. *Wired*, 03 2007.
- [18] David T Butterworth. Teaching c/c++ programming with lego mindstorms. In *Proc. 3rd Int. Conf. on Robotics in Education (RiE 2012), Prague, Czech Republic*, pages 61–65, 2012.
- [19] W. Campbell and E. Bolker. Teaching programming by immersion, reading and writing. In *Frontiers in Education, 2002. FIE 2002. 32nd Annual*, volume 1, pages T4G–23–T4G–28 vol.1, 2002.
- [20] Martin V. Covington. Goal theory, motivation, and school achievement: An integrative review. *Annual Review of Psychology*, 51(1):171–200, 2000. PMID: 10751969.
- [21] Azi Crawford. Botball programming. <http://botballprogramming.org/>.
- [22] R. Czaja and J. Blair. *Designing Surveys: A Guide to Decisions and Procedures*. Research Methods and Statistics Series. SAGE Publications, 2005.
- [23] Nikolaos Detsikas and Dimitris Alimisis. Status and trends in educational robotics world-wide with special consideration of educational experiences from greek schools. In *Proceedings of the International Conference on Informatics in Schools: Situation, Evolution and Perspectives*, pages 1–12, 2011.
- [24] eclipse foundation. eclipse.org. <https://www.eclipse.org>.

- [25] Festo. Festo - education and research robots: Robotino. <http://www.festo-didactic.com/us-en/products/education-and-research-robots-robotino/>, 2014.
- [26] Sergey Filippov, Alexander L Fradkov, and Boris Andrievsky. Teaching of robotics and control jointly in the university and in the high school based on lego mindstorms nxt. In *Actes 18th IFAC World Congress, Milan (Italie)*, pages 9824–9829, 2011.
- [27] KISS Institute for Practical Robotics. Botball educational robotics program. <http://www.botball.org/>, 2014.
- [28] Sonia M Goltz, Amy B Hietapelto, Roger W Reinsch, and Sharon K Tyrell. Teaching teamwork and problem solving concurrently. *Journal of Management Education*, 32(5):541–562, 2008.
- [29] Matthias Hirschmanner, Lara Lammer, and Markus Vincze. Mattie: A simple educational platform for children to realize their first robot prototype. In *Proceedings of the 14th International Conference on Interaction Design and Children, IDC '15*, pages 367–370, New York, NY, USA, 2015. ACM.
- [30] A.J. Hirst, J. Johnson, M. Petre, B.A. Price, and M. Richards. What is the best programming environment/language for teaching robotics using lego mindstorms? *Artificial Life and Robotics*, 7(3):124–131, 2003.
- [31] Tony Jenkins. The motivation of students of programming. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE '01*, pages 53–56, New York, NY, USA, 2001. ACM.
- [32] Tony Jenkins. On the difficulties of learning to program. In *Proceedings of the 3rd Annual Conference on the Teaching of Programming*, 2002.
- [33] Pertti Kansanen and Matti Meri. The didactic relation in the teaching-studying-learning process. *Didaktik/Fachdidaktik as Science (-s) of the Teaching profession*, 2(1):107–116, 1999.
- [34] ElizabethR. Kazakoff, Amanda Sullivan, and MarinaU. Bers. The effect of a classroom-based intensive robotics and programming workshop on sequencing ability in early childhood. *Early Childhood Education Journal*, 41(4):245–255, 2013.
- [35] Michael Kölling and John Rosenberg. Guidelines for teaching object orientation with java. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE '01*, pages 33–36, New York, NY, USA, 2001. ACM.
- [36] Chronis Kynigos. Black-and-white-box perspectives to distributed control and constructionism in learning with robotics. In *Proceedings of SIMPAR workshops*, pages 1–9, 2008.

- [37] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A study of the difficulties of novice programmers. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '05, pages 14–18, New York, NY, USA, 2005. ACM.
- [38] Pamela B. Lawhead, Michael E. Duncan, Constance G. Bland, Michael Goldweber, Madeleine Schep, David J. Barnes, and Ralph G. Hollingsworth. A road map for teaching introductory programming using lego&copy; mindstorms robots. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '02, pages 191–201, New York, NY, USA, 2002. ACM.
- [39] Lego. Home - mindstorms lego.com. <http://mindstorms.lego.com>.
- [40] Lego. Lego mindstorms support. <http://www.lego.com/en-us/mindstorms/support>.
- [41] Yi-Guang Lin, Wilbert J McKeachie, and Yung Che Kim. College student intrinsic and/or extrinsic motivation and learning. *Learning and Individual Differences*, 13(3):251 – 258, 2001.
- [42] Marcia C. Linn and Michael J. Clancy. The case for case studies of programming problems. *Commun. ACM*, 35(3):121–132, March 1992.
- [43] Marcia C. Linn and John Dalbey. *Studying the Novice Programmer*, chapter 4. Cognitive consequences of Programming Instruction. Lawrence Erlbaum Associates, 1989.
- [44] L. Major, T. Kyriacou, and O.P. Brereton. Systematic literature review: teaching novices programming using robots. *Software, IET*, 6(6):502–513, Dec 2012.
- [45] Richard E. Mayer. *Studying the Novice Programmer*, chapter 7. The Psychology of How Novices Learn Computer Programming. Lawrence Erlbaum Associates, 1989.
- [46] Karolina Mayerová. Pilot activities: Lego wedo at primary school. In *Proceedings of 3rd International Workshop Teaching Robotics, Teaching with Robotics*, pages 32–39, 2012.
- [47] Dawn McKinney and Leo F. Denton. Developing collaborative skills early in the cs curriculum in a laboratory environment. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '06, pages 138–142, New York, NY, USA, 2006. ACM.
- [48] Jerry Mead, Simon Gray, John Hamer, Richard James, Juha Sorva, Caroline St. Clair, and Lynda Thomas. A cognitive approach to identifying measurable milestones for programming skill acquisition. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '06, pages 182–194, New York, NY, USA, 2006. ACM.
- [49] Orazio Miglino, Henrik Hautop Lund, and Maurizio Cardaci. Robotics as an educational tool. *Journal of Interactive Learning Research*, 10(1):25–47, 1999.

- [50] Iain Milne and Glenn Rowe. Difficulties in learning and teaching programming - views of students and tutors. *Education and Information Technologies*, 7(1):55–66, 2002.
- [51] Oracle. Java for lego mindstorms ev3. <http://www.oracle.com/technetwork/java/embedded/downloads/javase/javaseemdedev3-1982511.html>.
- [52] Seymour Papert. Teaching children thinking\*. *Programmed Learning and Educational Technology*, 9(5):245–255\*, 1972.
- [53] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA, 1980.
- [54] Seymour Papert and Idit Harel. Situating constructionism. *Constructionism*, 36:1–11, 1991.
- [55] Marian Petre and Blaine Price. Using robotics to motivate ‘back door’ learning. *Education and Information Technologies*, 9(2):147–158, 2004.
- [56] Poppy-team. Poppy project. <https://www.poppy-project.org/>, 2015.
- [57] Ralph T. Putnam, D. Sleeman, Juliet A. Baxter, and Laiani K. Kuspa. *Studying the Novice Programmer*, chapter 15. A Summary of Misconceptions of High School Basic Programmers. Lawrence Erlbaum Associates, 1989.
- [58] REC-GmbH. Robotino documentation. <http://doc.openrobotino.org/documentation/OpenRobotinoApiHowTo/HTML/index.html>, 2010.
- [59] Stuart Reges. Back to basics in cs1 and cs2. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE ’06, pages 293–297, New York, NY, USA, 2006. ACM.
- [60] Mitchel Resnick. Distributed constructionism. In *Proceedings of the 1996 International Conference on Learning Sciences*, ICLS ’96, pages 280–284. International Society of the Learning Sciences, 1996.
- [61] Mitchel Resnick, Robbie Berg, and Michael Eisenberg. Beyond black boxes: Bringing transparency and aesthetics back to scientific investigation. *Journal of the Learning Sciences*, 9(1):7–30, 2000.
- [62] Eric Roberts. Strategies for encouraging individual achievement in introductory computer science courses. In *Proceedings of the Thirty-first SIGCSE Technical Symposium on Computer Science Education*, SIGCSE ’00, pages 295–299, New York, NY, USA, 2000. ACM.
- [63] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003.
- [64] ROBOTC.NET. Robotc.net: Home of the best robot programming language for educational robotics. <http://www.robotc.net>.

- [65] Natalie Rusk, Mitchel Resnick, Robbie Berg, and Margaret Pezalla-Granlund. New pathways into robotics: Strategies for broadening participation. *Journal of Science Education and Technology*, 17(1):59–69, 2008.
- [66] Marco Ruzzenente, Moreno Koo, Katherine Nielsen, Lorenzo Grespan, and Paolo Fiorini. A review of robotics kits for tertiary education. In *Proceedings of International Workshop Teaching Robotics Teaching with Robotics: Integrating Robotics in School Curriculum*, pages 153–162, 2012.
- [67] Elliot Soloway and James C. Spohrer, editors. *Studying the Novice Programmer*. Lawrence Erlbaum Associates, 1989.
- [68] Gary Stager. Papertian constructionism and the design of productive contexts for learning. *Proceedings of EuroLogo 2005*, 2005.
- [69] Gary S. Stager. A constructionist approach to teaching with robotics. *Proceedings for Constructionism 2010*, 2010.
- [70] Cathryne Stein. Botball: Autonomous students engineering autonomous robots. In *Proceedings of the ASEE Conference*, 2002.
- [71] Florence R. Sullivan. Robotics and science literacy: Thinking skills, science process skills and systems understanding. *Journal of Research in Science Teaching*, 45(3):373–394, 2008.
- [72] Phit-Huan Tan, Choo-Yee Ting, and Siew-Woei Ling. Learning difficulties in programming courses: Undergraduates’ perspective and perception. In *Computer Technology and Development, 2009. ICCTD '09. International Conference on*, volume 1, pages 42–46, Nov 2009.
- [73] Donna Teague and Paul Roe. Collaborative learning: Towards a solution for novice programmers. In *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78, ACE '08*, pages 147–153, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.
- [74] Christopher Watson and Frederick W.B. Li. Failure rates in introductory programming revisited. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education, ITiCSE '14*, pages 39–44, New York, NY, USA, 2014. ACM.
- [75] H. Weinert and D. Pensky. Mobile robotics in education and student engineering competitions. In *AFRICON, 2011*, pages 1–5, Sept 2011.
- [76] A.B. Williams. The qualitative impact of using lego mindstorms robots to teach computer engineering. *Education, IEEE Transactions on*, 46(1):206–, Feb 2003.
- [77] Leon E. Winslow. Programming pedagogy - a psychological overview. *SIGCSE Bull.*, 28(3):17–22, September 1996.



- [78] Christopher A. Wolters, Shirley L. Yu, and Paul R. Pintrich. The relation between goal orientation and students' motivational beliefs and self-regulated learning. *Learning and Individual Differences*, 8(3):211 – 238, 1996. Special Issue: A Symposium on Self-Regulated Learning.
- [79] Tom Wulf. Constructivist approaches for teaching computer programming. In *Proceedings of the 6th Conference on Information Technology Education, SIGITE '05*, pages 245–248, New York, NY, USA, 2005. ACM.