A Multi-Robot Search Using LEGO Mindstorms – An Embedded Software Design Project

Paula Herber
University of Potsdam
August-Bebel-Str. 89, 14482 Potsdam
Potsdam, Germany
paula.herber@uni-potsdam.de

Verena Klös
Technische Universität Berlin
Ernst-Reuter-Platz 7, 10587 Berlin
Berlin, Germany
verena.kloes@tu-berlin.de

ABSTRACT

Embedded software is concurrent, real-time dependent, typically networked, must meet strict resource and high quality requirements, and often runs on cheap hardware. Altogether, this makes the education of embedded software designers a difficult challenge. In this paper, we present an embedded software design project, where students have to develop a multi-robot search using Lego mindstorms. The main idea is to confront the students with all the spites that are typically present in embedded systems, while at the same time giving them an algorithmically non-trivial problem to solve. To this end, we let the students use a bio-inspired search algorithm (particle-swarm optimization) to detect survivors (led by cries for help) in an unknown disaster zone using a number of Lego Mindstorm robots. We have executed this project simultaneously at the University of Potsdam and TU Berlin and discuss results and evaluations. We think that this project is very well suited for the education of embedded software engineers.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-purpose and application-based systems—*Real-time and Embedded Systems*; K.3.2 [**Computing Milieux**]: Computers and Education—*Computer and Information Science Education*

General Terms

Design

Keywords

Education, Embedded Systems, Multi-Robot Search

1. INTRODUCTION

The amount of embedded software is steadily increasing. In [4], the authors state that the annual growth of the volume of embedded software varies between 10 and 20 %. Embedded software engineers have to cope with this growing complexity. At the same time, the systems are often networked and heavily interconnected. They typically consist of simple and often imperfect hardware, which imposes strict resource constraints, and, due to the interaction with the physical environment, the software must be designed to cope with real-time and concurrency. To meet all the requirements, embedded software designers need to know their hardware, and they need to cope with both functional and non-functional requirements.

The steadily increasing complexity of embedded software together with the multi-demands in interdisciplinary applications makes the education of embedded software engineers a difficult challenge. Embedded software engineers need to be able to apply software engineering methods, they need to know how to cope with the perils of cheap and unreliable hardware, and they need to be able to work together with experts from various application fields (and/or become experts in the application field themselves).

In this paper, we propose an embedded software design project, which we have designed to target all of the issues mentioned above. We are confident that this project can provide a valuable contribution to the education of embedded software engineers. The main idea is to confront the students with all the spites that are typically present in embedded systems (like imprecise sensors, limited resources and concurrency) and to give them an algorithmically non-trivial problem to solve. We let the students use a bio-inspired search algorithm (particle-swarm optimization) to explore an unknown disaster zone and to detect survivors (modeled as a non-linear sound source) using a number of Lego Mindstorm robots. Among the requirements is collision avoidance, the ability to cope with an unstable bluetooth connection, and the demand for an emergency mode where all robots have to return to their bases on the shortest possible path.

We have executed this project simultaneously at the University of Potsdam and TU Berlin and discuss results and evaluations. Our project is designed for students in the final year of their Bachelor or in the first year of their Master. A prerequisite for the successful execution of this project is that the students have already completed some background courses on software engineering and on embedded systems. They should have some knowledge about typical charac-

teristics of embedded systems, like concurrency, real-time, limited resources, and non-functional requirements. They should also have learned and practiced (formal) specification and modeling techniques for embedded systems, e.g. timed automata or Statecharts, and temporal logics. They should be familiar with programming techniques for concurrent systems (tasks and processes, inter-process communication, scheduling, shared resources), and, finally, they should have some experience in C programming.

We start the semester with a few introductory lectures and exercises to bring the students to the same level of knowledge and to equip them with the necessary basic knowledge on embedded systems, the employed software and project management. The project work itself starts two weeks after the beginning of the semester. Then, the weekly lectures are replaced by oral milestone presentations about the progress in each project group every three weeks. The semester is closed with a final presentation at the end of the semester and oral exams, where the students explain and defend their contributions to the project. The overall project is worth 9 credit points (ETCS).

The remainder of this paper is structured as follows: first, we give some background information on Lego Mindstorms, the real-time operating system nxtOSEK, UPPAAL timed automata, and particle swarm optimization. Then, we present our embedded software design project, with a focus on the project task and project organization and management. In Section 4, we discuss the technical challenges and results of the project. We briefly discuss our evaluation of the project in Section 5, and conclude in Section 6.

2. PRELIMINARIES

In this section, we introduce the preliminaries that are necessary to understand the remainder of this paper. We first give a brief overview over the Lego Mindstorms hardware and the real-time operating system nxtOSEK for the Lego Mindstorms programmable NXT. Then, we briefly introduce UPPAAL timed automata, a formal modeling language for real-time systems. Finally, we provide a short description of the general ideas of particle swarm optimization (PSO).

2.1 Lego Mindstorms

The Lego Mindstorms NXT 2.0 kit was released by Lego 2009 and consists of a set of Lego components to build customizable, programmable robots. It includes a set of modular sensors and actuators, in particular 3 servo motors, and ultrasonic, sound, touch, and light sensors. The main component is the NXT intelligent brick, which features a 32-bit Atmel main microcontroller with 256 KB flash memory and 64 KB RAM, a 100x64 pixel LCD screen, four sensor and 3 actuator ports, a USB port and Bluetooth V2.0. The NXT intelligent brick together with its sensors and actuators is shown in Figure 1.

Lego Mindstorms have been shown to excite a high level of interest among students [19] and have been used in many courses ranging from basic computer engineering courses [9, 10, 12] to more advanced control systems [7, 5]. In our setting, they provide an ideal target architecture, as they are cheap (approx. 300 $ per education kit), easy to build, and provide modular hardware including sensors and actuators. At the same time, they can be equipped with a small real-time operating system (e.g. nxtOSEK), have limited resources (memory, energy, processing power), and the hardware is neither reliable nor precise. Finally, the Lego Mindstorms NXT is programmable in many programming languages, ranging from various C dialects over high-level languages like Python, Ada, and Java to graphical languages like MATLAB Simulink. In our project, we chose a concurrent version of classical C as this is still the most widely spread language in embedded systems.

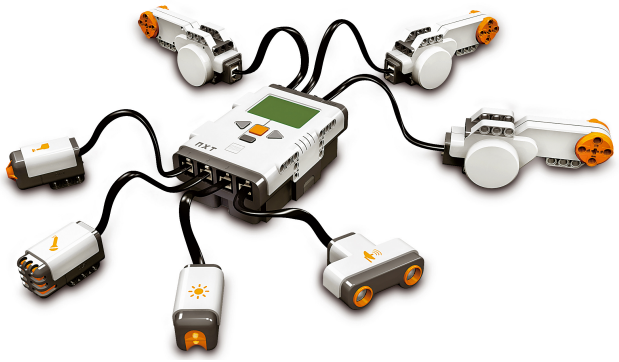

Figure 1: Lego Mindstorm NXT

2.2 nxtOSEK

The real-time operating system nxtOSEK [13] consists of two main components: a device API (leJOS [11]) that enables convenient access for NXT sensors, actuators and other external devices, and a standard software architecture for embedded operating systems in automotive systems, TOPPERS/ATK OSEK [18], which provides real-time multitasking features according to the OSEK standard. nxtOSEK is focused on real-time control applications and provides preemptive periodical and event-driven task scheduling.

2.3 UPPAAL Timed Automata

Timed automata (TA) [1] are finite-state machines extended with clocks, where clock conditions are used to model time-dependent behavior. A TA contains a set of locations connected by directed edges. Two types of clock constraints are used to model time-dependent behavior: *Invariants* are assigned to locations and enforce progress by restricting the time the automaton can stay in this location. *Guards* are assigned to edges and enable progress only if they evaluate to true. Networks of TA are used to model concurrent processes, which are executed with an interleaving semantics and synchronize on channels. UPPAAL [2] is a tool suite for modeling, simulation, and verification of TA. The UPPAAL modeling language extends TA by bounded integer variables, binary and broadcast channels, and urgent and committed locations. Binary channels enable a blocking synchronization between two processes, whereas broadcast channels enable non-blocking synchronization between one sender and arbitrarily many receivers. Urgent and committed locations are used to model locations where no time may pass. Furthermore, leaving a committed location has priority over non-committed locations. The formal semantics of UPPAAL timed automata (UTA) is given in [2].

UPPAAL timed automata can be verified using the UPPAAL model checker, which supports requirements specifications that are defined in a subset of the computation tree

logic CTL. The UPPAAL model checker explores all pathes of a given timed automata model to check whether a given formula is true. As all state-of-the-art model checking tools, it also provides a counter-example if a property is not satisfied on all pathes. This counter-example can also be used to demonstrate the reachability of a certain path. In particular, UPPAAL also provides a *shortest path* option, which can be used to compute the shortest possible path that witnesses the reachability of a certain property. In [17], the authors have shown how UPPAAL and the *shortest path* option can be used to compute the shortest pathes for a number of mobile robots on a given map with static obstacles.

2.4 Particle Swarm Optimization

In [14, 15], Pugh and Martinoli presented a concept to solve the multi-robot search problem by using particle swarm optimization. Particle swarm optimization (PSO) as an optimization technique was developed by Kennedy and Eberhart [8]. The main idea is to model a set of potential solutions for a given problem as a swarm of particles searching in a virtual space for good solutions. The method was inspired by the movement of flocking birds and their interactions with their neighbors.

The PSO algorithm works as follows [14]: Every particle in the swarm begins with a random position x_i and (possibly) randomized velocity v_i in an n-dimensional search space, where $x_{i,j}$ represents the location of particle i in the j-th dimension of the search space. Each particle remembers at which position it achieved the best result so far $x^*_{i,j}$, and which particle achieved the best overall position in its neighborhood $x^*_{i',j}$. Then, at each step, a PSO algorithm executes the following equations:

$$\begin{aligned} v_{i,j} &= w \cdot v_{i,j} + w_p \cdot \mathit{rand()} \cdot (x^*_{i,j} - x_{i,j}) \\ &\quad + w_n \cdot \mathit{rand()} \cdot (x^*_{i',j} - x_{i,j}) \\ x_{i,j} &= x_{i,j} + v_{i,j} \end{aligned}$$

Where w_p is the weight given to the previous best location of the current particle and w_n is the weight given to the previous best location of the particle neighborhood, and *rand()* yields a uniformly-distributed random value in $[0, 1]$. The PSO-inspired multi-robot search presented in [14, 15] uses a one-to-one matching between particles in the PSO swarm and robots in the multi-robot system.

3. EMBEDDED SOFTWARE DESIGN PROJECT

The aim of our embedded software design project is to teach the students to work together in a team on a complex embedded software design. Besides the embedded software design task itself, our goal is that they gain experience in project management and organization. In particular, they should learn how to divide a comparatively large design task into smaller tasks, and how to communicate between multiple teams that are working on subsystems, which are later integrated into one system. The system integration is one of the main challenges in embedded system design. This has several reasons:

- Subsystems are designed by experts of various domains. The behavior of one subsystem is only partially understood by the designers of another subsystem.
- All subsystems are *concurrent*. Thus, the overall system behavior is hard to predict.
- Processing and memory resources are limited. Resource access must be managed, and the overall resource consumption must be controlled.
- Some processes are real-time dependent. Their correct timing behavior must still be ensured if their execution is interleaved with other processes.
- All sensors, actuators, and communication devices must be assumed to be unreliable and inaccurate. As a consequence, subsystems can not always rely on results or answers from other subsystems.

In the following subsections, we will first define the project task and briefly summarize the corresponding requirements definition, which we have given to the students. Then, we will briefly present our technical equipment and support. After that, we will discuss our project organization and project management. In the next section (Section 4), we will present some more details about the project execution itself, including exemplified design decisions, to give an impression of the technical challenges and detailed content of the project work.

3.1 Project Task

The aim of the project is to implement a multi-robot search in an unknown environment using a number of Lego Mindstorms. The goal of the search is a non-linear sound source. The motivating real-world application is the detection and localization of survivors in a disaster zone, led by cries for help. In such scenarios, the use of autonomous mobile robots renders the use of human search personal unnecessary and thus may save lives and cost, in particular if the disaster zone is still considered unsafe, as is the case in the aftermath of earthquakes or flooding. A swarm of simple and cheap robots has the further advantage that it is tolerant against failures of a number of single robots, can explore a given area quickly and scales well for larger areas.

Project Goal.

The aim of the project is to design a swarm of at least three mobile robots, which localize a sound source using sound sensors. All movements must be planned and carried out by local rules on each robot, e.g. by particle swarm optimization as presented in [14, 15]. The swarm may communicate via bluetooth, but has to cope with unstable and temporarily unavailable bluetooth connections. Collisions must be avoided at all times and in all cases. At any time, a central control element may recall all robots to their base stations using an emergency mode. In this mode, the shortest path to return all robots to their base station is centrally computed and communicated to each robot, for example by using the UPPAAL based approach presented in [17]. In the emergency mode, sound and ultrasonic sensors are switched off, so it is important that the algorithm to compute the pathes back to the base station for each robot is provably collision free. This can be ensured by using the UPPAAL model checker for the computation of the shortest path. If the sound source is detected, the robots should indicate this

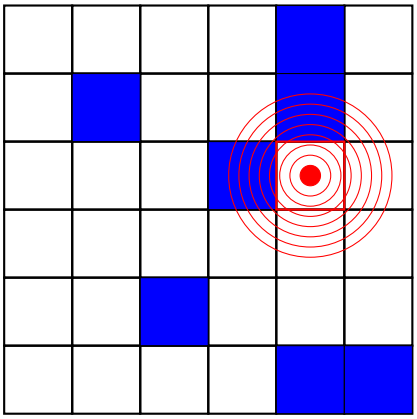

Figure 2: 2D search space

with a sound signal and broadcast the information to all other swarm elements and to the central control.

Simplifications.

To enable our students to solve the project task within one semester, we use the following simplifications of this scenario:

- The robots move in a 2-dimensional search space, as shown in Figure 2.
- The search space is rectangular and its size is known.
- The search space is marked by a (physically detectable) grid, which enables orientation and positioning of the robots.
- The location of the sound source is as also marked on the grid to simplify the physical identification of the goal.
- Obstacles are cubes which have the same edge length as one field in the grid and are aligned to the grid.
- A PC may be used as a hub for the communication between the robots, and to build and distribute a common map from the information collected by all robots.

Additional Requirements.

Besides the project goals defined above, we give the students the following additional requirements:

- The hardware of the robots is identical for each group and is developed by all groups together.
- All robots of one group run the same software.
- All robots start at a defined base station.
- The robots must never collide with each other or with obstacles.
- The resulting map should be shown on a PC.
- The performance of each solution is measured by the time needed to find the sound source and by the quality and robustness of the solution.
- Each solution must cope with unstable bluetooth connections.

Note that we give the complete project goal and all requirements to the students at the beginning of the semester. No requirements are changed during the semester, as this would go beyond the scope of one semester.

3.2 Technical Equipment and Support

For the execution of this project, it is most advantageous to have a project room that is solely dedicated for the use of the project during the whole semester. In our case, the project rooms are equipped with a number of PCs running Ubuntu where we pre-installed the following software:

- The GNU ARM cross compiler from the Gnu Compiler Collection (GCC) [6], which supports the ARM7 CPU inside the NXT.
- John Hansen's NeXTTool from the Programmable Brick Utilities repository [3], which can be used to communicate with the NXTs via USB & Bluetooth (and thus also enables program upload).
- The nxtOSEK real-time operating system, which can be compiled together with user-defined C or C++ code into nxtOSEK applications for NXTs.

As building equipment for the mobile robots, we provide our students with five boxes of the *9797 LEGO© Mindstorms© Education Base Set* together with five boxes of the *9695 LEGO© Mindstorms© Education Resource Set*, plus five additional lightsensors.

As a project management platform, we have set up a Redmine server together with a subversion (SVN) repository. The former is used by the students as a project planning tool, ticketing system, and online platform for information exchange (discussion forums, document upload). The latter is used as a versioning and revision control system for shared development of the code base developed in the project.

3.3 Project Organization

The challenges of system integration in embedded software design projects can only be met if the overall project is properly managed. In students' projects, the students have to self-manage their project work, typically in a non-hierarchical fashion.[1] In our embedded software design project, we build groups of 8-9 students. Each group has to name persons that are responsible for:

1. requirements, change, and risk management (definition and evolution of requirements, identification and communication of changes and risks)
2. project planning (work breakdown structure, scheduling, milestones)
3. technical project management (interface definition, interprocess communication, communication between teams working on different subsystems, system integration)
4. quality assurance (coding standards, review strategies, verification and test plans)

[1]A hierarchical project management would not have the same learning outcome for all the participants, and is difficult to deploy within groups of students, for many reasons.

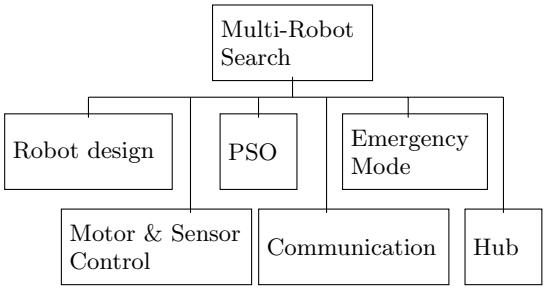

Figure 3: Work Breakdown Structure

It is important that each student assumes one of these responsibilities. This ensures that all of the participants take part in the project management. Furthermore, it results in the fact that 2(-3) students are responsible for each role, introducing yet another type of teamwork in the project. The students that are assigned with the above responsibilities are in charge of their role during the whole project, in addition to the later assigned design tasks. The definition of the roles is important to make sure that the project groups self-organize their work and that there are cleary defined contact persons for each problem scope.

3.4 Project Management

Besides the experience in embedded software design, the students should also gain experience in project management and organization. In large and interdisciplinary projects it is important to divide a large design task into smaller tasks, and to communicate between multiple teams that are working on subsystems, which are later integrated into one system. Therefore, the students have to develop their own project plan consisting of a work breakdown structure, deliverables for four predefined milestones and a gantt chart which shows the project schedule, the schedule of human resources and the current progress. Depending on their project plan they divide their group into teams and define work packages and interfaces for each team. Besides a weekly meeting with their supervisor the groups have to organize meetings and communication themselves. To ease communication and version control we provide each team with a group forum *(moodle)*, an issue tracking system *(redmine, gitlab)* and a version control system *(svn, git)*.

In their kick-off meeting, all project groups had to define a work breakdown structure. The work breakdown structure was then used to define work packages and assign teams of 2-3 students to each of them. An exemplary work breakdown structure is shown in Figure 3. The work breakdown structure is the basis for the project plan and scheduling in each group, including milestone definitions. An exemplary milestone definition is shown in Table 1.

4. PROJECT EXECUTION

In this section, we give details on the main components of our embedded software design project. We present some important design decisions and discuss the main challenges and technical difficulties the students had to cope with.

4.1 Emergency Mode

In the emergency mode the robots have to return to their base station on the shortest possible path. To calculate this path the students used the UPPAAL based approach presented in [17]. There, the authors propose to model the environment, the robots and a control for each robot as UPPAAL Timed Automata. The robots can move horizontal and vertical in a Cartesian grid and have to reach their goal positions while not colliding with each other or obstacles. The UPPAAL verifier is used to coordinate the robots and to calculate a collision free path for all robots. This is done by calculating the *shortest path* that witnesses the reachability property

$$E <> robot1.atBase \;\&\&\; robot2.atBase \;\&\&\; robot3.atBase$$

which states that all robots are at their base position, as described in Subsection 2.3. The resulting path is then filtered for the necessary movements of each robot and distributed to the NXTs.

The authors evaluated their approach with a grid of 5x5. In our setting we experimented with grid sizes of 6x6 and 6x9. The experimental setup is shown in Figure 4. As these grid sizes increased the state space for the verifier drastically and as we required the calculation of the path back to the base station to be quite fast, the students had to optimize the UPPAAL model.

Some example modifications are listed below:

- Remove all clocks and use discrete steps instead. This reduces the state space drastically but requires a synchronized movement of the robots.
- Combine movements and turns or two consecutive turns to one step to decreases the depth of the search space. This is possible, as the robots can turn much faster than they can move.
- Remove unnecessary moves like turning towards an obstacle or turning three times on the same field.
- Introduce a sequential control of the three robots to reduce interleavings.
- Move simple calculations to native functions.

With those optimizations our students where able to handle bigger grids and to reduce the calculation time drastically.

Milestone	Deliverables
MS 1 05/05	Project plan (incl. responsibilities & schedule) PSO paper read and understood UPPAAL paper read and understood Prototypical robot design Experimental setup Evaluation of sensor precision and reliability
MS 2 05/26	PSO simulation UPPAAL model Motor control: move forward, rotate Bluetooth: basic setup
MS 3 06/16	PSO implementation UPPAAL interface & hub Emergency mode Collision avoidance via ultrasonic sensor
MS 4 07/07	System integration System testing

Table 1: Milestones

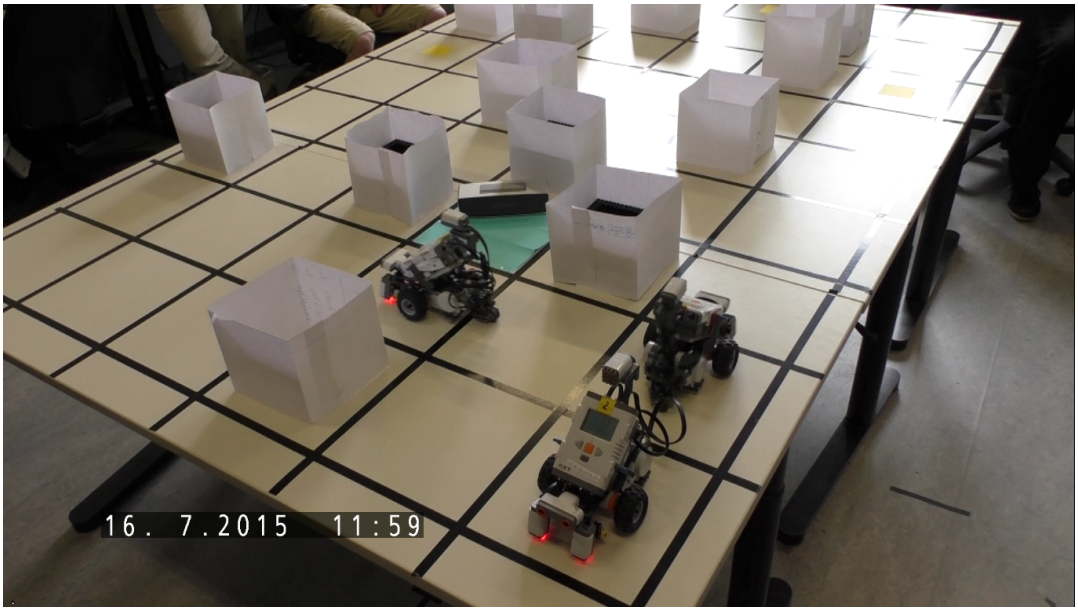

Figure 4: Experimental setup

4.2 Particle Swarm Optimization

As basis for the local planning of movements the students used the PSO algorithm from [14, 15] as explained in Subsection 2.4. This algorithm can be adjusted and customized for the project setting to optimize the performance of the swarm. In our setting robots are only allowed to move up, down, left or right, but not diagonal on the grid. Thus, the actual movement has to be in the direction of the maximum of the x and y component of the velocity vector. An important adjustment concerns the target function. In our setting the main task is to find the sound source. Therefore, the measured sound value on a field is used to calculate the personal and global best position for the robots. As the sound is varying and as the sound sensor is not precise this measurement can only be used as an estimate for the location of the target. The possibility to physically detect the target field allows for a further adjustment of the target function. As long as the position of the sound source is unknown, it has to be located on an unexplored field. Thus our students adjusted the velocity vector to prefer unexplored fields. To achieve this, the calculated vector can be shifted towards an unexplored field nearby or clusters of unexplored fields can be considered during the calculation by adding a personal and global best position concerning unexplored fields. The following formula shows such an adjusted calculation. The personal best position b_p^* consists of the personal best position related to the sound ($s_{i,j}^*$) and the best position related to map information ($m_{i,j}^*$). Each part can be adjusted by a weight ($w_{p,s}$, $w_{n,s}$). The best position in the neighborhood b_n^* is calculated in the same manner.

$$\begin{aligned} v_{i,j} &= w \cdot v_{i,j} + w_p \cdot rand() \cdot (b_p^* - x_{i,j}) \\ &\quad + w_n \cdot rand() \cdot (b_n^* - x_{i,j}) \\ x_{i,j} &= x_{i,j} + v_{i,j} \\ b_p^* &= w_{p,s} \cdot s_{i,j}^* + w_{p,m} \cdot m_{i,j}^* \\ b_n^* &= w_{n,s} \cdot s_{i',j}^* + w_{n,m} \cdot m_{i',j}^* \end{aligned}$$

To enable early evaluation and testing of the PSO algorithm without depending on the progress in the other components, a PSO simulation environment is needed. This can be combined with the task to display the resulting map on a PC. Figure 5 shows a particular nice solution which was written in python by using the library pygame [16] which is a set of python modules for video games. This simulator allows for the initialization of the grid, a real time visualization of map informations (base, robot position, detected obstacles, unknown fields, volume values, calculated PSO-vector, detected dead ends) as well as switching between the modes (search and emergency). It also includes a visualization of the calculated UPPAAL paths (incl. calculation time & number of steps of longest path).

4.3 Embedded Software Design

The real embedded software design part of our project consists of the components that finally run on the NXTs themselves. These comprise the motor and sensor control, and a bluetooth communication layer. An important issue was the system integration, namely to smoothly run all tasks together on one NXT. Some groups had to reduce the number of tasks in order to meet the strict memory constraints imposed by the Lego Mindstorm architectures. All groups had to carefully design the timing of their tasks by using preemptive periodic and event-driven tasks.

An exemplary task structure is shown in Figure 6. All sensor tasks are periodic (indicated by a circular node), the motor control task and the main task are event-triggered (indicated by rectangular nodes). The tasks exchange events (E) and data (D). The motor stops (E0) in case of a detected light change (E1) or obstacle (E2). It is controlled by events from the *MainTask* (E3: one of *adjust*, *moveF*, *rotateL* or *rotateR*) and corresponding motor control values (D0) and reports whether the move was successful (E4: one of *moveDone*, *moveFailed*). The *MainTask* also sets default values for the light sensor (D1) and exchanges audio and ultrasonic sound signals (D2) with the *SensorTask*.

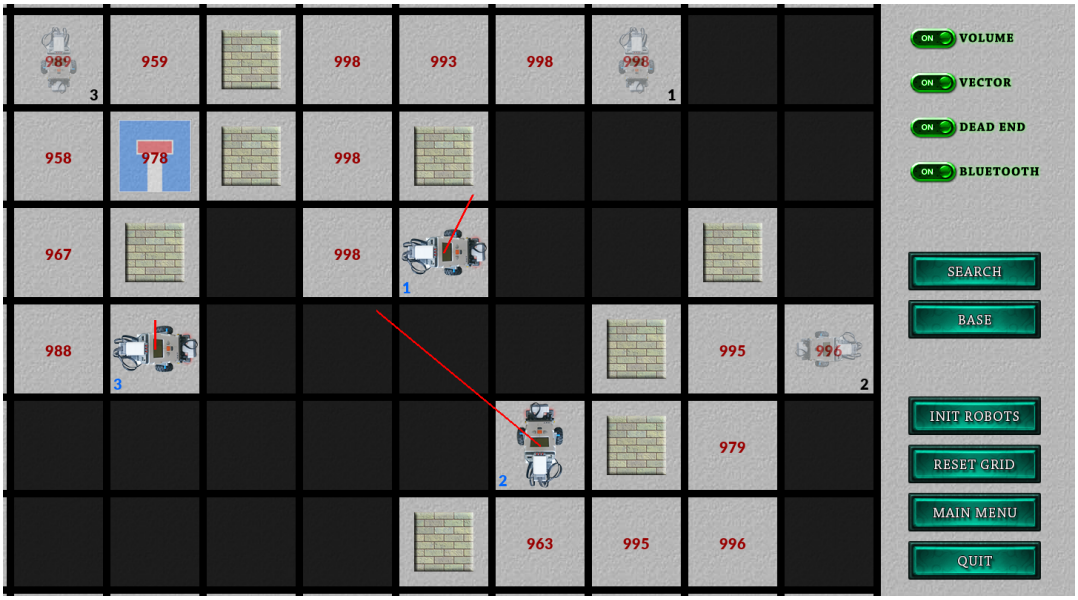

Figure 5: Simulator and Live-GUI

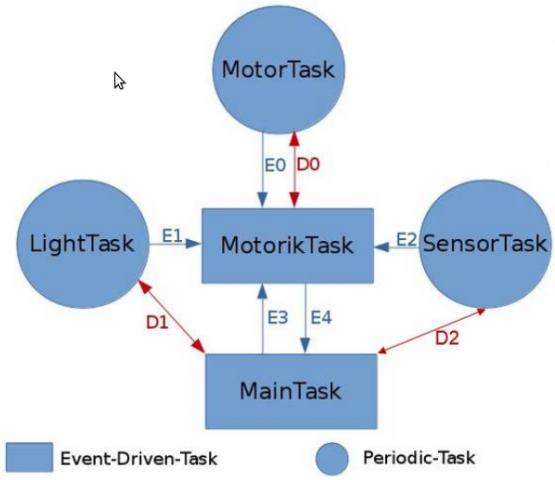

Figure 6: Task structure

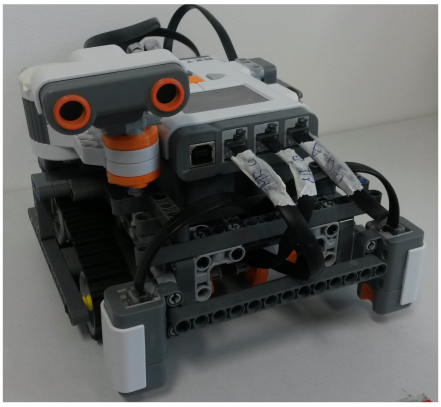

Figure 7: Robot design Potsdam

Note that the *SensorTask*, which reads audio and ultrasonic sound signals, is periodically executed every 25 ms, the *LightTask* every 10 ms, and the *MotorTask* every 20 ms. In the following, we discuss the robot design, the motor and sensor control, and the hub and communication layer.

4.3.1 Robot Design

As mentioned in the introduction, the Lego Mindstorm NXTs only have 3 motor and 4 sensor ports. This constraint is the main limitation for the robot design. As our robots are moving along a grid, they need light sensors to safely detect the edges of the grid fields. All groups decided to use two light sensors, which are parallely deployed at the front. The main advantage of this construction is that the two light sensors can be used to calibrate the course of the robot by aligning both light sensors to a grid edge. As one sensor port is needed for a sound sensor to detect the sound source, this leaves only one sensor port for ultrasonic sound, which is necessary to detect obstacles and avoid collisions. As ultrasonic sound signals are generally unreliable and may be reflected by other sensors or by sloping surfaces, our students decided to make the ultrasonic sound sensor rotatable. By scanning an arch of up to 120 degrees, the probability to detect obstacles and other robots is significantly increased. As gears, the students of TU Berlin decided to use a two-wheel drive and one passive centered wheel for stabilization. The Potsdam students decided to use a chain-drive. An exemplary robot design from Potsdam is shown in Figure 7, an robot design from Berlin in Figure 8,.

4.3.2 Motor and Sensor Control

The aim of the motor control component is to enable movements and rotations of the robots. The sensor control is responsible for collecting and processing sensor data. For the motor control component, the challenge is that two motors that are driven with the same force still produce slightly varying wheel/chain speeds. To cope with that (and ensure

Figure 8: Robot design Berlin

that the robots move straight forward) the students have implemented varying versions of simple proportional control systems up to complete proportional-integral-derivative (PID) controllers. In addition to motor control functions like *nxt_motor_set_speed()*, the nxtOSEK API enables acquisition of the current motor count using *nxt_motor_get_count()*. This can be used in a feedback loop to measure the error and adjust the speed values. In doing so, the robots adjust their speed gradually and reach converging wheel/chain speeds on both sides with very little overshoot, so the result is much smoother than on-off control.

While the motor control component is triggered by the corresponding events to move forward or rotate, all sensors are periodically polled. Sound sensor data is collected and used by the PSO algorithm and communicated to the central hub. The light sensors are used to detect the grid on the experimentation field. Any significant deviation from the standard value is reported and used by the motor control to check whether a movement is finished, or whether the search goal is found. Note that some groups always perform a plausibility check when a grid marker is detected. If it is detected too early, they assume that a side line was hit and start a course correction routine. This significantly increases the robustness of the overall system.

The ultrasonic sound sensor is used to detect obstacles and other robots, and eventually, to avoid collisions. Unfortunately, there exist some situations, where ultrasonic sound signals alone might not be sufficient for collision avoidance. In particular, if two robots are nearing each other in a 90 degree angle and an obstacle is obstructing their view from each other, it cannot be ensured that the ultrasonic sound sensors will detect the other robot before they crash. As a consequence, collisions can only be safely avoided with communication. To this end, each robot acquires a lock on its target field before making any move. If no communication is available (due to temporary bluetooth failure), the robots move sequentially in dedicated time slices, such that no two robots might move at the same time.

4.3.3 *Hub & Communication via Bluetooth*

A central hub is responsible for the communication between the NXTs, collects and merges their sensoric results to establish a common map, and distributes the measured sound information among the robots. It also detects bluetooth failures and tries to reconnect at all times. Furthermore, it takes care of the interactions with UPPAAL and the graphical user interface, including the simulator as described above. In the emergency mode, it calls UPPAAL for the path calculation, filters the resulting trace and passes the commands that establish the path for each robot to the NXTs. Finally, it is also used for debugging purposes.

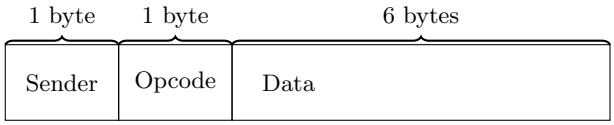

Figure 9: Bluetooth message

The bluetooth connection between the hub and the NXTs is established via a socket mechanism. All groups have used a simple protocol, where an opcode defines which kind of data is sent (e.g., debug map, or sound data). An exemplified structure of a bluetooth message is shown in Fig. 9.

One of the challenges in the design of the bluetooth communication layer was that the bluetooth connections are unstable. As discussed above, it is not possible to avoid collisions without communication if all robots move simultaneously. As a consequence, if the hub looses the connection to one NXT, the whole system has to switch into a safety mode where only one robot moves at a time. To make sure that bluetooth failures are detected and properly handled, the students have used heartbeat signals and an upper time limit on reconnects. If the central hub looses connection to at least one NXT, it switches to a safety mode. The safety mode is only left if a connection to all robots can be reestablished.

4.4 Quality Assurance

As mentioned above, each team had to name two persons responsible for quality assurance. Although the extent of the project task and the strict time limit of one semester left little time for a full quality assurance approach, the students at least defined some basic coding standards, review strategies and rudimentary verification and test plans.

Coding Standards and Documentation.

As basic coding standard, all teams agreed on some naming conventions for variables, methods, tasks, and events. Some teams also defined consistent interface description, commenting, and documentation styles.

Review Strategies.

The review strategies used by the students ranged from ad-hoc partial code reviews to well-organized cross-reviews for each module and each component. The latter worked well in early project phases, but the scheduled reviews were more and more reduced and finally neglected when the pressure of keeping the implementation milestones increased towards later project phases.

Verification and Test.

The students used the UPPAAL model checker to verify

their central planning algorithm, which was already implemented in UPPAAL. Beyond that, no formal verification was applied. However, the students also used the UPPAAL model checker to evaluate the efficiency/ performance of their planning algorithm on several test maps. For testing their PSO algorithms in early project phases (before the mobile robots were functional), all teams implemented simulation environments and simulated the exploration of a number of virtual test maps on a host PC. Some teams already defined a number of test scenarios at this point of the development process and reused them in later project phases and with the final robot implementation. For the main testing phase, the students developed test plans. The test plans defined for each test case their identification number, their type, a short description, necessary preconditions, necessary steps to execute the test case, expected results and the actual results. For an example test definition, see Table 2.

The test cases defined by the students were categorized into unit tests, integration tests, and system tests. The main goal of the unit tests was to validate the basic functionality of each component. Example unit tests for each component are:

- Planning algorithm: find path for given test map
- PSO algorithm: compute next move, compute vector
- Sensors and Actuators: detect line, measure sound, move one field forward, rotate by 90 degrees
- Hub & Communication: establish connection, send and receive data

The main goal of the integration tests was to validate the interplay between connected components. For example, the integration tests for the central hub and the UPPAAL planning algorithm validate that maps that are stored in the central hub are correctly transfered to UPPAAL and that the pathes computed by UPPAAL are correctly passed back to the central hub.

For system testing, the students developed a number of scenarios that might be challenging for the overall system behavior. In particular, they defined a number of test maps and, for each of them, various environment parameters like the day light conditions, battery levels, and background noise levels. During system testing, they also put a particular focus on validating the reliability and robustness of the overall system. For example, they tested the overall system behavior in the following scenarios:

- missing robots (manually removed)
- movement disturbances with varying degrees (manual pushing up to manual field changes)
- bad light conditions
- missing grid lines
- 'caged' robots (surrounded by obstacles)
- global and local bluetooth failures

Overall, the degree to which the students validated the correctness, reliability and robustness of their implementations varied between the five project groups. For example, some groups went immediately from unit to system testing and skipped integration tests, and while some groups performed extensive reliability and robustness tests, others only tested the basic fulfillment of the required functionality.

5. EVALUATION

We have executed this project simultaneously at the University of Potsdam and TU Berlin with 17 and 26 students, respectively (43 students in total). All of the students are either in the final year of their Bachelor or in the first year of their Master. The students where from the following majors: computer science (29), computational science (8), computer engineering (4), industrial engineering (1), mathematics (1).

Out of the 43 participants, we have built 5 groups with 8-9 participants per group. Having such comparatively large groups made the internal management in each group (including the division into work packages and the communication between subgroups) quite a challenge. However, the project is designed in a way that the overall task can adequately be split into smaller subtasks. We believe that the teamwork in a comparatively large group is one of the most important skills that is teached in this project, and our supervision keeps a careful eye on a fair work distribution. Still, during the oral feedback session, most students said they would have preferred smaller group sizes.

Our evaluation of the course has shown that the project was very well accepted. On a scale from 1 to 5 where 1 is very good and 5 is very weak, the project task itself achieved an average rating of 1.25. The supervision of the project achieved an average rating of 1.39. Note that the evaluation results from Potsdam and Berlin were very similar. At TU Berlin, where we have executed the project for the fourth time, we always have more applicants than available places. In Potsdam, we planned to start the project with only one group and a maximum number of 9 participants, but opened a second group due to the many applicants.

In our evaluation, we also asked the students to assess their own learning outcome. On a scale from 1 (I learned very much) to 5 (I learned very little), the average rating of various learning outcomes is shown in Table 3. As can be seen, the most valued learning outcome of the project was teamwork. Furthermore, the students had the feeling that they have learned a considerable amount of embedded software design, algorithm engineering, project management and project organization. Note that the evaluation results from Potsdam and Berlin varied slightly. In particular, it is interesting to note that the students from TU Berlin, which we believe to have a stronger background in foundations of computer engineering, had the feeling they learned more in the soft skill categories project organization, project management and teamwork. On the other hand, the students from Potsdam University, which we believe to have a stronger background in software engineering and project management skills, had the feeling they learned more in the technical categories embedded system design and algorithm engineering. Note also that the standard deviation in each category is pretty high. This is due to the fact that the students divided responsibilities, i.e., some students were mainly responsible for embedded software design, others for the high-level algorithms. This was also one of the main disadvantages mentioned in the textual evaluation (at both universities): many students would have liked to have more time to work on all the topics and not only on their as-

Id	Type	Component(s)	Short Description	Preconditions	Test execution	Expected results	Success	Error Report
1	Unit Test	Communication	Establish a connection to a robot.	Robot is not connected, bluetooth switched on.	1. Establish connection	Display: BT_STREAM	yes	
2	...	...	...	...	...	...	...	...

Table 2: Example test definition

Category	Rating Berlin	Rating Potsdam	Overall Rating	Std. dev.
Embedded system design	2.59	2.19	2.39	0.81
Algorithm engineering	2.78	2.69	2.73	0.82
Project organization	2.13	2.34	2.23	0.84
Project management	2.13	2.47	2.3	0.86
Teamwork	1.66	1.78	1.72	0.78

Table 3: Learning outcome assessment

signed work packages. This could be solved by extensive cross-reviews/testing or rotating students through the various teams. Both would require more time.

6. CONCLUSION

Overall, our embedded software design project was very successful. The students gained deep insights into the challenges and perils of embedded software design and they were highly motivated throughout the whole project. The Lego Mindstorm hardware is affordable for student's projects and, at the same time, provides a perfect training platform for embedded software design, as motors, sensors, and communication are unreliable, and processing power and memory are both severely limited. By assigning the non-trivial task of a distributed multi-robot search, the students are also challenged with respect to the algorithm design, and there is a lot of room for optimizing performance and for increasing fault-tolerance, robustness, and the overall safety and reliability of possible solutions.

For future work, we plan to extend our one-term project to a second semester. This would enable us to put some additional focus on requirements engineering and quality assurance. In particular, it would be interesting to formulate the project task much more open (e.g., not to specify the use of the PSO algorithm or UPPAAL as optimal path calculation tool), and to let the students define detailed requirements on their own. In the past, we have also experimented with independent development and quality assurance teams, which introduces a broader scope and additional complexity to the project organization, but is also difficult to realize in a one-term project.

7. REFERENCES

[1] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] G. Behrmann, A. David, and K. G. Larsen. A Tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems*, LNCS 3185, pages 200–236. Springer, 2004.

[3] Bricx Command Center. http://bricxcc.sourceforge.net/.

[4] C. Ebert and C. Jones. Embedded software: Facts, figures, and future. *Computer*, (4):42–52, 2009.

[5] P. J. Gawthrop and E. McGookin. A LEGO-based control experiment. In *IEEE Control. Syst. Mag.*, pages 43 – 56. IEEE, 2004.

[6] GNU ARM toolchain. http://gnuarm.com/.

[7] W. Grega and A. Pilat. Real-time control teaching using LEGO MINDSTORMS NXT robot. In *International Multiconference on Computer Science and Information Technology (IMCSIT)*, pages 625–628, 2008.

[8] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942 – 1948, 1995.

[9] S. H. Kim and J. W. Jeon. Educating C language using LEGO Mindstorms robotic invention system 2.0. In *Proc. IEEE Robotics and Automation Conf.*, pages 715 – 720. IEEE, 2006.

[10] S. H. Kim and J. W. Jeon. Introduction for Freshmen to Embedded Systems Using LEGO Mindstorms. *IEEE Transactions on Education*, 52(1):99–108, 2009.

[11] leJOS. http://www.lejos.org/.

[12] S. McNamara, M. Cyr, C. Rogers, and B. Bratzel. LEGO brick sculptures and robotics in education. In *Proc. Amer. Soc. for Engineering Education Annu. Conf.*, 1999.

[13] nxtOSEK. http://lejos-osek.sourceforge.net/.

[14] J. Pugh and A. Martinoli. Inspiring and modeling multi-robot search with particle swarm optimization. In *IEEE Swarm Intelligence Symposium (SIS 2007)*, pages 332–339, 2007.

[15] J. Pugh and A. Martinoli. Distributed adaptation in multi-robot search using particle swarm optimization. In *From Animals to Animats 10*, volume 5040 of *Lecture Notes in Computer Science*, pages 393 – 402. Springer, 2008.

[16] Pygame. http://www.pygame.org/.

[17] M. Quottrup, T. Bak, and R. Zamanabadi. Multi-robot planning : a timed automata approach. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 5, pages 4417 – 4422, 2004.

[18] TOPPERS/ATK. http://www.toppers.jp/en/.

[19] A. B. Williams. The qualitative impact of using lego mindstorms robots to teach computer engineering. 46:206, 2003.