Exciting Students for Systems Programming Through the Use of Mobile Robots

Alexander Förster
Roboter und Technik
Bremen, Germany
technik-in-die-schule.de

Anna Förster
University of Bremen
Bremen, Germany
afoerster@comnets.uni-bremen.de

Jürgen Leitner
Centre of Excellence
for Robotic Vision
Queensland University of Technology
j.leitner@qut.edu.au

ABSTRACT

In this paper we present our experience teaching Systems Programming in C to undergraduate students[1]. Additionally to traditional Unix-like operating system approach, we employed a robotic platform – the e-puck mobile robot – to increase the students' motivation and improve their learning experience. A robotic platform provides high attraction for students, making the class stand-out compared to other courses. Yet it is not only a playground, rather, the platform allows to present very challenging and sophisticated real-life programming problems in a tangible way. The chosen robot provides an open-source operating system with a well structured programming interface and thus offers a real-world, complex example of systems programming to the students.

We describe the overall curriculum and the syllabus of the course itself. Emphasis is put on the design of the in-class and homework assignments, but the robotic platform is briefly described as well. Our success is confirmed by the end-of-semester evaluation by the students, who ranked our course among the top of all bachelor-level courses.

KEYWORDS

teaching, computer science, systems programming, bachelor, undergraduate

1 INTRODUCTION

Teaching systems or low-level programming at the bachelor level has developed into a challenging task. Not only are there a variety of embedded and operating systems available, all with their own quirks and specialities, but undergraduate students seem to get most excited these days about web programming, graphical user interfaces, game programming and lately smartphone programming. In comparison, theoretical computer science, calculus and low-level programming have a low attraction value and are generally perceived by students as "boring and outdated". However, every experienced computer scientist knows how pivotal this knowledge is for developing solid programming skills. Only these fields enable a deep understanding of hardware and software, their close interaction and enable high quality, reliable applications. Furthermore, innovations in both software and hardware often come down to the use of traditional programming languages, e.g. to develop a new programming language or to program a sophisticated system, such as a robot.

[1]This work has been conducted by the authors while still teaching at the University of Lugano, Switzerland, in 2011-2013.

In this paper, we present our experience of teaching a bachelor-level course in Systems Programming using mobile robots. These robots are very attractive to students for several reasons: they represent a hot and interesting research area with high future potential and a taste of science fiction; they offer a real platform for experimenting and programming; and last but not least, the developed applications are highly visible and worth showing to others as they surely attract the attention of colleagues and faculty. At the same time, robots are not only playful pals; in fact, they are currently one of the most challenging real-life hardware systems, employing numerous sensors and actuators and requiring complex programming.

The Systems Programming course at the Universita della Svizzera Italiana (USI) in Lugano has been taught in its current form for three years. It is a 6 ECTS (European Credit Transfer System) course and aims to teach the traditional topics of the C programming language [6], with its different standards (C99, C POSIX library), programming tools (compiler, debugger, source code documentation, build system), and access to operating system functionality (system calls). The course is loosely following the text book by [4]. However, instead of traditional programming assignments, we extensively use the e-puck robotic platform [7]. The outlook to program "their own robot" greatly motivated the students from the very beginning, enabling them to grasp the importance of each part of the course and its contents in an enjoyable way.

In the next sections, we first present in more detail the existing computer science curriculum at USI and the background of our course (Section 2). Section 3 describes our complete course syllabus, including the individual in-class and homework assignments and the timing of introducing the mobile robots. In Section 4 we give more details about the used robotic platform, the e-puck mobile robot, while Section 5 discusses the achievements of our course and the students' evaluations of it. Section 6 finally concludes the paper. We believe this paper can be used to re-design other low-level or systems programming courses by showing what worked for us. We believe that through the integration of the robot in the second half of the course we were able to better motivate the students and significantly increase their learning experience of a topic they considered "boring and outdated" at the beginning.

2 CURRICULUM AND BACKGROUND

USI is the youngest university in Switzerland. It was founded in 1996 and has since then become Switzerland's most international university. It is distinctive because of the originality of its degree curricula, including courses taught fully in English. While relatively

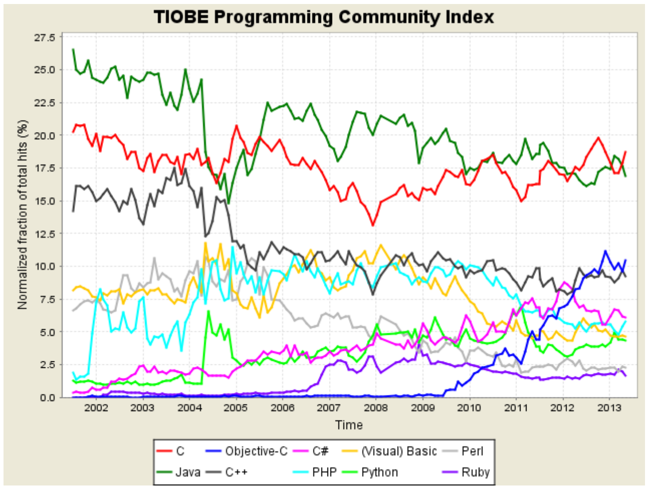


Figure 1: TIOBE Programming Community Index for May 2013 (courtesy: http://www.tiobe.com). Java was the number one over years and C at the second place. However, in the last couple of years C is growing and and overtaking the leadership.

contained in size it provides a collegiate, dynamic, and multicultural atmosphere. The Faculty of Informatics has been the most recent addition to USI, founded only in 2004. Its Bachelor of Science (BSc) curricula has been designed to better address the rapidly advancing and changing field of computer science [5]. To provide a pervasive view of current information technology (IT) and its developments, the faculty aims for a continuous integration of application areas and application-oriented projects into and across traditional courses. Furthermore, technology and theory are addressed together in the curriculum to better adapt to the rapidly changing nature of IT.

In the first years of the BSc curriculum at USI, systems programming was taught in a small 2 ECTS (European Credit Transfer System) course called “Programming in C”. The course was situated in the third semester, thus the students already had prior experience with programming, but mainly in Python and Java. The focus of this course was on teaching ANSI C programming, without the direct relationship to systems programming or other application areas. However, these courses were not met with great enthusiasm by the students, who generally complained about the “uselessness” of this course, teaching them a “dinosaur” programming language. This attitude was well understandable, as the students were already able to program quite complex applications with sophisticated graphical interfaces, yet were only taught to write simple programs to learn the C syntax. Compared to modern programming languages which are also suitable for the systems programming level, C has many drawbacks [9]. However, it is still broadly used for both historical and functional reasons. In fact, it recently became again the most commonly used programming language (see Figure 1). Therefore the faculty decided to continue teaching C, but started looking at ways to design a better course for the students.

To improve, one of the authors, while teaching this class, decided to introduce small projects at the end of the course to better

Table 1: Student performance over the semester.

Assignments	**Average grade (max 100)**
All students	84
Students with robotics projects	94
Students with sensor network projects	97
Students with software projects	73
Midterm exam	**Average grade (max 100)**
All students	40
Students with robotics projects	70
Students with sensor network projects	65
Students with software projects	51
Project	**Average grade (max 100)**
All students	77
Students with robotics projects	93
Students with sensor network projects	70
Students with software projects	73
Complete course	**Average grade (max 100)**
All students	68
Students with robotics projects	82
Students with sensor network projects	76
Students with software projects	58

demonstrate the application fields of ANSI C in modern IT. Over the years these projects spread over three fields:

- Software programming: small projects, e.g., a math terminal tool.
- Wireless sensor networks: small projects with the Scatterweb MSB430 nodes [1, 2].
- Robotics: small projects with the e-puck robot (see Section 4).

Table 1 presents some concrete performance results of the students over the semester. However, the total number of students was 19 (high for the University of Lugano, where typical courses have 5-20 students most), which makes the results statistically insignificant. However, it provided us with valuable experience and motivation.

Table 1 reads as follows. All students completed the same lectures and assignments until the midterm exam. Directly afterwards they selected a topic for the project and worked in their project until the end of the semester, when they received a grade for the project itself and for the complete course (which includes their grades for assignments, midterm and project). The data in the table is organised into the project topic the students have selected after the midterm. In the midterm exam, the students who later selected robotics or sensor networks, perfumed equally well. The others, who later chose the software project, performed worse than the other two groups. This can be easily explained the other way around: the weaker students chose a project topic, which they considered "simpler", because they did not want to invest time and effort in learning the software architectures and the user interfaces for robots or sensor networks.

However, what is most interesting is the development of the two stronger groups. In the midterm exam, they have shown almost identically good results. Then, during the project phase, the robotics group performed extraordinarily well, while the sensor networking group fell back. This could have two explanations: either the robot was easier to learn and work with than the sensor nodes or the motivation is greater. We believe the main reason is the first one, since working with both types of hardware required similarly steep learning curves. Thus, pure motivation and fun led the students to better learning results and better performance. This is also supported by their final results, which show that the robotics students perfumed best from all groups.

It is also interesting to compare the students' performance during the assignments and during the project. Especially for weaker students, who need more advise and support, the assignments are the better solution than a project.

Given the presented results and taking into account the students' comments and own ideas, we decided to implement the following two concepts:

- **In-class assignments with teacher support** throughout the whole course, instead of homework assignments and an individual project.
- **Usage of a mobile robot in the in-class assignment** to increase the motivation and the fun factor of the course.

The course was further extended to 6 ECTS to allow for sufficient time to complete the in-class assignments. The mobile robots could not be introduced immediately, but only in the middle of the course, since some experience with C is needed to get them running.

This new version of the course exists now in the faculty for two years and has been evaluated both by students and staff as one of the most attractive courses in the current BSc curriculum.

3 SYLLABUS

The Systems Programming class is a 6 ECTS course with 2 lectures per week and additional homework [11]. A non mandatory lab — led by the teaching assistant — is also offered to explain lecture material in more detail and answer comprehensive questions that might arise during the homework assignments.

The lectures can be considered traditional topics in systems programming and programming in C. However, we put special emphasis on in-class assignments, where the students can experiment with the presented theoretical concepts (like memory management, threads or debugging) under guidance and supervision from the teacher and teaching assistant. In addition homework assignments are given to encourage the reuse of the taught subjects in a more application/project-oriented way. The complete list of assignments with short descriptions is provided in Table 2. Each lecture of our course included at least one in-class assignment and every week an additional homework assignments was given. The homework was discussed in the lecture during assignment and again after the submission.

The syllabus is further structured into two main parts: programming in C (including tools, build systems and debugging) and systems programming with a robot. The first part, as can be seen also in Table 2, covers traditional programming in C, including advanced concepts, such as function pointers, multidimensional arrays, dynamic memory management, etc. Already in this first part of the course, it is very important to design the in-class and homework assignments in an attractive and practical way, so that students are able to recognise the merits and the need of the C language. This very often requires thorough preparation of the assignment, such as debugging an existing program (Lecture 6) or re-structuring a given program with the help of `struct` data types (Lecture 8). Furthermore, in order to gradually increase the difficulty and complexity of the assignments, they need to be built on top of each other, for example the sequence of in-class and homework assignments about binary trees starting at Lecture 11 and reaching until Lecture 16.

Another approach we found useful to keep the students engaged is to offer small "gags". For example, the homework after Lecture 9 is the implementation of a minimal program interpreter, based on the esoteric programming language called BrainFuck [8]. This language has only eight instructions, is Turing complete and is designed to amuse programmers with its rather unreadable code.

The second part of the course was concentrating on using systems programming on the mobile robots. At this point the students were already well acquainted with the necessary prerequisites, e.g. how to make libraries, makefiles, etc. Therefore programming a real robot seemed a manageable task, also to themselves. From the teaching staff's point of view, the time and effort invested into preparing the assignments increased further. We had to prepare a bare-bone programs, which the students were able to complete in the short time provided for a homework assignment and at the same time experience the satisfaction of seeing the robot perform relatively complex tasks. Again, the assignments built on top of each other by providing insight into the individual functionalities of the robot: uploading a program via Bluetooth, steering the robot via remote control, using the IR proximity sensors, flashing LEDs by configuring hardware timers or by using a timer API, etc. The final assignment in Lecture 24 was to write a program, where the robot randomly explores the environment by avoiding obstacles. Figure 2 presents a screenshot of a video taken by a student (Georgios Samaras) of his final assignment. The full video is available online [10].

As mentioned before a hands-on lab was also offered in addition to the lectures. This lab, held once a week, roughly mid-way between the assignment and its submission deadline, offered the students the possibility to ask questions about the assignments, clarify the topics taught in the lectures, and allow for experimentation with their code and the robot. While the lectures themselves included many hands-on programming tasks, this aspect had even more prominence in the lab. It encouraged the students to try various approaches to tackle given, or even self-created, programming tasks or solve small problems. The use of the robot enabled the students to get instant feedback of their code in a tangible way. This instant gratification for writing working code, e.g. when managing to use the infra-red sensors on the robot correctly to avoid obstacles, spurred them on to try different ways of optimising and further experimentation.

Additionally to the lectures, assignments and the lab, there were also a midterm and a final exam. However, these were designed relatively simple and both did not include any robot-specific questions.

Table 2: Complete syllabus of our Systems Programming class.

Nr	Lecture Topics	Hands-on Assignments
1	Systems Programming, C and Unix in the historical context. Terminal, text editors and IDEs. Basic types, literals, variables, includes. Unix manual (man) pages.	*In-class:* Installing compiler and "Hello world" program. *Homework:* Modify existing program. Find out lengths of basic data types.
2	Control structures, boolean expressions, minimal I/O.	*In-class:* 7 bit to 8 bit encoder and decoder.
3	Declaring and implementing functions (names, parameters, return values). Call by value and addresses (pointers).	*In-class:* Implement swap function. *Homework:* Implement a parser as finite-state machines.
4	Arrays and strings. Initialisation, null-termination. Strings as function parameters. Const declaration.	*In-class:* Implement string comparison function. Implement get line function.
5	Characters, character sets, locale, wide characters, multibyte characters.	*In-class:* Applying character / wide character functions. *Homework:* Implement standard character and string functions.
6	Type of error in programs. Finding and fixing errors. Using the debugger. Using assertions.	*In-class:* Debugging a given program.
7	Data structures, enums, typedefs.	*In-class:* Analysing enums in the debugger. Using `structs`. *Homework:* Using `structs` as a data-base.
8	Nested structures and unions. Initialisation of structs. Structs and pointers.	*In-class:* Structure a given program by using structs.
9	Introduction to pointer programming. Memory system. Arrays and pointers. Pointer arithmetic.	*In-class:* Inspecting position of variables in memory. *Homework:* Write a program interpreter.
10	Advanced pointers and multidimensional arrays.	*In-class:* Analyse and debug the differences of array of pointers and array of arrays.
11	Dynamic memory management, `malloc` and `free`.	*In-class:* Modify a program with fixed arrays to a tree-based one with dynamic memory allocation. *Homework:* Write your own tail program.
12	Memory management by ownership of pointer.	*In-class:* Implement a linked list structure with access and management functions.
13	Pointer to functions. Qsort example.	*In-class:* Modify a program to allow flexible sort order *Homework:* Implement a binary tree with unspecific node type.
14	Advanced function pointers. Analysing and simplifying function pointer declarations.	*In-class:* Implement a map function.
15	Preprocessor, compilation unit and linker.	*In-class:* Split a large program into files. *Homework:* Split the binary tree implementation into header, implementation and test file.
16	Organizing source code and using build systems.	*In-class:* Write a makefile for the binary tree project.
	Midterm exam	
17	Files, file handling, streams and dictionaries.	*In-class:* Write your own copy file program.
18	Unix system calls, processes and signal handling.	*In-class:* Handling processes from the terminal. *Homework:* Implement your own system function.
19	Signals, core dumps and debugging.	*In-class:* Debug a core dump.
20	Introduction to the robot and its remote control software.	*In-class:* Install remote control software, test the robot. *Homework:* Write a program to steer the robot in a quadrangle.
21	Pipes for process communication.	*In-class:* Communicate with the robot over a pipe.
22	Using standard and third party libraries. Creating libraries.	*In-class:* Using readline library and pipes to extend binary programs. *Homework:* Create a library from an earlier homework and add documentation (doxygen and man page).
23	Cross compiler and robot programming.	*In-class:* Install cross compiler and upload firmware to the robot.
24	Writing programs for the robot with an API.	*In-class:* Access the LEDs of the robot. *Homework:* Using the proximity sensors of the robot (exploring randomly the environment, avoiding obstacles)
25	Direct hardware access for robot programming.	*In-class:* Access the LEDs of the robot without the API.
26	Microcontroller timer programming and the robot timer API.	*In-class:* Access the LEDs of the robot with the API timer functions.
	Final exam	

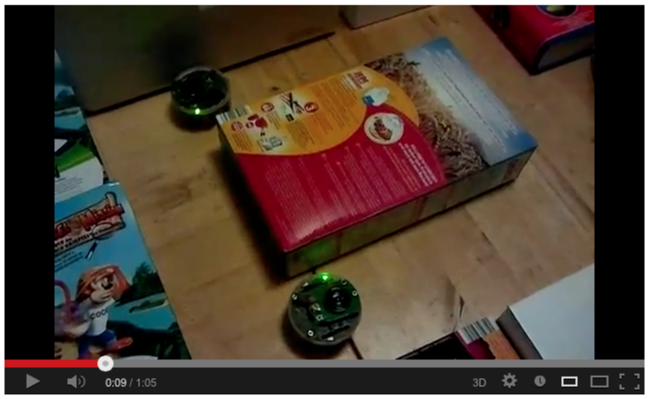


Figure 2: Screen shot of a demonstration of the final homework assignment. Each robot should explore the environment randomly, while it is avoiding obstacles (other robots and walls).

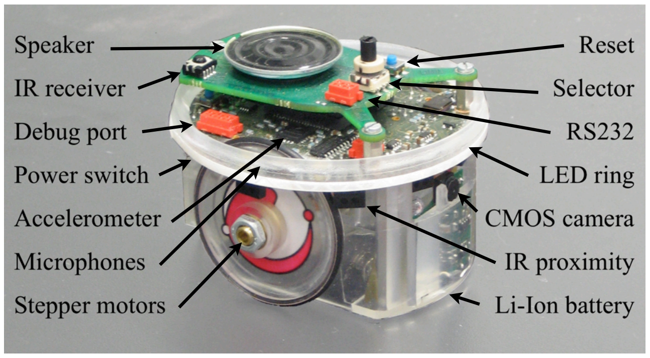


Figure 3: The e-puck robot with its components. It is a small, but complex robot with a lot of components. Additional parts can be added to the robot, e.g., in this picture a loudspeaker is included.

In general, the complexity level of the assignments was higher than the exams'.

4 THE ROBOT

As already explained before, we chose to use mobile robots to show the students some of the applications of Systems Programming for its high attraction factor. However, it is very important to provide "one robot for one student" for a successful hands-on learning experience. Sharing a robot between students usually leaves one student working with it and the others only observing, which has the opposite effect. Another crucial aspect for our specific course was that the robot is programmable in C, supports various levels of programming, supports the operating system which most of the students are using (Mac OS X), and allows easy software uploading without special hardware.

Nowadays, the preferred programming language for many low-price mobile robots is not C, e. g., the Thymio II (http://www.thymio.org) uses Aseba, Arduino (http://www.arduino.cc) based robots use Processing, and other robots are mainly controlled by a personal computer (iRacer http://www.sparkfun.com) or smartphone (Romo http://romotive.com, Wheelphone http://www.wheelphone.com). These robots have often an open source based firmware and an adequate description of the used hardware. Programming these robots in C is in generally possible, but not common and thus not trivial.

We have around 25 e-puck robot at our institute, which were enough for the amount of students in our Systems Programming class. The e-puck robot is a small-size mobile robot developed at EPFL, Switzerland, for educational and research purposes [7]. The robot is also available in various simulators, e. g. Webots [3]. The robot is available in Europe from GCtronic (http://www.gctronic.com/), in Japan from AAI Japan (http://www.aai.jp/) and in North America from AAI Canada (http://www.aai.ca/). The robot is following completely the open source and open hardware paradigm. More information and documentation can be found at its main web site at http://www.e-puck.org/. The page contains the complete PCB layout, BOM, links to the software repository and developer mailing list, where user questions are usually answered quickly by the community.

The robot is equipped with a lot of sensors to gather feedback from the environment. It can also communicate with a host computer or other (e-puck) robots (see Figure 3). Additionally, it is possible to plug-in specialised modules on top of the robot to extend its capabilities. The Gumstix Overo COM (https://www.gumstix.com) adaptor board, for example, allows to install a complete Linux system on the e-puck. During the Systems Programming course we used the accelerometer, microphones and infra-red proximity sensors to collect environmental information, while the LEDs and motors were used to interact with the environment. The runtime of the robot on a single charge is approximately four hours before the battery has to be exchanged or recharged.

The default e-puck robot firmware allows to control the robot remotely via an IR remote control and via Bluetooth, using the serial port profile (SSP). The latter allows to communicate with the robot with a standard terminal program or any programming language which supports Bluetooth/TTY communication. The boot loader of the e-puck robot also permits firmware updates via Bluetooth with a small upload program, available for download for the major operating systems from the website. The standard firmware is open source, well documented and easy to understand. In our experience a student was able to compile and upload his/her own modified firmware within only lesson. The GNU based cross-compiler tool chain MPLABÂő XC 16, which provides the capability to develop code for the robot on a standard PC, is available for download from the company's web site (http://www.microchip.com/).

5 STUDENT SATISFACTION AND EVALUATION

Evaluation of our approach is not trivial in our context. The main problems are of bureaucratic nature, since full evaluation of courses and the grades of the students are only available to the teacher. However, we took over the course only in 2011 and introduced immediately the mobile robots, which renders it impossible to compare the performance of the students numerically. Furthermore, the course's volume was increased at the same time from 3 ECTS to 6 ECTS and new topics were introduced.

However, we believe motivation and excitement of the students is the first step to learning success. Thus, here we will present a qualitative evaluation of the course from the students, trusting that this impacts significantly their willingness to invest time and effort into the course and its learning objectives.

After introducing the projects into the old course a clear increase of participation and motivation within the students was visible (see Section 2). During the project phase the students working with the robots were the most motivated ones and invested most of their free time to bring the project forward and beyond the proposed project goal. Also in discussions after the project the robotics project students were the most excited ones, rating the whole course experience very high.

The first feedback from the new Systems Programming course in 2011 was quite positive. Of course, in this first year the flow of the lectures was not optimal and some of the assignments were not prepared in the detail we would have liked, allowing us to improve in the following year. One of the assignments, programming a simple shell terminal, was dropped in the second year, as it turned out to be too difficult and too ambitious for the given timeframe. It was reduced to programming the system function. Furthermore, we have moved the introduction of the robot from Lecture 22 to Lecture 20, thus before introducing third-party libraries. In the third year, we moved the robot even earlier to lecture 18. This is possible, as the first "Hello World" examples for the robot are simple and do not require more experience with libraries than using, e.g., the C standard library. However, although we tried, moving the robot even earlier in the lectures is hardly possible.

5.1 Official cours evaluation

In the new evaluation system in 2012, students had to fill out an online course evaluation form, before they receive their grades for individual courses. In these evaluations the Systems Programming course ranked 4th out of the 19 bachelor courses, including highly-liked courses, such as, web/game programming projects (see Figure 4). In 2013, it ranked even 3rd. The third rank is clearly an improvement for a course whose main content was seen as an old, outdated programming language class with historic restrictions, pitfalls and without modern concepts like object orientation, dynamic memory management, or integrated programming environments.

The ranking of all courses was based on the mean calculated for eight questions asked in the evaluation. The students had to give grades for questions, such as "The teaching material helped me learn this course." and "The teacher was effective in showing how the subject matter is relevant to everyday life.", based on how much they agreed or disagreed with the statements. Additionally to numerical grades in the questionnaire for the course evaluation the students had the opportunity to give further comments on the course. Unfortunately, not many students used this chance and left the answer blank. Here are two full responses we received:

> Lectures were interesting and quite entertaining at the same time! The homework assignments were also very interesting. In-class assignments were helpful. Both the professor and the TA were very nice, helpful and explained everything clearly.
>
> I really liked working with the robot. I would simply suggest improving a bit on the organisation such as having the homework instructions available on icorsi [http://www.icorsi.ch, the online course platform used in the university; note from the authors] separately from the slides. And be a bit more clear about the program's structure and during lecture assignments. (At some point it felt like everyone was lost for a minute.)

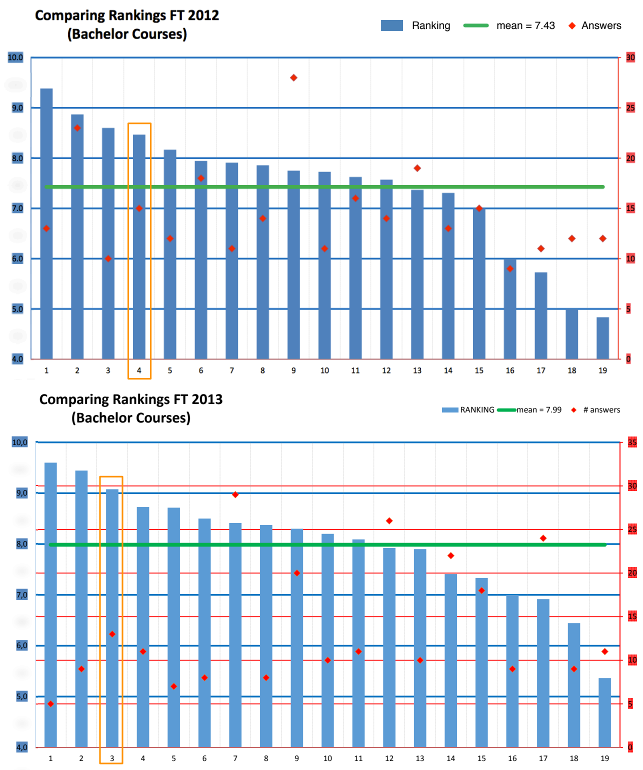
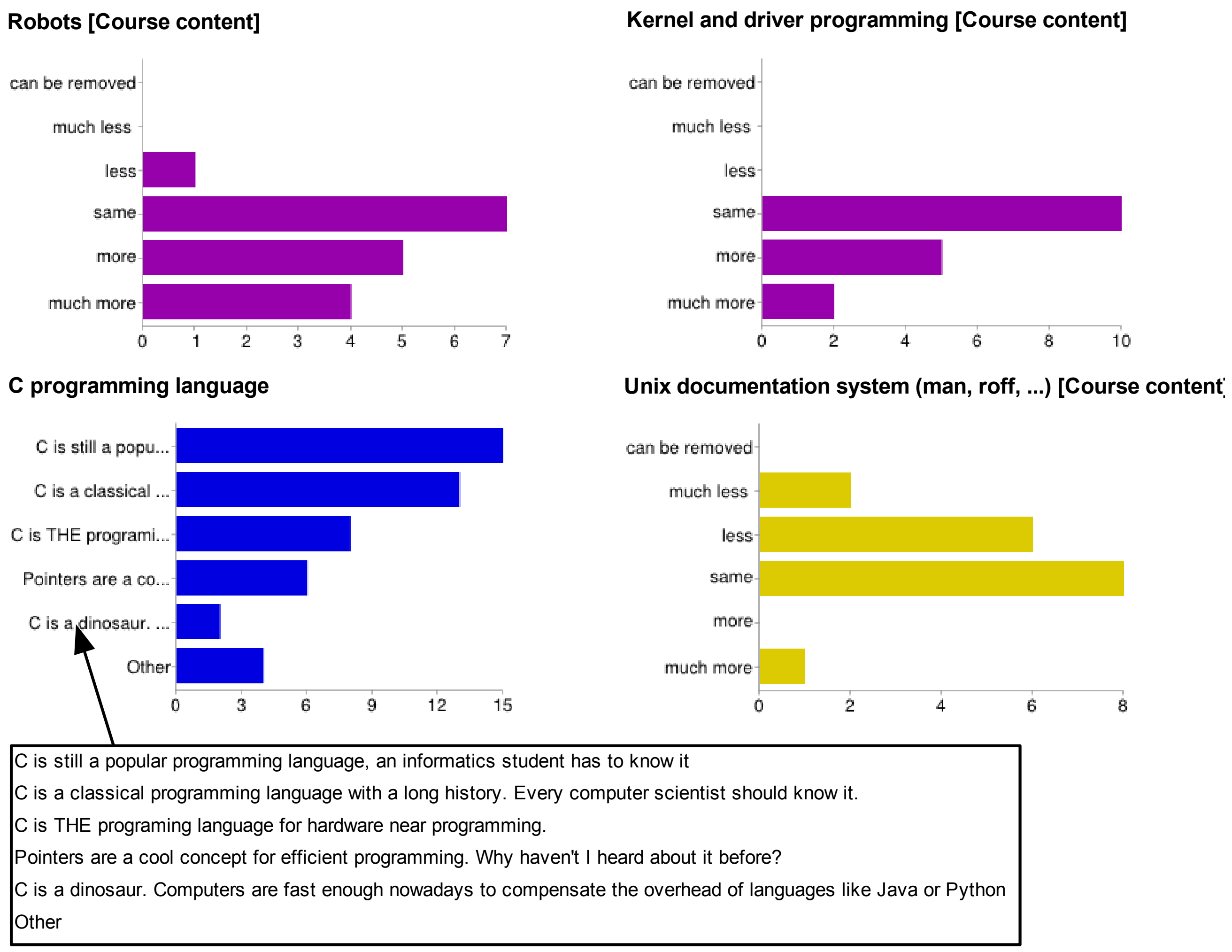


Figure 4: Ranking of the bachelor courses held during the fall 2012 (top) and fall 2013 (bottom) semester. The Systems Programming course was ranked 4th resp. 3rd best by the students.

In general, we are very satisfied with the evaluation from the students and we will further improve individual assignments to better meet their interests and needs.

5.2 Internal course survey

Additionally to the official evaluation of the course, we also asked the students some more content-related questions, which the official evaluation cannot cover. We were mainly interested in the view of the students, which parts they found most useful and which they believed are not necessary. We used these answers to either optimise the presentation of some topics, which the students did



C is still a popular programming language, an informatics student has to know it
C is a classical programming language with a long history. Every computer scientist should know it.
C is THE programing language for hardware near programming.
Pointers are a cool concept for efficient programming. Why haven't I heard about it before?
C is a dinosaur. Computers are fast enough nowadays to compensate the overhead of languages like Java or Python
Other

Figure 5: Some of the answers given by students to our course internal survey in 2012.

not like or to increase the volume in other topics. Sometimes we also removed a topic, which was introduced purely as an example, and substituted it with another.

The survey was performed with all students who have attended the class in fall 2011 and fall 2012. Figure 5 gives some of the answers of the students, which we considered most useful for the course design. After examining these results, we decided to remove some of the Unix documentation system topic and to motivate it better, as well as to increase the kernel and driver related examples.

6 CONCLUSION

In this paper we presented our experience in teaching a Systems Programming class in the undergraduate level. We showed that the students motivation to learn C was improved by using a robotic platform. Robots present a playful environment for the students, but at the same time represent also one of the most challenging real-life hardware systems. Our approach has stimulated the motivation and enthusiasm of the students, which was otherwise low in comparison with other courses, such as web, graphical or game programming. In this paper we described the overall curriculum and the syllabus of the course itself, as well as the used robotic platforms, e-puck and thymio. Furthermore, we explained in detail the design of all assignments (robotics oriented and traditional), so that other course lecturers can re-use our approach easily. Our felt success to provide a more interesting course to the students was confirmed by the end-of-semester evaluation. In those the students ranked our course 4th resp. 3rd from all bachelor-level courses. This has motivated us to share the experience in the firm belief, that a similar approach can be adopted in other Systems Programming (or related) courses to increase the learning experience of the students and thus also their knowledge about programming when leaving academia. At the same time it guarantees that future IT experts are still in possession of Systems Programming skills.

ACKNOWLEDGEMENTS

Thanks to all the students and all the previous lecturers of the course, who gradually developed the in-class exercises, homework assignments and lecture slides.

REFERENCES

[1] M. Baar, H. Will, B. Blywis, T. Hillebrandt, A. Liers, G. Witternburg, and J. Schiller. The scatterweb msb-a2 platform for wireless sensor networks. Technical Report B-08-15, Free University of Berlin, 2008.

[2] A. Förster and M. Jazayeri. Hands-on approach to teaching wireless sensor networks at the undergraduate level. In *Proceedings of the 15th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, Bilkent, Turkey, 2010.

[3] L. Guyot, N. Heiniger, O. Michel, and F. Rohrer. Teaching robotics with an open curriculum based on the e-puck robot, simulations and competitions. In *Proceedings of the International Robotics in Education Conference (RiE)*, 2011.

[4] A. Hoover. *System Programming with C and Unix.* Addison-Wesley, 2009.

[5] M. Jazayeri. The education of a software engineer. In *Proceedings of the 19th IEEE international conference on Automated software engineering (ASE)*, pages .18–xxvii, Washington, DC, USA, 2004.

[6] B. W. Kernighan and D. M. Ritchie. *The C Programming Language.* Prentice Hall, 1978.

[7] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, 2009.

[8] U. Müller. Brainfuck 2, 1993.

[9] R. Pike. Another go at language design.

[10] G. Samaras. Robots avoiding obstacles (e-puck) /samaras@usi. online, 2009.

[11] Faculty of Informatics USI. Bachelor curricculum, systems programming, 2012.