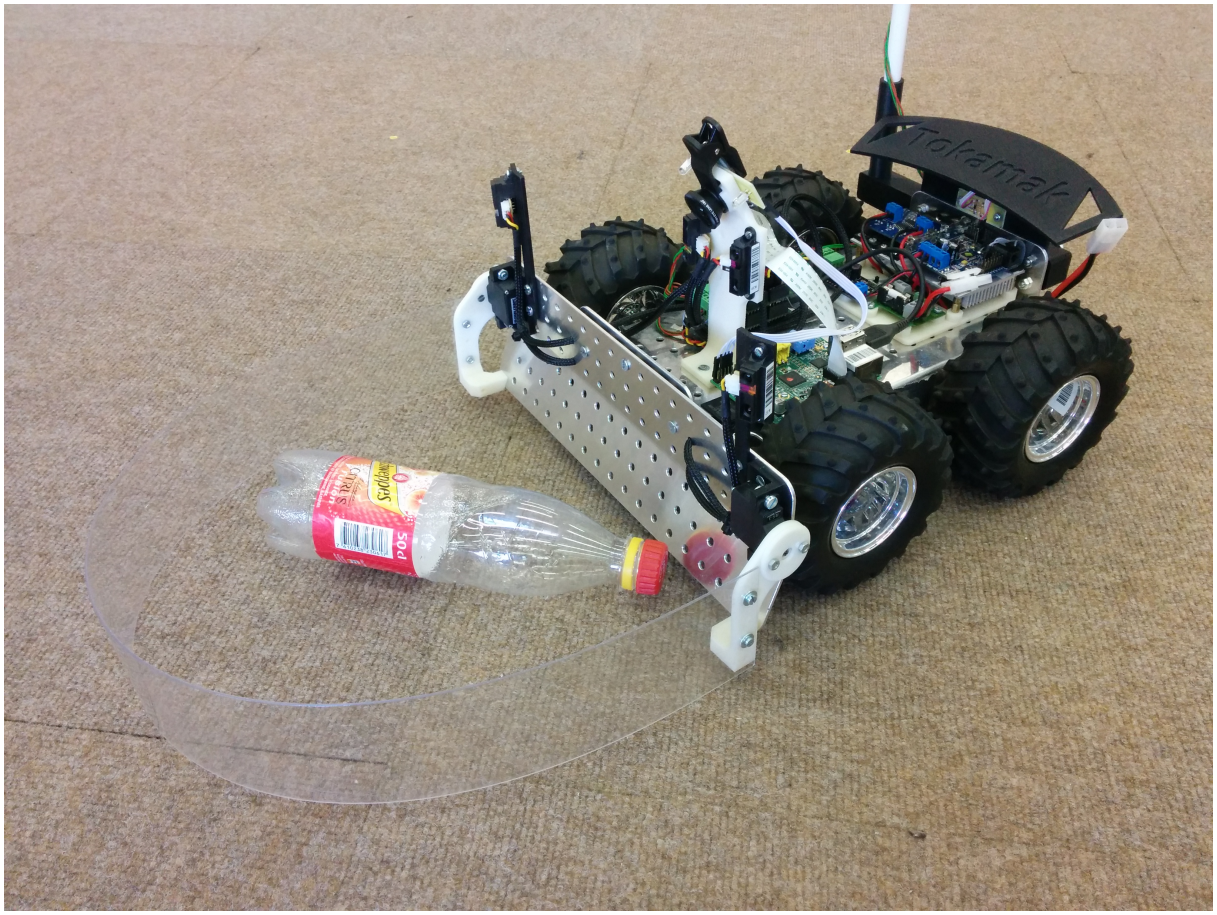
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

# STI interdisciplinary robot competition

Karl Kangur, Marcel Starein, Chun Xie (Group 5)
Assistants: Alessandro Crespi, Gorecki Tomasz
Professor: Auke Ijspeert

11th June 2014



# Contents

# 1 Abstract

The STI competition is a interdisciplinary Master semester project in form of a competition between five teams of three people from different educational backgrounds. The competition's goal is to design and build a litter collecting robot and outperform the other robots. The teams are given funds in order to buy parts, order custom parts from the in-house mechanics or make use of the available 3D printers.

# 2 Introduction

## 2.1 STI interdisciplinary robot competition

This semester project is about the design and conception of a litter collecting robot. Five teams of 3 members compete against each other, the teams have members from different educational backgrounds.

The team must use methodological product development approach as well as learn to communicate between peers of different educational backgrounds representing actual product development.

The teams are given 1000CHF that they can use to buy elements in order to build the robot and an additional 1000CHF "virtual" money that is used to order custom parts from workshops or made with 3D printers and reuse already available elements left from last year's competition.

An assistant is assigned per team to supervise the work, give advice and report to the professor in order to evaluate the group's work.

In the end the students will have received practical experience in product development with all that it implies: project development, time management, sourcing, communication, testing and competition.

We developed the most simple robot we could think of in order to have something working early on, the last year competitors had a lot of trouble during the testing and finishing part so we wanted to put as much effort as we could in the actual implementation process and thus we needed a simple platform that was easy to work on. In the end we made a small and very capable robot that was easy to maintain and modify.

## 2.2 Team members

**Karl Kangur**
: Master student in Robotic and Autonomous System at EPFL where he did his whole degree course.

**Marcel Starein**
: Master student in Robotic and Autonomous System at EPFL where he did his whole degree course.

**Chun Xie**
: Master student in Mechanical Engineering at EPFL.

# 3 Project description

## 3.1 Competition specifications

### 3.1.1 Arena

The arena is a 8x8$m^2$ square area with different zones and the recycling area as the targeted delivery area, with obstacles (bricks) inside each zone. The main area is flat with a carpet-like floor, and it's the most accessible and bottles brought back from this zone, the 1st zone, give 10 points. The 2nd zone is covered with artificial grass making access a little bit more difficult and each bottle from that zone gives 20 points. The 3rd zone is surrounded by rocks which makes access quite difficult, bottles from this area give 40 points. Finally the 4th zone is a raised platform with 2 access points, one ramp (B2) and some stairs (B3), the bottles from this area also give 40 points.

The plan view of the arena is as figure 1 shows.

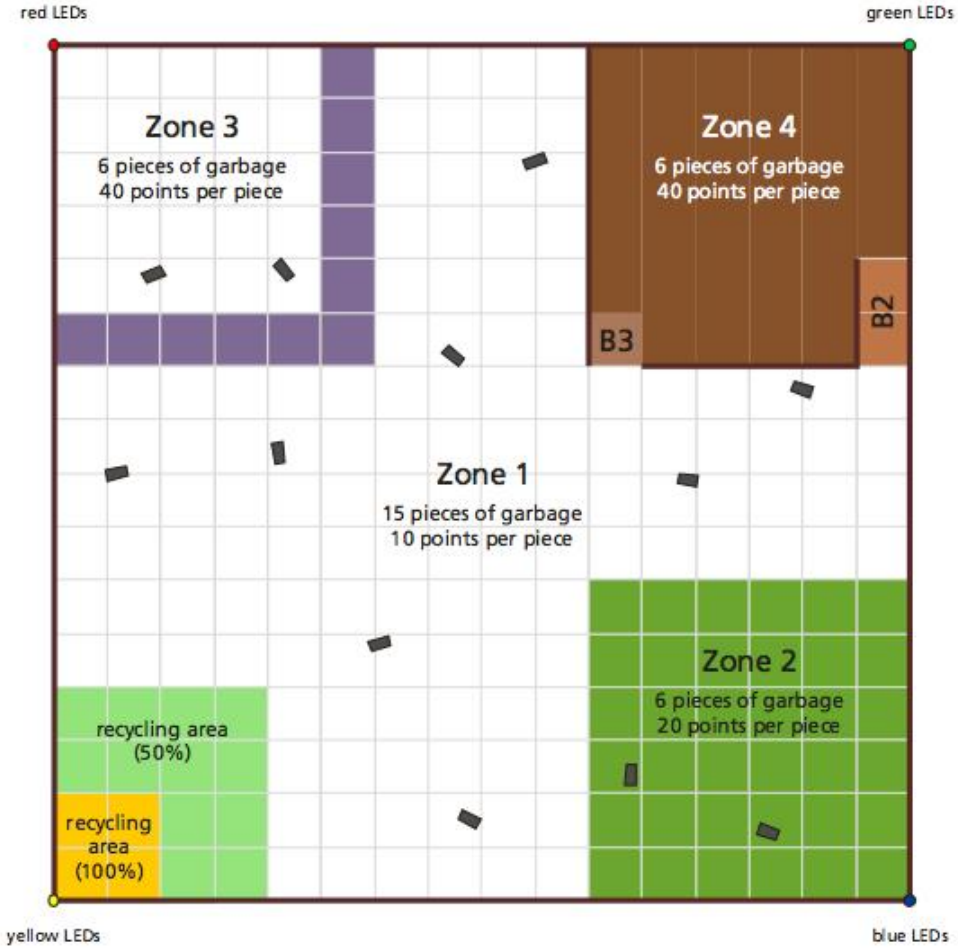



**Figure 1:** Arena

The actual whole arena and specific Zone 3 and Zone 4 are as figure 2 shows.

(a) Actual arena

(b) Zone 3

(c) Zone 4

**Figure 2:** Actual arena, Zone 3 and Zone 4. The brick obstacles are not positioned in the competition configuration, there should be 2 stacked bricks.

### 3.1.2 Bottles

The bottles are common plastic beverage bottles with the volume not exceeding 500ml, transparent or opaque ones, like the following figures, which are randomly placed in the garbage located areas.


**Figure 3:** Bottles

### 3.1.3 Goals

The robot must first explore the arena in order to find something. It must be capable of avoiding the obstacles (bricks), detect bottles and then somehow move the bottle to the recycling area, repeat the process and accumulate points to win.

## 3.2 Strategy options

### 3.2.1 Non-selective storage

#### 3.2.1.1 Maximum volume storage

When exploring within the arena, the robot stores the maximum number of bottles at once, and then brings them all back to the recycling area.

#### 3.2.1.2 Single-piece storage

When exploring within the arena, the robot stores only one bottle at once, and then brings it back to the recycling area.

### 3.2.2 Selective storage

Collect the bottles that yield the most points or focus on bringing back the most accessible bottles.

### 3.3 Selected solution

Since we wanted a simple robot we decided we'd focus on the main goals only, that is bringing back a single bottle, once that was done the robot could theoretically fetch other bottles or we could even make a multi-robot system. This solution had the main advantage of keeping the whole system easy to manage and build while keeping our options open for further development.

Our robot was thus built to bring back one bottle at a time and we conceived a simple bottle storage system that only needed 2 actuators. For bottle detection we decided to go with the same solution that the last year's group 5 did: a classifier using Haar Cascades algorithm that proved to be reliable, we also used the same hardware as they did (Raspberry Pi).

# 4 Project analysis

We have applied the standard product development approach to help us decide on the best solution for this project, it begins with a list of needs and this will be the basis for the project scope. The items shall define the goals of the project and in no way hint to a solution.

## 4.1 List of needs

- Being able to move inside the arena
- No human interaction
- On-board computation
- Detect garbage
- Move garbage
- Dispose of garbage in the designated area
- Avoid obstacles
- Autonomy of at least 10 minutes

## 4.2 Function specification

### 4.2.1 External

The robot must be able to run automatically without interaction from external controller, and persist its stable behaviors to external noise and disturbance from the surrounding environment.

- No human interaction
- Robustness to external noise and disturbance

### 4.2.2 Internal

The robot must be able to move itself in the arena on wheels. It must detect and avoid obstacles, find the bottles and transport as many as possible back to the designated area in 10 minutes.

- Being able to move
  - Flat surface
  - Power autonomy for at least 10 minutes
  - Fast enough so that some waste could be collected within 10 minutes
- Localization
  - Find the recycling area
- Object detection
  - Differentiate between litter and obstacles
  - Avoid obstacles
- Manipulate objects
  - Move the litter

## 4.3 Critical technical points

The critical technical points were (in order): the locomotion, obstacle avoidance, bottle detection, bottle manipulation and finding the recycling area. We proceeded in this order to solve all these problems so that we could concentrate on one problem at a time.

## 4.4 Solutions identification

### 4.4.1 Movement

The first consideration is the robot's locomotion. Several strategies for its movement are listed in table 1, including the specific principles, advantages, disadvantages and risks corresponding to the strategies.

**Table 1:** Movement strategy

| Strategy | Principle | Advantage | Disadvantage | Risk |
|---|---|---|---|---|
| Custom chassis | Custom-made chassis | Can take any form needed | Takes times to make | Might not finish building on time, might not work |
| Wild Thumper | Powerful differential mobile base | Can navigate any terrain. Readily available with the control electronics, can start working on it immediately | Requires adaptation | Motors might break when load too heavy |
| Rover 5 | Differential mobile base on tracks | Can navigate any terrain | Must be ordered | May not be powerful enough |
| Cartesian robot | Moves in x-y axis over the whole terrain | Can move over any terrain | Too big and heavy, takes time to put in place, doesn't really go with the spirit of this competition | Too expensive |
| Quadcopter | Exploits the 3rd dimension | Can move over any terrain, can see everything from above | Not allowed | Getting disqualified |
| 2-wheel differential robot | 2 wheeled robot | Easy to control, only 2 motors | Needs a 3rd passive wheel | May not be powerful enough |
| Swedish wheels | Wheels on wheels. 4 wheels | Allows movement in any direction | Not as precise as conventional wheels, odometry very difficult. Cannot climb slopes | No net gain in locomotion compared to other methods |
| Hexapod | 6 legged robot | Navigation through complex terrain | Hard to program, slow, lots of parts | Might take too long to implement |
| Quadruped | 4 legged robot | Can move over any terrain | Statically unstable, difficult to control | Might take too long to implement |
| Biped | 2 legged robot | Can access all terrains | Extremely difficult to program and control. Not statically stable | Too ambitious |

| | | | | |
|---|---|---|---|---|
| Synchrodrive | All wheels turn synchronously and the chassis doesn't rotate | Nothing really | Requires a custom chassis and lots of moving parts | |
| Large robot arm | A serial robot that is placed in the middle of the arena and can reach any place on it | Can reach any area with ease | Extremely expensive. Difficult to put in place. Heavy, potentially dangerous | Probably too expensive. Might kill somebody |
| Crawling | Chassis consisting of multiple sections with actuated motors in-between | Can access all terrains | Very difficult to control, expensive and time consuming to make | Might not finish building on time, might not work |
| Hopping | Movement with a series of jumps | Can move over obstacles and large distances fast | Hard to control, to manufacture as there aren't any commercial products | Getting yelled at because it's a stupid idea |
| Hovercraft | Movement on an air cushion | Can move over any terrain | Cannot move up slopes, difficult to control, noise, needs a lot of power | Not enough autonomy, too complicated |

In line with the design concept of "as simple as possible", and through the analysis of the advantages, disadvantages and risks for each listed strategy, a combination of customer chassis and Wild Thumper becomes the final decision, which means using the wheels of Wild Thumper and customer chassis, due to the following reasons. On one hand, customer chassis has more simple structure which is qualified enough for the flat arena, lower cost and more flexibility to add any needed components; on the other hand, the more powerful chassis of Wild Thumper makes itself oscillate more on flat arena, even though it is appropriate for many more extreme road conditions, which makes the robot body lack the stability while moving, not quite satisfying the objective of moving stable on flat arena, and easily creating some issues during the later sequence of robot behavoirs.

### 4.4.2 Object detection

It is crucial for the robot to do object detection, including the bottles, the obstacles (bricks) and surrounding walls, so that it could do the sequence of behaviors containing avoiding the obstacles, avoiding the walls, and finding the bottles. Several strategies for object detection are listed in table 2, including the advantages, disadvantages and risks corresponding to the relative strategies.

**Table 2:** Object Detection Strategy

| Strategy | Advantage | Disadvantage | Risk |
|---|---|---|---|
| Ultrasound | Linear response with distance, not affected by target materials, surfaces and color. Can detect small objects over long operating distances. Resistance to external disturbances such as vibration, infrared radiation, ambient noise, and EMI radiation | Must view a surface (especially a hard, flat surface) squarely (perpendicularly) to receive ample sound echo. Requires time for the transducer to stop ringing after each transmission burst before they are ready to receive returned echoes. Have a minimum sensing distance. Changes in the environment, target of density, smooth of surfaces affect ultrasonic response | False positive outputs due to a large operating angle, detecting an object other than the desired target. |
| Infrared | Detect infrared light from far distances over a large area. In real-time and detect movement. | Incapable of distinguishing between objects. | Strong infrared sources might be detected as obstacles. |
| Laser rangefinders | Better accuracy more quickly. Easy alignment by employing visible red laser beam. Detects of very small targets due to small measuring spot size | Suffer from laser noise, stray light, and speckle effects interference. | Detect an object other than the desired target. |
| Structured light | Can do 3D imaging using a simple and cheap algorithm. | Needs lots of processing power | Powerful computer needed. |
| Tactile sensors | Guaranteed obstacle detection. Allow physical interaction with objects | Must be close enough to the obstacle, cannot abvoid without physical interaction. | Hit the obstacles while detecting |
| Color sensor | High speed, easy to use and relative intensity display. | Complex calibration and limited accuracy | Just detect objects with certain colours |
| Surface transducer | Less sensitive to surface condition | Low transduction efficiency | Need more time and detect an object other than the desired target |
| Camera | Cheaper, more informative and more compact | Limitation of its view fields | Might not detect the whole targeted space |

| | | | |
|---|---|---|---|
| Stereovision | Can do 3D vision. | Complex, poor dynamic range and still not very reliable | Powerful computer needed. |
| Millimeter Wave Radar | Accurate, excellent image identification and resolution | Too expensive. | More expensive than other technologies |

At first we wanted to use only one camera to do everything in order to keep a simple system. The camera could do different kinds of image processing and in theory detect and differentiate all the objects. When testing we saw that the camera could not differentiate between the floor and walls using the color information and was actually quite slow when processing the video stream, this meant we had to use other sensors to complement the camera. In the end we chose to use the camera only for the bottle detection as it could do it reliably and use infra red sensors for wall and obstacle detection.

### 4.4.3 Bottle grasping

After the achievement of finding bottles, the actuator should have the capacity of grasping and storing the bottles, so that the robot could complete the recycling target. There are a lot of practical ways for the robot to the grasp bottles, and several strategies are listed in table 3, including the principles, advantages, disadvantages and risks corresponding to the relative strategies.

**Table 3:** Bottle Grasping Strategy

| Strategy | Principle | Advantage | Disadvantage | Risk |
|---|---|---|---|---|
| Robotic arm | Arm with a few DDLs mounted onto the robot | Allows picking up bottle in every position and in every terrain | Difficult mechanical realisation and time consuming programming | Complex implementation. High possibility that mechanism won't work as intended for different situations |
| Clamp | Grabbing a bottle in front of the robot with a clamp | Depending on DDLs wanted, can be very easy to realise | Needs good precision in positioning to grab a bottle | Not being able to position the robot to pickup the bottle |
| Suction | Suction mechanism to hold bottles | Easy pick-up and release | Requires compressor, energy consuming, can be hard to position on bottle | No good seal between bottle and suction mechanism, therefor not being able to apply enough suction |
| Pushing | Bottle being rolled with the robots chassis | No extra mechanical parts | Can be hard to perform complex movements while keeping bottle pushed. Hard when bottle positioned close to an edge or corner | Losing the bottle underway, hence loosing time with trying to recuperate it if it is even possible |
| Storage bay for single bottle | Robot ingests bottle inside a storage bay | Easy mechanical implementation, carrying bottle around relatively easy | Must return to base for every single bottle, must be well aligned with the bottle | Low risk |
| Storage bay for multiple bottles | Robot ingests bottle inside a storage bay, while being able to store a few of them | No time lost going back to the base each time | Robot ingests bottle inside a storage bay, hard to implement storage system, requires a bigger robot | Failure of the storage system, not releasing or stocking bottles correctly |
| Deployable cage | Deploying a cage to surround the object, and bring it back to the base. The bottle rolls on the floor | Very easy to implement, bottle position doesn't have to be exact, can grab bottle in any position or orientation | Can't bring bottle over rough terrain, only one bottle at a time | Low risk |

| | | | | |
|---|---|---|---|---|
| Harpoon | Throwing a harpoon to grab a bottle | Robot doesn't need to move around much | Requires good precision, launching system, retrieval system | Failure to aim correctly, and for the harpoon to pierce the bottle |
| Net | Throwing a net | Robot doesn't need to move around much | Requires good precision, launching system, retrieval system | Failure to aim correctly, net not deploying as planed |
| Compressed air | Blowing compressed air on the bottle to move the bottle around | No mechanical moving part | Complex aiming and bottle trajectory planing, requires compressor or to carry compressed air | Hard to predict bottle movement |
| Scotch | Sticky surface | Cheap, big supply | Adhesive wears off with dust | Bottle does not stick to it |

In accordance with the concept of "as simple as possible", and through the analysis of the advantages, disadvantages, and risks for each listed strategy, deployable cage becomes the final decision, which calls for much more simple mechanical structure, and actuator motion to grasp and store bottles, with less cost and risk but more availability and reliability.

### 4.4.4 Localisation

The robot had to have a way to go back to the recycling area when it had collected a bottle so it had to have some information about where that goal was. It didn't really need to know where it was with absolute positioning on the arena as it was meant to roam around randomly to find the bottles anyway. As long as it could find the recycling area after having collected a bottle it was enough. Several localisation strategies are listed in table 4, including the advantages, disadvantages and risks corresponding to the relative strategies.

**Table 4:** Localisation Strategy

| Strategy | Advantage | Disadvantage | Risk |
|---|---|---|---|
| Beacon-based positioning | Active beacons available on the terrain | External conditions may influence results and interfere with sensor values | Sensible to environmental conditions |
| Odometry or encoders | Easy to implement in software, integrated into motors. Very precise | Cumulative error. Absolute positioning still required | Wheel slip makes robot lost immediately |
| Inertial measurement unit | Cheap and easy to integrate, ready-made boards exist with Kalman filters that return the x-y position | Cumulative error. Absolute positioning still required | Fast accelerations might disturb the system |

| | | | |
|---|---|---|---|
| Global positioning system | Absolute position anywhere on earth | Cannot be used inside. Consumes a lot of power. Not precise (+/-3m) | Won't work |
| Motion field and optic flow | Precise, fast | Cumulative error. Absolute positioning still required | |
| RFID | Available. Absolute positioning | Not easy to detect, need to be right over it to detect it | Robot might not pass over a tag for a long time |
| Linear cameras | Easy to implement | Needs a powerful light source from a beacon. Might interfere with other surrounding lights | |
| SLAM (camera) | Can be made to be very robust, use existing algorithms | Complex to use, heavy processing needed | Too slow or not enough time to optimal implementation |
| Markovian localisation (camera) | Particle filter localization. Algorithm estimates the position and orientation of a robot as it moves and senses the environment | Terrain changes (removed litter) | Might not converge to actual robot position |
| Monte Carlo localization (camera) | Grid-based localization, which uses a histogram to represent the belief distribution | Terrain changes (removed litter) | Might not converge to actual robot position |
| SURF, SIFT... (camera) | Feature point detection | Terrain changes (removed litter). Needs a lot of computing power | Might not converge to actual robot position |
| Color blob based localisation (camera) | Detect colors on images and interpret. Active beacons around the arena can be used. Terrain features are of different color | Bottles are transparent. Computationally expensive | Might not converge to actual robot position |
| Kinekt | 3D imaging, depth information can be useful | Needs a powerful computer to process data and correlate to a virtual map | Too time consuming |
| Visual odometry (egomotion) (camera) | Existing algorithms | Needs a lot of computing power | Computationally too expensive, cumulative error |

Since the camera is focused on the bottle detection, to reduce the complexity of localisation and the workload of the camera, the localisation strategy which adopts camera will not be considered,

which corresponds to the concept of "as simple as possible". Taking the availability, implementation difficulty, reliability and cost into account, the inertial measurement unit becomes the final decision. Since the target is just to let the robot go to one certain direction that directs the robot to the recycling area, which is fixed, the absolute heading direction for the robot to is the same, wherever the robot is within the arena. Then the inertial measurement unit is used as a compass for absolute heading towards the direction parallel to the recycling area. When a wall is detected the robot will simply follow it until it finds the only corner which means it has reached to goal.

## 4.5 Risk analysis

We first identified the problems the groups competing last year had, to summarise: all projects were way too complicated and it took them a long time to get the mechanical parts they ordered, once they had them they didn't have much time to test the robot before the competition and thus they couldn't fix the minor problems they didn't think of at the time. We took note of that and that was the main reason we searched for a simple solution from the beginning.

The risks we had with our project were mainly the locomotion and software. One of the groups last year broke their motors right before the competition and their robot couldn't move, so the motors have already been a problem and we needed to test them to make sure they worked well. The software needed to be quite complex and took time to write, so we started to test early on with the vision system and see what performance we got from the image processing. Again last year multiple groups were far from ready in terms of robot intelligence during the competition and were not able to complete the required tasks.

## 4.6 Gantt diagram

The main tasks and development plan are as follows, and detailed Gantt diagram is seen in appendix A.

- MS1: Function analysis and solutions identification
- MS2: Function analysis and solutions refinement
- MS3: Development of key functions individually, Refinement of design, Planifications of project, Feasibility study
- Development and production according to planning
- Software development
  - Bottle detection algorithm
  - Object and obstacle detection algorithm
  - Motor controller programming (H-bridge)
- Hardware development
  - New 4WD chassis (custom made)
  - Component sourcing
  - Deployable cage
- Final assembly
- Trials and optimisation, Testing in real conditions on the arena
- Rehearsal competition
- MS4: Competition, Report writing

# 5 Project design

## 5.1 Robot design

Our robot was designed using an incremental approach. We designed the various systems and pieces one after the other, making sure everything is working before passing to the next sub system.

Hardware and software were developed in parallel. On the hardware side, we started with the motor controller, linking it to all 4 motors and performing various tests. We performed tests with the motors in order to determine the required reduction gear. Then, we proceeded with the 5V power supply and with the Arduino board. Once the two were connected through I2C, and testing had been done, we proceeded with connecting the IR sensors, IMU unit and the servo motors. During all phases, the different components were linked together with veroboards. Once everything had been tested, we created final printed circuit boards.

## 5.2 Hardware

### 5.2.1 Motors

We initially opted for using the WildThumper chassis, with its wheels and motors, as a base for our robot. However, when we first tried out the chassis, we discovered several flaws. One of them being the springs for the chassis' adaptation to extreme road conditions, which makes robot oscillate more when moving on the flat arena. Therefore, we made a custom chassis, but kept the motors and wheels.

Initially, we had motors with 34:1 reduction gears, which we thought were fast and powerful enough for our light robot design. However, after initial testing, we discovered that these motors offered far too low torque at low speeds and hence decided to try out 75:1 motors. Again, these motors didn't offer enough torque at low speeds in order to move our robot, and therefore we finally switched to 172:1 gearing ratio.

We tried both high powered and low powered motors as the motors were sold in 2 types, the high power motor offering more torque as per the specifications. However, the high powered motors aren't reliable enough for our taste, as the 172:1 version can easily break the gearing, we actually broke one motor simply when testing it without any load, that shows how reliable these motors are and we cannot recommend them for the next year competition.

We finally opted for 172:1 low powered motors, which allow us to attain reasonable speeds and also let us move slowly. The robot needs to be able to move slowly because it is limited by the slower image processing script used for bottle detection.

### 5.2.2 Servomotors

We first tried using standard, lower priced servos, in order to actuate the cage. However, the servo-arms offered with those servos weren't strong enough in order to directly screw the cage onto them. Therefore, we decided to order metal arms, as well as new metal servos which would rigidify our cage mechanism. However, once receiving the servos, we busted one and hence decided to go back to the smaller servos, which, with an adapted arm mount, custom 3D printed, ended up offering good performance.

### 5.2.3 Cage

The cage was initially designed to be square, and when in the upper position, placed around the robot. We quickly found it would be much easier to create a cage which wouldn't surround the

whole robot, but stay against the upper position of the robot chassis, as we remove the risk of getting the cage stuck with the chassis or the wheels. We opted for a round design, which doesn't need any precise folding and being constrained is more rigid.

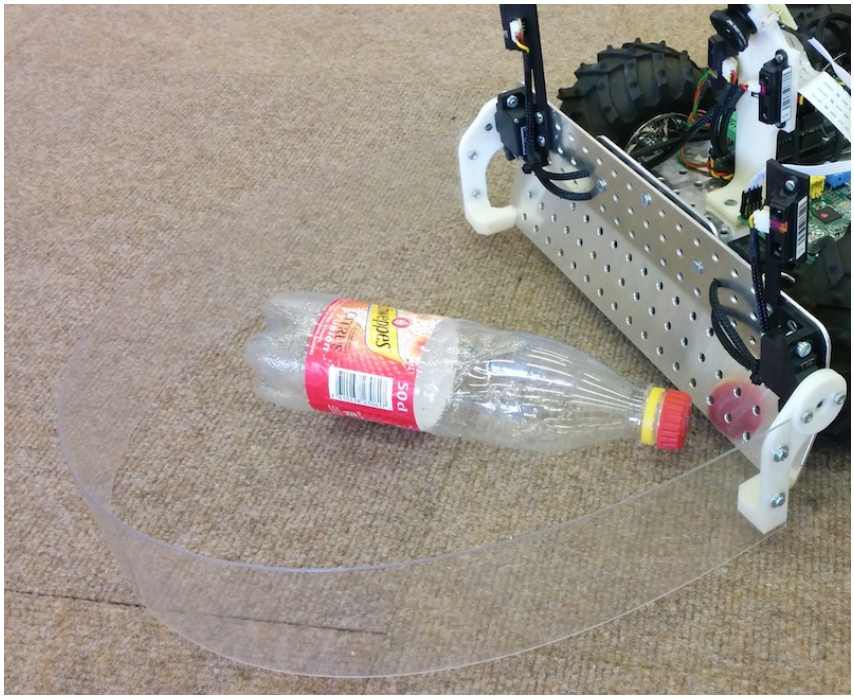


**Figure 4:** Deployable bottle cage

We tried searching for transparent materials, which would allow us to let the camera see through the cage. We also wanted a material which deforms in case an unwanted collision occurred. We ended up opting for a transparent plastic, linked to the servo with custom printed 3D parts.

## 5.3 Electronics

### 5.3.1 Raspberry Pi

The Raspberry Pi is a very cheap and relatively powerful small 700MHz computer (figure 5). We chose it because we already had some experience working with it and it has a significant community behind it that could help us out in case of problems.

### 5.3.2 Raspberry Pi camera

For the imaging system there was two alternatives: use a USB webcam or buy the Raspberry Pi Camera module as seen in figure 6. Since with the webcam, the USB bus would have been a bottleneck in terms of data exchange with the processor, we opted for the Raspberry Pi Camera, not only could it take very high resolution pictures (8MP) with a decent quality but also it was very fast in image acquisition and as we needed live image processing this was an important aspect.

### 5.3.3 PRismino

We decided to use this board as it was cheap and available. It's an Arduino clone board made by the EPFL robotics club - Robopoly (figure 7).

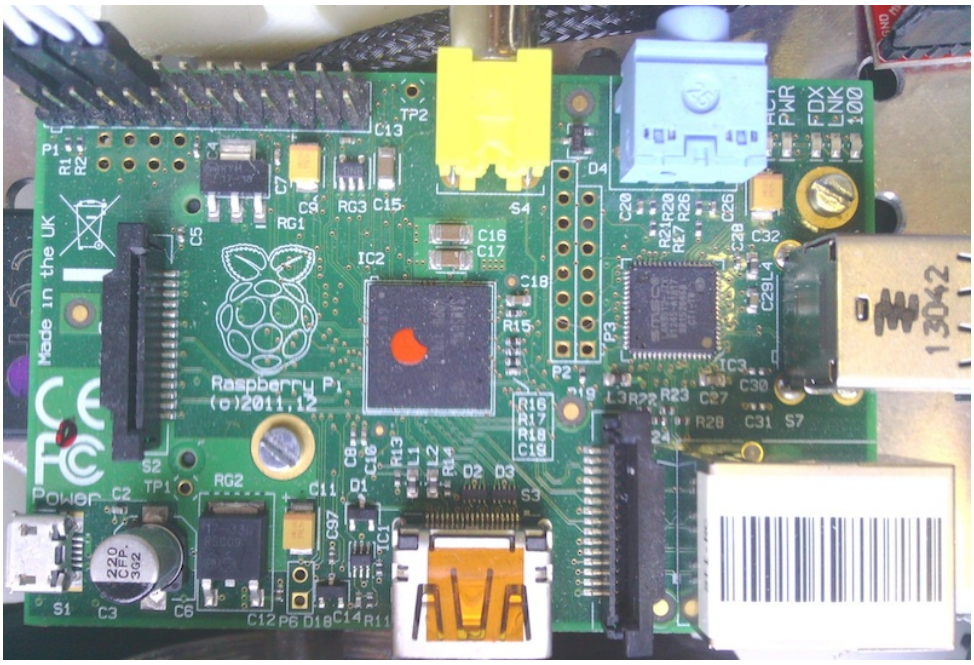



**Figure 5:** Raspberry Pi on-board computer runnung Linux and OpenCV image processing software


**Figure 6:** Raspberry Pi camera module

It offers more than enough control pins for all our sensors and servos. The Arduino boards have a very large community and most libraries are already existing, which make them very easy to implement and use effectively and fast.

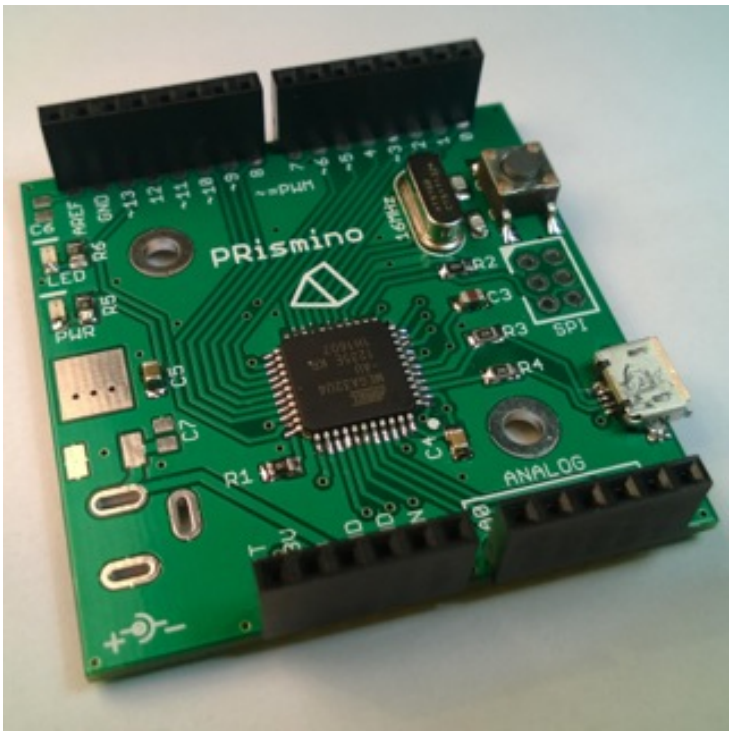



**Figure 7:** Robopoly robotics platform micro-controller board

We also made a custom shield for our PRismino, which offers connectors for the servo motors and IR sensors. It also offers 3.3V $I^2C$ lines, a buzzer and a Bluetooth module for wireless communication, in order to test our robot's functions easily and for debugging.

### 5.3.4 Motor controller

Since we were initially using the WildThumper, we also chose the WildThumper motor controller shown in figure 8. Since we already had the board implemented correctly, we didn't want switch to another controller once we decided to ditch the WildThumper.

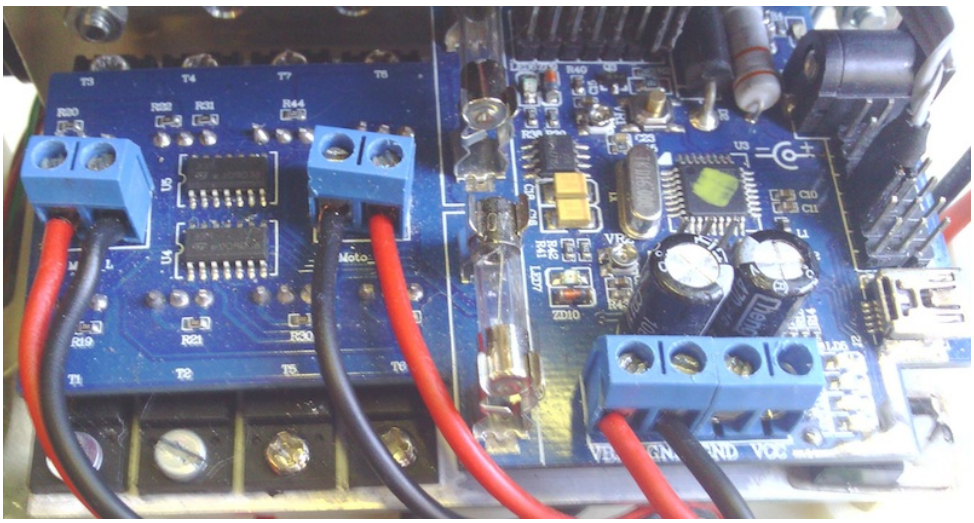


**Figure 8:** Motor controller for powering the wheels.

On top of that, the controller was initially designed to work with the WildThumper motors,

which we were using. It is also equipped with an Arduino, which makes it easily reprogrammable and easy to integrate with the PRismino

We reprogrammed the WildThumper micro controller to use its timers more efficiently than the provided code, the new code was based on the Robopoly shield that has a similar way of controlling its H-bridge.

### 5.3.5 Compass

#### 5.3.5.1 MPU-9150

This IMU unit offers 3 axis acceleration, gyro and compass outputs and has an integrated DSP (figure 9). Unfortunately InvenSense has a discouragement policy, by not supplying enough information for the use of the DMP. It works at 3.3V, instead of using logic level converters, as the PRismino works with 5V, we simply use pull-up resistors to 3.3V on the $I^2C$ lines which works just as well.

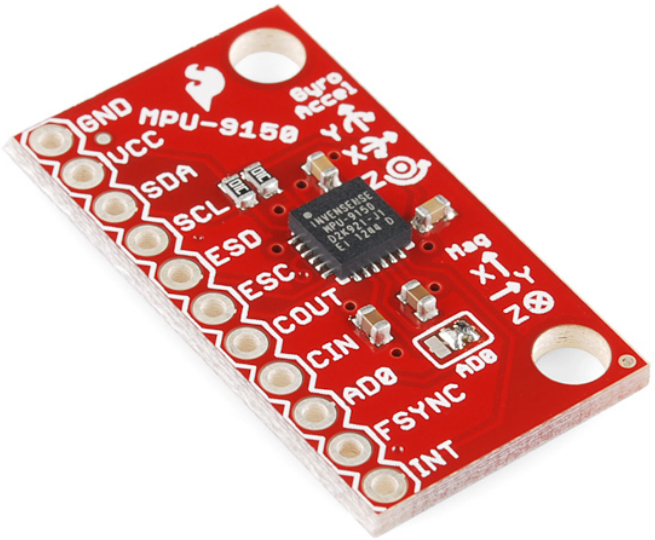



**Figure 9:** MPU-9150 breakout board

#### 5.3.5.2 GY-85

The GY-85 (figure 10) is a very cheap and also very capable IMU. Instead of having 1 chip that does all like the MPU-9150 it has 3 chips: one for acceleration, one for gyroscope and a compass. It also has a 3.3V supply and logic level converter for hte I2C lines making it compatible with 5V logic. When we compared the MPU-9150 and GY-85 we found that the GY-85 was easier to use and we could drive our I2C lines at 5V, so we opted for this option.

### 5.3.6 5V regulator

As the on-board 5V regulator on the motor controller is a linear regulator (LM1084) we decided that for safety we would decouple the controller and Raspberry Pi from each other. We used the TPS62133 switching step-down regulator, which was part of the kit developed by the EPFL

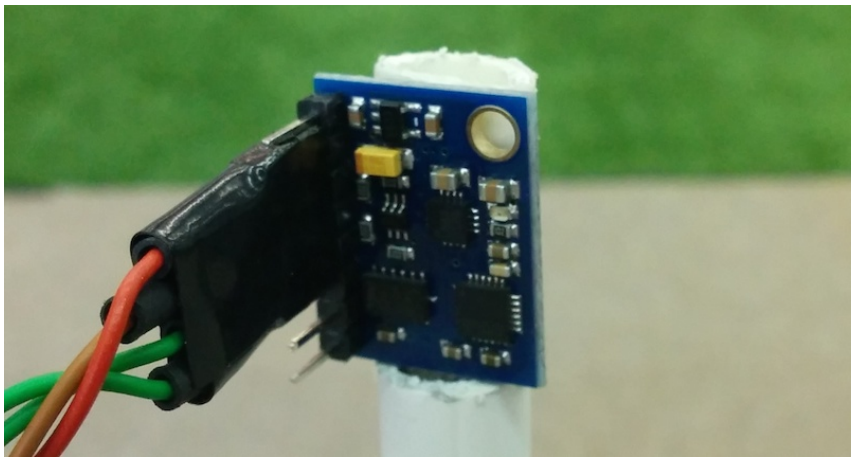


**Figure 10:** GY-85 IMU board

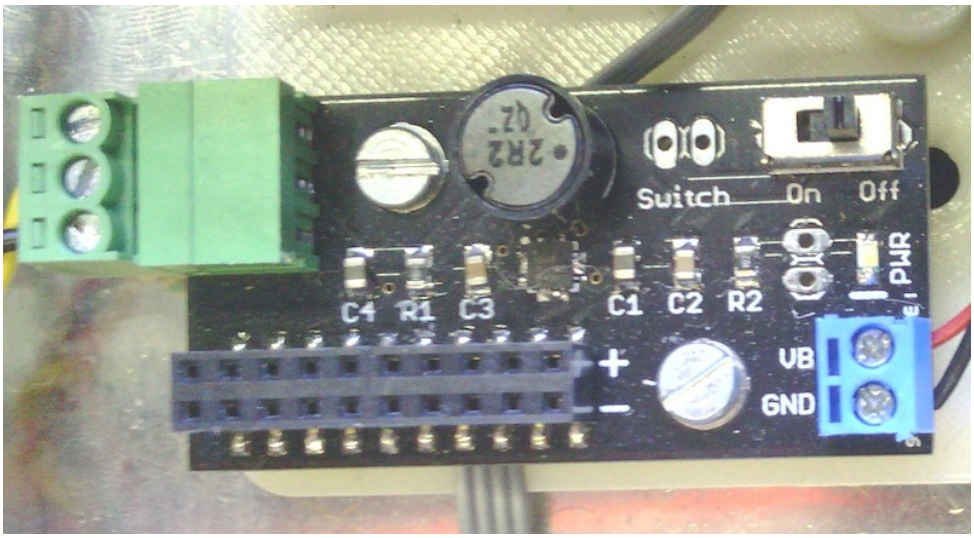



**Figure 11:** 5 volt regulator for powering all the logic electronics.

robotics club (Robopoly) as shown of figure 11. This allowed us to get efficient 5V regulation for the logic part of the robot as the Raspberry Pi was consuming quite a lot of power (300-400mA).

We also noticed that the motor controller 5V regulator actually output 5.54V instead, this was out of specifications for the Raspberry Pi as well as the micro controller on the motor controller, which might have an effect on its longevity. Again we must point out the lack of quality with Pololu products.

### 5.3.7 Power

Our estimations showed us that the available 7.2V, 3000mAh NiMH battery was more than enough to power the robot for the expected 10 minutes of the competition. So we bought 2 with the virtual budget. During testing one could last almost a complete day any they recharged in only 1 hour.

### 5.3.8 Custom PCBs

We made a custom connector shield to connect all the sensors and to have the buzzer, I2C lines for compass and motor controller communication and Bluetooth (figure 12). This made it look much nicer and more reliable than prototyping cables that were all over the robot.

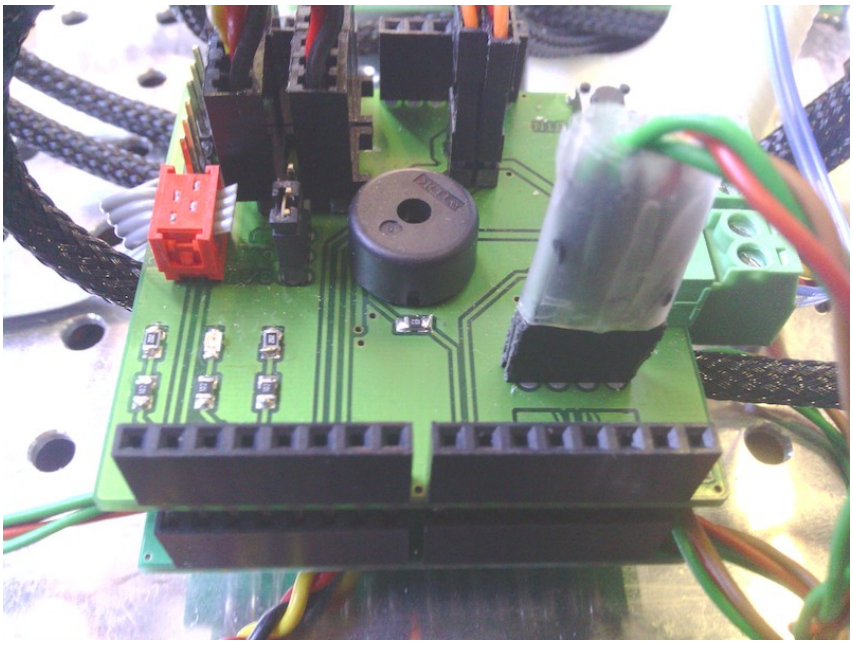


**Figure 12:** Custom connector shield PCB

We also made a PCB for the front lights, we found that when we wanted to use the camera as obstacle detection system we could see the bricks better when some light was shining on them, so we made a PCB for high-power LEDs that were attached besides the camera. We also made a small PCB for rear lights (figure 13) to make the back look like Formula 1 cars in case of low-visibility on the arena.

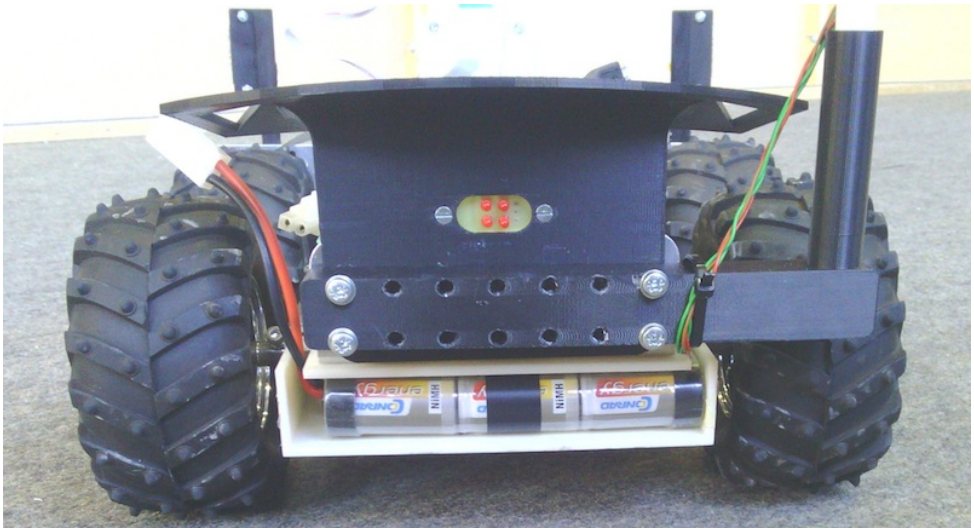


**Figure 13:** Back lights

## 5.4 Communication between modules

The PRismino, motor controller board and magnetometer unit are all connected onto a 3.3V $I^2C$ bus. The Prismino serves as the master, and controls all motors as well as reads all sensors. The Raspberry Pi, on the other hand, is communicating via serial (USB) with the PRismino. The Raspberry is sending bottle positions to the Prismino.

Since the Raspberry Pi is quite a complex system and may fail in unexpected ways such as memory corruption we had a backup plan where the robot isn't able to bring back bottles, but can still roam around using a simple collision avoidance program on the PRismino and the IR sensors.

## 5.5 Mechanical design

### 5.5.1 Bought parts

The components bought with virtual budget and real budget are respectively shown in Table 5 and Table 6 as follows.

**Table 5:** Expenses for parts bought with the virtual budget

| Part | Quantity | Price (CHF) | Total (CHF) | Description |
|---|---|---|---|---|
| Raspberry Pi | 1 | 36.20 | 36.20 | Main computer board for the robot intelligence, does the image processing of the camera. |
| Raspberry Pi Camera | 1 | 31.55 | 31.55 | Camera for robot vision |
| SD Card 8 GB | 1 | 10.00 | 10.00 | Needed for the Raspberry Pi |
| Motor controller | 1 | 75.70 | 75.70 | Wild Thumper motor controller |
| Battery | 1 | 19.95 | 19.95 | NiMH rechargeable battery pack |
| Fuse board | 1 | 0.00 | 0.00 | Battery connector/fuse board for security |
| Motor | 4 | 34.95 | 139.80 | 172:1 DC motor with encoder |
| Wheel | 4 | 7.50 | 30.00 | WildTumper 120x60 mm wheel |
| IR sensor | 4 | 18.60 | 74.40 | 80 cm IR proximity sensor |
| 9 Degrees of Freedom IMU | 1 | 34.95 | 34.95 | Used as compass for robot return |
| Total | | | 472.50 | |

### 5.5.2 Custom parts

We asked the mechanics to make some of the parts for the robot as they had to be adapted for the task in hand. The chassis was custom made as the WildThumper chassis was impractical and bulky, and we needed a solid frame with lots of attachment points in order to mount all the electronics on it.

Although the WildThumper chassis does have a lot of holes and is quite versatile, but we did not want to have the suspension which could influence the camera's point of view, and indeed if the robot was swaying every which way, the camera, which had to be quite high, would move a lot as well, and the image processing would have been affected.

The custom chassis was made to be easily manufactured and as modular as possible, and we ended up with two pieces of 2mm thick aluminium sheet metal parts that had to be bent. The main chassis part had holes for the 4 motors and lots of holes for electronics mounting. The

**Table 6:** Expenses for parts bought with the real budget

| Part | Quantity | Price (CHF) | Total (CHF) | Provider |
|---|---|---|---|---|
| Commande Conrad | 1 | 159.10 | 159.10 | Conrad |
| Commande Pololu | 1 | 211.37 | 211.37 | Pololu |
| Header 1x3P, 6373-A3A-102/2223-2031, Molex | 10 | 0.14 | 1.40 | Distrelec |
| PRismino | 1 | 7.00 | 7.00 | Robopoly |
| Bluetooth module (HC-05) | 1 | 6.00 | 6.00 | Robopoly |
| Lentille optique | 1 | 7.00 | 7.00 | Robopoly |
| Composants électriques pour shield | 1 | 3.00 | 3.00 | Robopoly |
| Sevomoteurs | 3 | 10.00 | 30.00 | Robopoly |
| Power-board | 1 | 5.00 | 5.00 | Robopoly |
| Divers composants électroniques | 1 | 5.00 | 5.00 | Robopoly |
| TVA | 1 | 5.00 | 5.00 | Robopoly |
| GY-85 6DOF 9DOF IMU Sensor Module | 1 | 8.5 | 8.5 | DealExtreme |
| Total | | | 448.37 | |

second part was a plow that had the function of pushing the bottles, otherwise the robot would have rolled over the bottles, and also it held the servomotors for the deployable cage. It took three weeks for the 2 custom parts to be made.

As the hole separation on the custom chassis could be arbitrary, we decided for 16mm separation as this makes it compatible with LEGO parts as they are perfect for very fast prototyping.

### 5.5.3 3D printed parts

We had a total of ten 3D printed parts on the robot to hold our various electronic parts on the custom chassis.

- Battery holder
- H-bridge, fuse board and 5V regulator board support
- Raspberry Pi support
- Camera support
- Servo motor holders
- IR sensor supports
- Cage - servo links
- PRismino support
- Compass holder
- Read cage support

We also made some prototype parts that needed to be modified, but did not end up on the robot itself, which were also included in the 3D printed parts total cost. In total we spent 186.01CHF on the printed parts.


**Figure 14:** 3D printed rear cage support with robot name on it

## 5.6 Budget management

We tried using as many parts as possible available from the catalogue, as they are easily available and we were able to continue our project as fast as possible. Of course, some of the components required, such as the small servos for the cage, weren't available and hence were bought with the real budget.

The whole budget for this project is shown in Table 7 as follows.

**Table 7:** Expenses for the whole budget

| Provider | Virtual budget(CHF) | Real budget (CHF) |
|---|---|---|
| Distrelec | 0.00 | 1.40 |
| Pololu | 0.00 | 211.37 |
| Robopoly | 0.00 | 68.00 |
| Conrad | 0.00 | 159.10 |
| Virtual | 472.50 | 0.00 |
| 3D printed parts | 186.01 | 0.00 |
| DealExtreme | 0.00 | 8.50 |
| Total | 658.51 | 448.37 |

## 5.7 Software

The software design approach is similar to the rest of the project, and keeping it simple is the most important. According to the targeted functions, the design theory and flowchart is shown as Figure 15.

The obstacle avoidance was the first thing we had to implement, this meant that the robot was able to roam around without hitting anything and thus the next step of actually detecting bottles could be implemented.

The obstacle detection is done using the camera and 4 infra-red sensors on the front of the robot, as the camera cannot discern between the floor and wall colors (maybe this could be changed in the next year's competition) we had to use additional sensors.

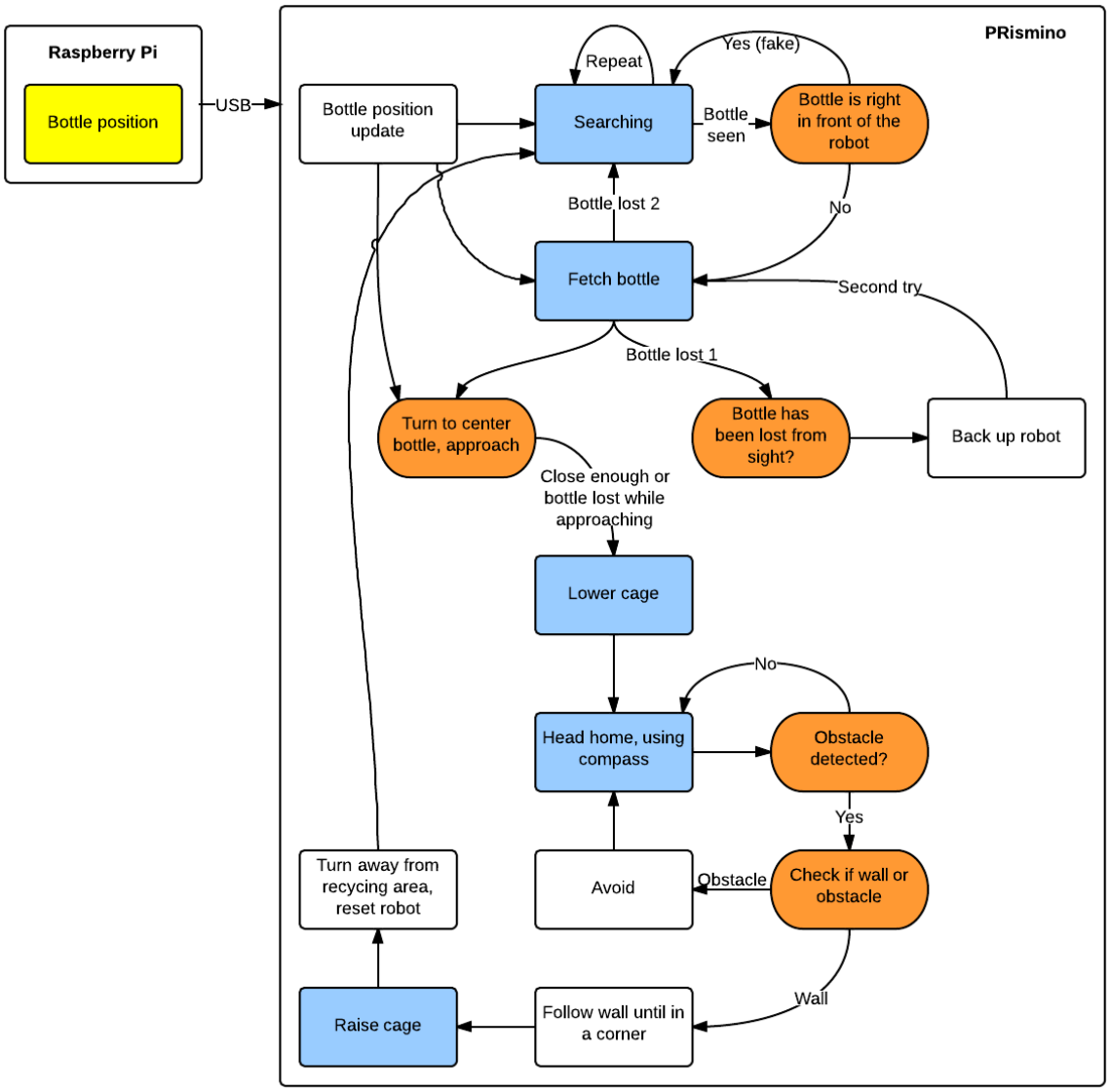



**Figure 15:** Program flowchart as a state machine

# 6 Testing

## 6.1 Navigation

There are several tasks during the navigation, including the avoidance of obstacles, finding the recycling area, and the appropriate time point to grasp or release bottles. And as one of most important sensors for the robot, IR sensors simply returned analog values according to if something was blocking it or not, and we interpreted this on the PRismino and made it avoid obstacles, find a wall when homing and follow a wall when trying to reach the recycling area.

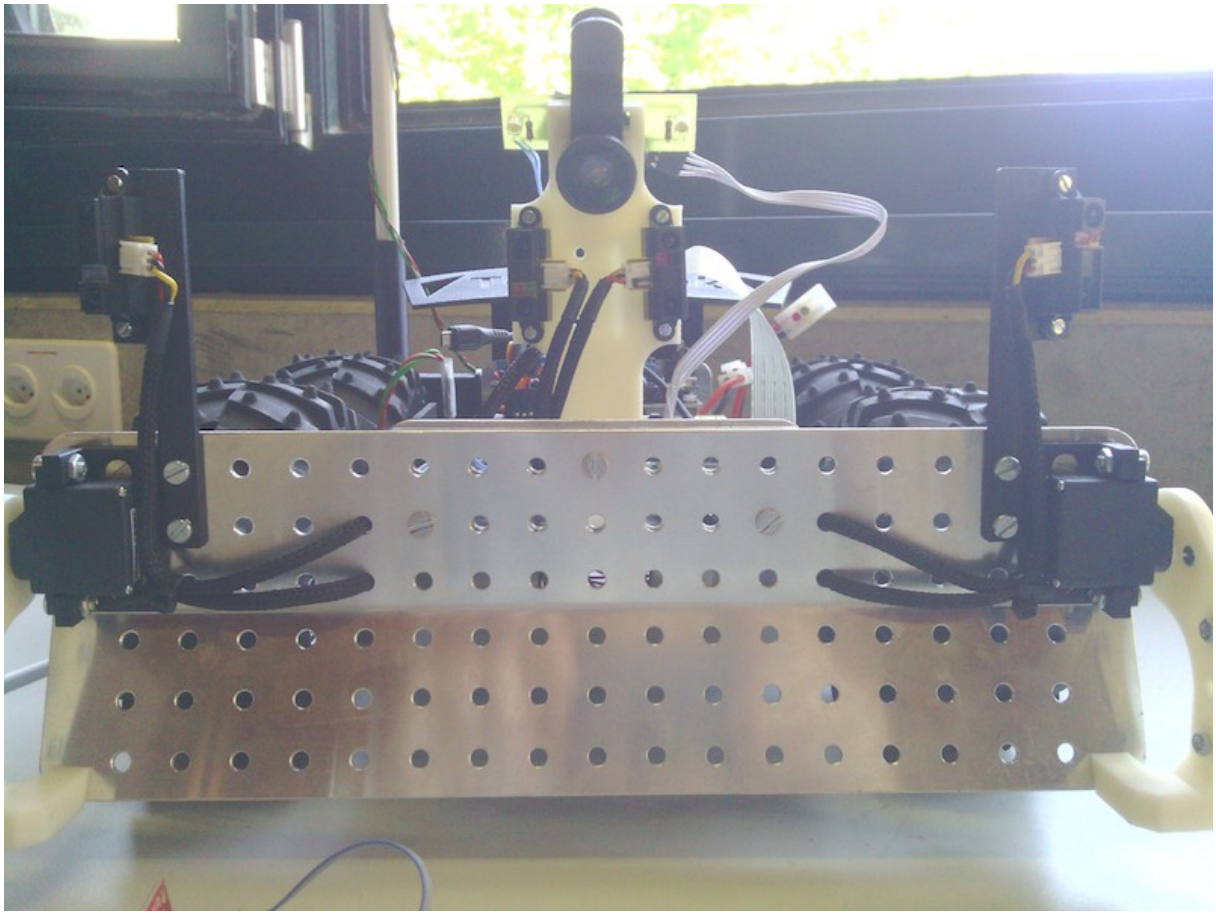


**Figure 16:** Front sensors for navitation and bottle detection, 1 camera and 4 infra-red sensors. Some high-power LEDs besides the camera allow the robot to detect bottles in the dark.

### 6.1.1 Obstacle avoidance

4 Infrared sensors are used for obstacle avoidance, and using some simple thresholding values the robot is able to avoid all bricks and walls inside the arena. The IR sensors are located at a high enough position to only detect bricks. However, this system will also detect bottles which are standing up as the obstacles, and hence will not be picked up by the system. The obstacle avoidance is implemented in the navigation function, and will always be "active" when the robot is moving forward. Hence, when a desired heading is given and an obstacle is in the way, the robot will not follow the desired heading but instead avoid the obstacle.

### 6.1.2 Finding the recycling area

Once a bottle has been collected, the robot uses the compass readout to follow the heading of the base. The robot will start searching for a wall. Each time it encounters an obstacle it will check

if the obstacle is a brick or a Wall. If a wall has been detected, the robot will determine if it has encountered the left or right wall, and if a brick is detected, it will simply continue towards the desired heading while avoiding the obstacle. The robot will then follow the wall until it reaches the recycling area.

We wanted to put the compass right behind the camera for it to be accessible, but it turned out that being so close to the floor affected the compass in unpredictable ways as there were metal bars in the building structure. After having done various tests, it was discovered that if the compass is placed high enough, at least 60 cm above the ground, it will not be affected by the metallic structures. Hence, the compass is placed on a pole which allows to ensure correct heading readouts.

## 6.2 Bottle grasping

The first strategy we had was to use the camera for obstacle and bottle detection, but when we tried in real conditions, the changing lighting conditions made it impossible to make a robust system using the low level image processing we wanted (color thresholding) so we had to abandon this idea. We also wanted to use the camera to detect the yellow beacon, but it was really hard to see it from far away.

Finally we decided that we would only use the camera to report all the bottle positions as it was already a slow algorithm and use lower-level sensors for obstacle avoidance (IR sensors) and homing (compass) with another micro-controller (the PRismino).

Bottle grasping was based on the last year competition group 5 approach. They used the same hardware and bottle detection, and they had created the classifier for the Haar Cascade algorithm to recognise plastic bottles, aluminium cans and glass bottles (which was their goal), which took them 3 days to generate.

We tested it with our script and it worked really well. Instead of reusing their C++ code we wanted to make something different at first, but ended up doing the same approach, that is dedicating bottle detection to a higher level computer and doing lower level computation with a micro-controller. Since the deadline was really close and we had a tested system we stuck with our program.

### 6.2.1 Electronics interfacing

The Raspberry Pi worked with 3.3V and the Rismino at 5V and applying 5V to the Raspberry Pi pins could damage it, then we found an elegant solution of using the USB port of the Raspberry Pi. It turns out the Raspberry Pi can be powered via the USB port with 5V, so we connected the PRismino and the Raspberry Pi via USB, and power was going from the PRismino to the Raspberry Pi, but communication from the Raspberry Pi to the PRismino via the serial connection to send the information about the bottle position.

We could not use I2C as there was an issue of the Raspberry Pi not being able to be set as slave. We needed the PRismino to control the I2C elements like the motor controller and compass, so even if we wanted we couldn't have used this communication method.

### 6.2.2 Programming language selection

When choosing the programming language we had to consider multiple things: computation power needed for image processing algorithms, experience with the language, available libraries, etc. The most important aspect was to get something to work in order to see how the camera performs and then work on optimising and making the code run faster.

After installing the Raspberry Pi camera driver [2] we first tried to make a test program using Python language, and it was relatively easy to implement image processing using OpenCV as there are lots of examples on the Internet and in the OpenCV documentation.

Then we tried some lower level approach with C++, also using OpenCV image processing libraries. The frame rate was better compared to Python, but it used the OpenCV native methods to grab the frames which could be improved on.

Josh Larson has developed a camera API [1] for the Raspberry Pi camera that uses the Multi-Media Abstraction Layer (MMAL) which is a Broadcom API and allows for lower level camera access than OpenCV methods. This improved the frame rate yet more.

### 6.2.3 Benchmarks

We ran some benchmarks to compare the 3 methods we tested in order to get a good idea on the performance gains: C++ using MMAL, C++ with native OpenCV functions and Python using the Picamera package [4].

The tests consisted of running the same image processing algorithms that we considered for the competition and we also compared with or without preview as while programming we needed visual feedback as to see what the camera was doing, but during the competition there won't be a screen and it adds some significant overhead.

The results were interesting as with a small image of 256 by 128 pixels we obtained similar results in frame capture using Python and C++ with MMAL where as native OpenCV functions were much slower. Showing the the preview window made only a difference of about 1 frame per second.

In the end we kept Python as our main programming language because it was much simpler to implement and offered decent speed when doing image processing. Some important aspects such as serial communication with the PRismino and memory management require very tedious work and having to compile the program with C++ every time we needed testing was wasting too much time.

The final program has quite a complex structure using multiple threads for serial communication and image processing. Python offers a very easy way to implement all these features, but has some overhead compared to C++. We made the compromise of having a slower program on the Raspberry Pi than we could have made, but one we knew was tested and reliable enough.

## 6.3 Simulation in Webots

We used Webots to simulate our robot in a virtual environment, identical to the competition arena for the most part. Webots allows to simulate mobile robots and to make a completely custom robot type, and it can even import 3D models from other programs such as SolidWorks. We modeled our robot and the arena to have an idea on the issues that may arise during the competition. It was really helpful to have this tool as we immediately saw some potential problems that would've taken a lot of time to fix later.

The potential problems we saw were the camera position: the robot could not see all the area in front of it and thus some obstacles might remain outside of its field of view, if it were to collide with them it could get stuck. We tried two solutions for this: moving the camera back or use a 180° lens in order to widen the field of view. Cellphone lens' are really cheap nowadays and we got some off of eBay, but the delivery was really slow so thanks to the simulation we were able to order the parts early in the competition and have and test them in time.

Another issue that could have been a problem is the bottle grabbing area in front of the robot: when the bottle was inside and the robot was pushing it tented to roll over the bottles, because of

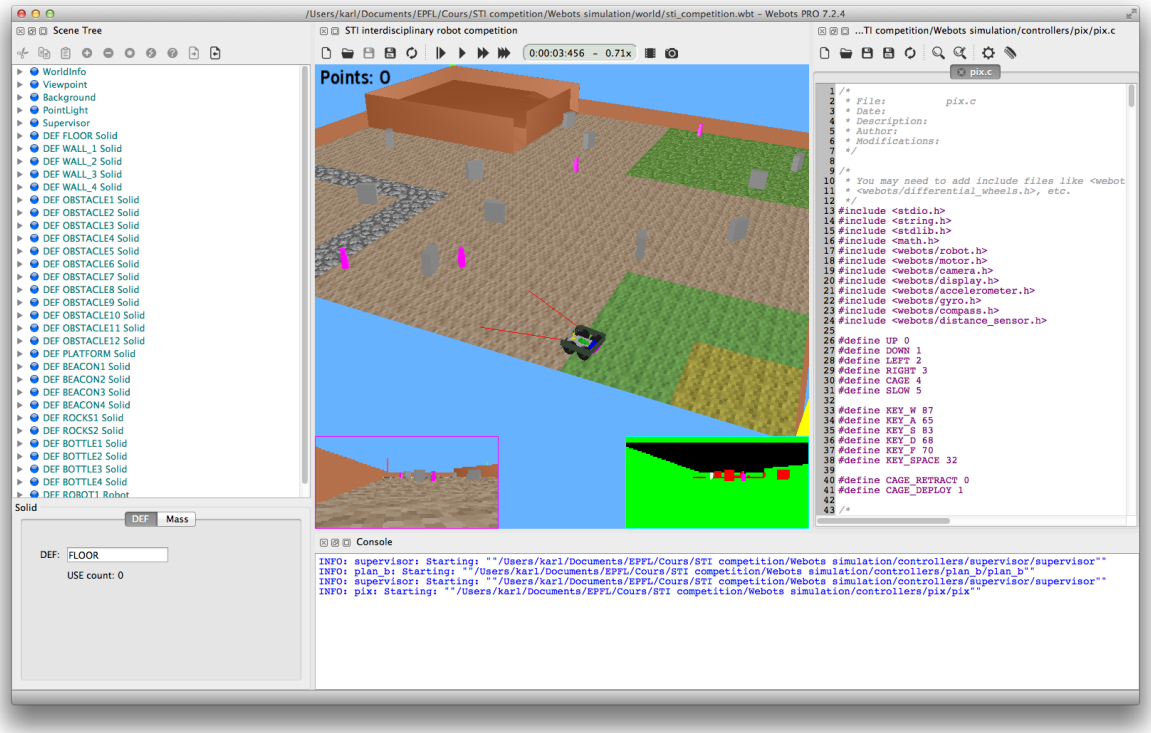



**Figure 17:** Webots simulation with the robot programmed to detect walls and bottles with a simplistic color based detection. On the bottom left is the camera output, on the bottom right is the processed image.

**Figure 18:** 180 degrees lens bought from eBay to augment the camera's field of vision

the WildThumper's big wheels, good adherence and powerful motors it simply might have rolled over the bottle it was delivering to the recycling area. Adding a simple plow in front of the robot immediately fixed that.

The Webots simulation was by no means perfect, since it ran on a much more powerful computer than what was going to be on the robot itself so image processing was much faster in simulation. Colors and lighting conditions were absolutely perfect, and something that was going to be a problem during the competition. Physics were not always true to what was going to be on the arena, tuning all the parameters to get a near-perfect simulation would've taken too much time.

We do recommend making a simulation at the beginning for the next year competitors, an easily

simulated robot can be actually made just as easily and much of the implementation problems can be seen in advance. Webots is quite well documented [5] and only takes a couple of hours to master.

# 7 Results

The robot performed really well during the competition, and it won with 45 points bringing back 4 bottles from the main area to the 100% zone and leaving one in the 50% zone. At first it had a problem when closing the cage as the bottles were not perfectly aligned. It also lost a bottle while trying to bring it back to the recycling zone, because the robot tried passing though the rocks.

As expected it detected some false positives, but our countermeasures worked really well and it by double checking it never brought back a "virtual fake" bottle.

Some areas were really difficult to predict, especially the rocks and the area near the ramp. Near rocks the camera picked up a lot of false positives (but didn't bring them back) and near the ramp it simply went crazy and couldn't get out of a loop, even with all our countermeasures. Fortunately during the competition it didn't go near the ramp.

On grass the IR sensors had tendency to pick up the grass as an obstacle sometimes, putting them higher up might solve this issue.

Our state machine code that made the robot play a sound every time it changed states, which made it really easy to know what the robot was doing during the competition, when it searched for bottles and found one we knew right away that it had found one and was in the next state of trying to grab it. This was also really useful when debugging.

We are quite satisfied with our controller, lots of code was written the day before competition in order to solve the particular issues the robot had, for example we noticed that when the robot detected a false positive bottle right in front of it for some reason it tried to bring it back, we assumed a bottle cannot appear in front of the robot magically and made it consider it a false positive in this particular case (bottle appearing right in the pick-up zone). When it had to move towards the bottle it had to detect the bottle multiple times which ensured that bottle was a genuine one. Another issue we solved at the last minute was that the robot could not turn on itself after grabbing a bottle and orienting itself towards the goal as sometimes an obstacle was right out of its field of vision, special cases on IR sensors when detecting obstacles were implemented to avoid damaging the robot.

Much more could been done if we had time, we could probably detect the rocks somehow in order to avoid that area, add more special cases to minimise the chance of coming back empty handed, such as a final check on the bottle when the cage was closed, but we thought with such short notice we could lose more bottles than win... it was a compromise between making the code even more complex and trusting the program we had already tested and approved.

We got enough time to test our robot on the real arena, but if we had more time on it we could have probably made an even better controller. Having the arena in advance and being able to test on it one week before in the real competition setting really helped us to understand where we had to put more effort in order for our robot to work.

# 8 Conclusion

This project was a very difficult one, we learned to apply the things we were taught in some courses and we had to design, develop, program and test a robot from scratch.

We wanted to make the most simple robot we could think of and reuse parts from the last years competition in order not to wait for parts deliveries. We ended up with a more complex robot than we needed, but we still succeeded in building it and making it work.

Out of the 2000CHF we were given, we used about 1000CHF, so we could've made a second robot, but already making one took so much time and energy and since one of the most important part on the robot was the software we decided to invest our time in the programming instead of building multiple robot.

We knew that during the last year competition some groups had problems with broken motors so we took special care when choosing ours, we always used them within specifications but we still managed to break all 4 motors that were supposedly impossible to break the way we used them. The lessons we learned from this is: do not buy cheap products if you want them to work, especially Pololu products, we have learned to never trust motors sold by this company or cheap electric motors altogether. Since it's a critical part of the robot it's important to invest what is needed and not take the cheap and unreliable stuff.

For the next year's competition we recommend doing a simulation (using Webots or another robot simulation software) when the main idea is found and while the parts arrive. This way it's possible to work on the software even before the robot is completed and it allows to see some potential problems the robot might have. We also recommend a modular design, we were able to change our strategy 2 days before the competition because we had such an easily modifiable platform.

We found out that we know a lot about the theory of building a robot, but in practice theory is not applicable. The most demanding part was, of course, the testing and finding out what could go wrong, this needed testing on the real arena and we had to fix problems fast as we only had a couple days to do it.

We had a great time building the robot and thanks to all the friendly "competition" we got to share our ideas, help out and learn from each other, see different approaches for the same problem and learn to work together.

## References

[1] Josh Larson *Camera Board API*
https://github.com/Josh-Larson/CameraBoardAPI

[2] Linux Projects *How to install or upgrade UV4L on Raspbian (for the Raspberry Pi)*
http://www.linux-projects.org/modules/sections/index.php?op=viewarticle&artid=14

[3] OpenCV *Documentation*
http://docs.opencv.org

[4] Picamera *Documentation*
http://picamera.readthedocs.org/en/release-1.4/

[5] Webots *Tutorial 6: 4-Wheels Robot*
http://cyberboticspc1.epfl.ch/cdrom/common/doc/webots/guide/section7.7.html#tutorial_4_wheels_lowlevel

[6] David Hamp-Gonsalves *OpenCV/Python Color Tracking*
http://www.davidhampgonsalves.com/opencv-python-color-tracking

[7] pySerial *Documentation*
http://pyserial.sourceforge.net/

[8] Achu's TechBlog *Object detection in OpenCV using Haartraining*
http://achuwilson.wordpress.com/2011/02/13/object-detection-using-opencv-using-haartraining/

[9] All about openCV *Creating a haar cascade classifier aka haar training*
http://opencvuser.blogspot.ch/2011/08/creating-haar-cascade-classifier-aka.html

[10] Computer Vision Software *FAQ: OpenCV Haartraining*
http://www.computer-vision-software.com/blog/2009/11/faq-opencv-haartraining/comment-page-1/

[11] Naotoshi Seo *Tutorial: OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features)*
http://note.sonots.com/SciSoftware/haartraining.html

[12] DAGU product support *Understanding the Wild Thumper controller*
https://sites.google.com/site/daguproducts/home/tutorials/understanding-wild-thumper

[13] Let's make robots *Wild Thumper Robot Controller*
http://letsmakerobots.com/node/21598

# A Gantt diagram

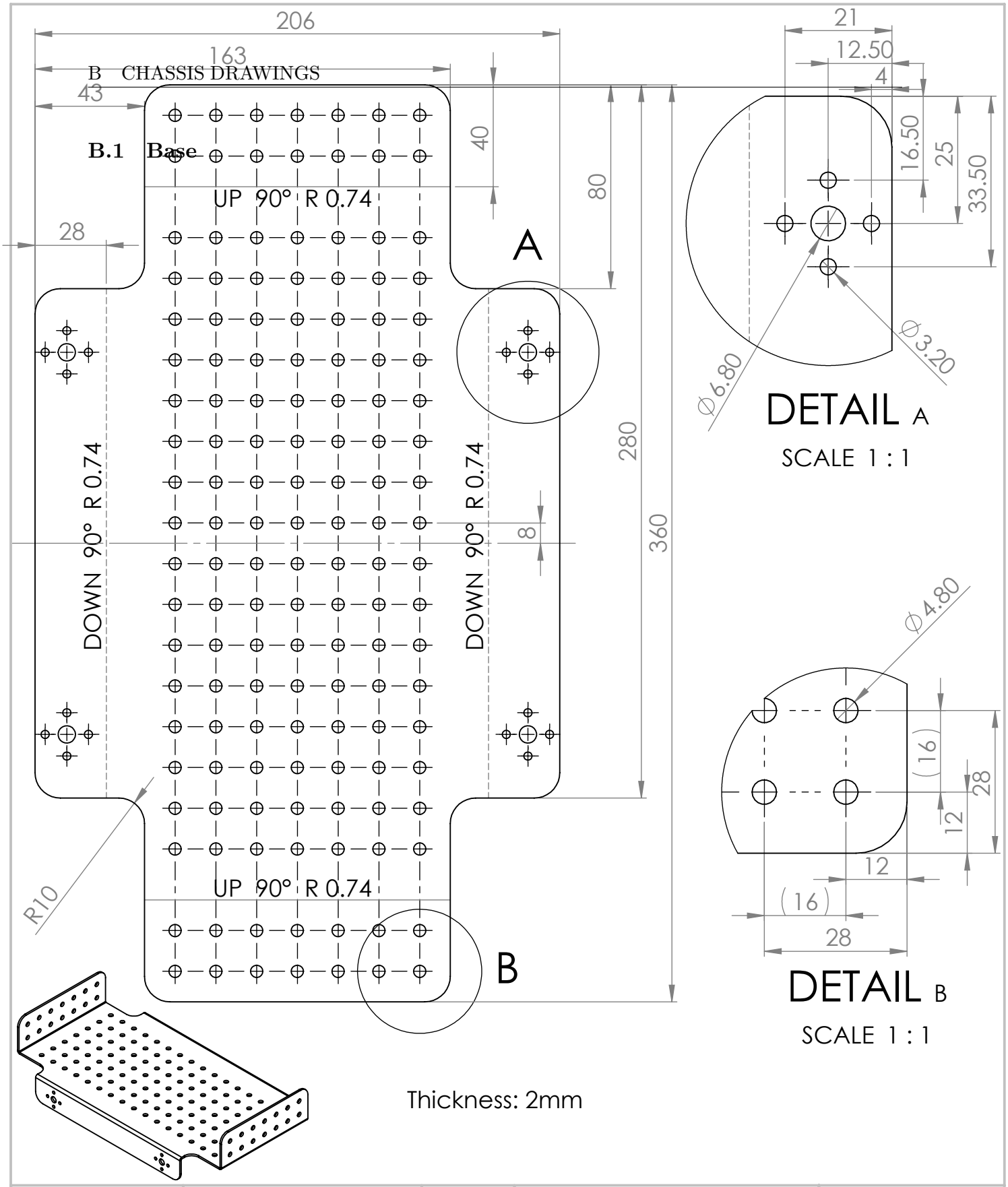
| | Name | Duration | Start | Finish |
|---|---|---|---|---|
| 1 | Function analysis and solutions identification. | 1w | 20/02/2014 | 27/02/2014 |
| 2 | Function analysis and solutions refinement. | 1w | 27/02/2014 | 06/03/2014 |
| 3 | MS1 presentation preparation | 1w | 27/02/2014 | 06/03/2014 |
| 4 | MS1 presentation | 0d | 06/03/2014 | 06/03/2014 |
| 5 | MS1 | 0d | 06/03/2014 | 06/03/2014 |
| 6 | MS2 presentation preparation | 1w | 06/03/2014 | 13/03/2014 |
| 7 | MS2 presentation | 0d | 13/03/2014 | 13/03/2014 |
| 8 | MS2 | 0d | 13/03/2014 | 13/03/2014 |
| 9 | Development of key functions individually. Refinement of design. | 2w | 13/03/2014 | 27/03/2014 |
| 10 | Feasibility study | 2w | 13/03/2014 | 27/03/2014 |
| 11 | MS3 presentation preparation | 1w | 27/03/2014 | 03/04/2014 |
| 12 | MS3 presentation | 0d | 03/04/2014 | 03/04/2014 |
| 13 | MS3 | 0d | 03/04/2014 | 03/04/2014 |
| 14 | Development and production according to planning | 6w | 03/04/2014 | 15/05/2014 |
| 15 | **Software development** | 6w? | **03/04/2014** | **15/05/2014** |
| 16 | Bottle detection algorithm | 30d? | 03/04/2014 | 15/05/2014 |
| 17 | Object and obstacle detection algorithm | 30d? | 03/04/2014 | 15/05/2014 |
| 18 | Motor controller programming (H-bridge) | 6d? | 03/04/2014 | 11/04/2014 |
| 19 | **Hardware development** | 3w? | **03/04/2014** | **24/04/2014** |
| 20 | New 4WD chassis (custom made) | 15d? | 03/04/2014 | 24/04/2014 |
| 21 | Component sourcing | 15d? | 03/04/2014 | 24/04/2014 |
| 22 | Deployable cage | 15d? | 03/04/2014 | 24/04/2014 |
| 23 | Final assembly | 2w | 01/05/2014 | 15/05/2014 |
| 24 | Trials and optimisation | 2w | 15/05/2014 | 29/05/2014 |
| 25 | Testing in real conditions on the arena | 8d | 30/05/2014 | 11/06/2014 |
| 26 | Rehearsal competition | 1d | 05/06/2014 | 06/06/2014 |
| 27 | Competition | 1d | 10/06/2014 | 11/06/2014 |
| 28 | MS4 presentation preparation | 1w | 29/05/2014 | 05/06/2014 |
| 29 | MS4 presentation | 0d | 05/06/2014 | 05/06/2014 |
| 30 | MS4 | 0d | 05/06/2014 | 05/06/2014 |
| 31 | Report writing | 3m | 19/03/2014 | 11/06/2014 |




**Figure 19:** Gantt Diagram

# B Chassis drawings

## B.1 Base







DETAIL A

SCALE 1 : 1




DETAIL B

SCALE 1 : 1



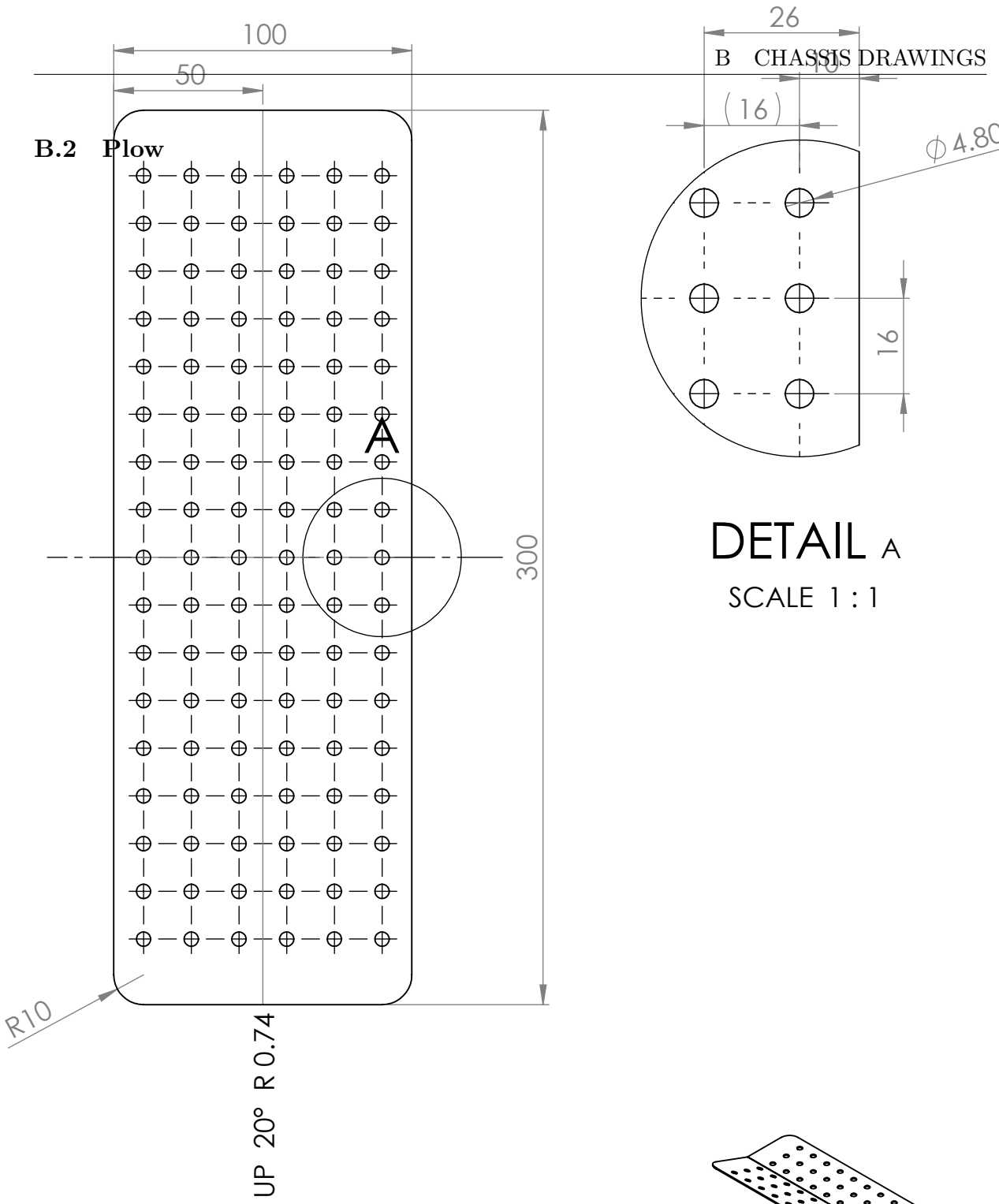
Thickness: 2mm

| | | | | | | |
|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR: | FINISH: | DEBUR AND BREAK SHARP EDGES | DO NOT SCALE DRAWING | REVISION 0 | | |
| | | | Group 5 | | | |
| | NAME | SIGNATURE | DATE | TITLE: | | |
| DRAWN | Karl Kangur | | 2014-04-14 | Base | | |
| CHK'D | | | | | | |
| APPV'D | | | | | | |
| MFG | | | | | | |
| Q.A | | | MATERIAL: Al, 2mm | DWG NO. P01 | | A4 |
| | | | WEIGHT: | SCALE:1:2 | SHEET 1 OF 1 | |



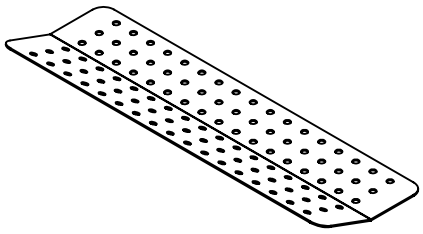
K. Kangur, M. Starein, C. Xie

## B.2 Plow




DETAIL A

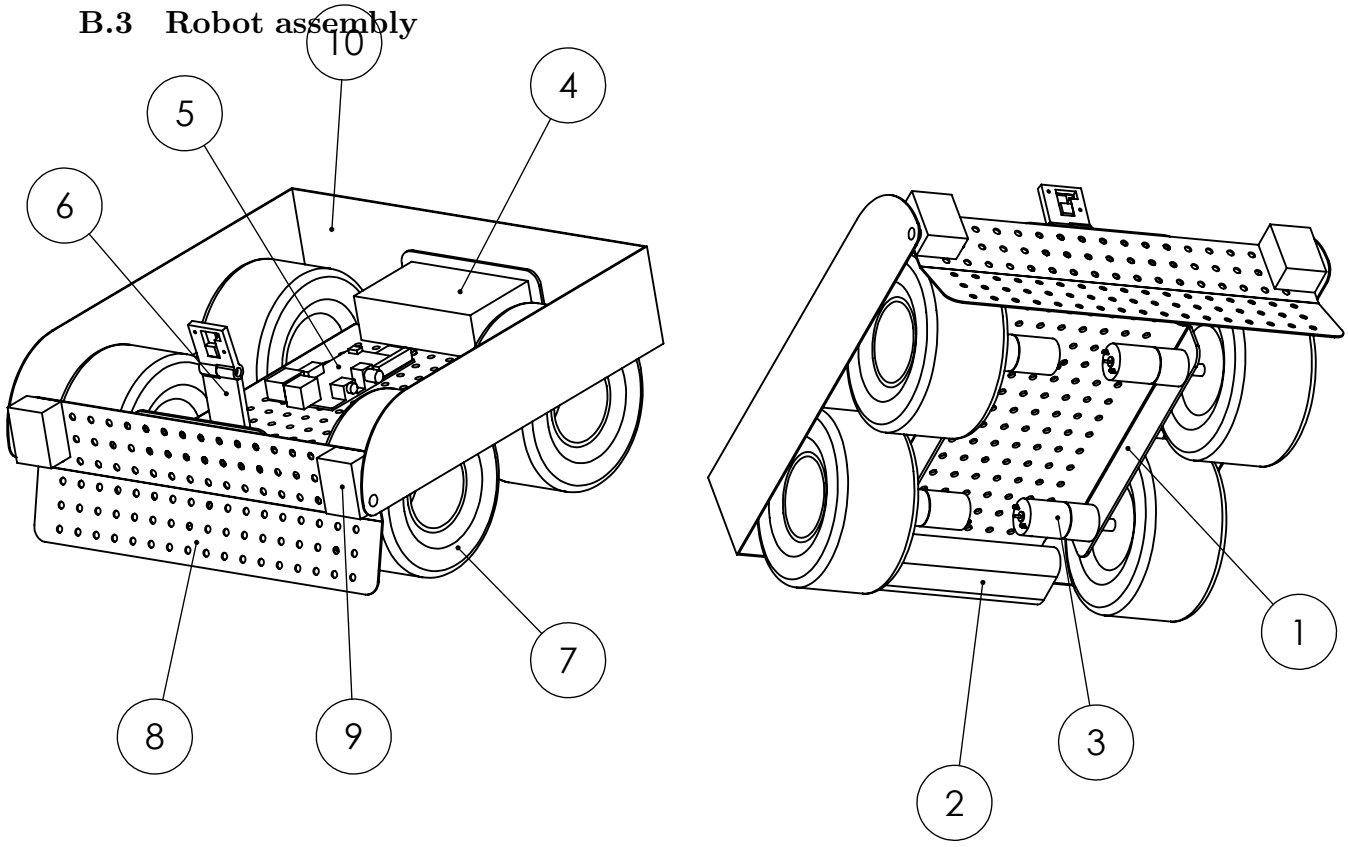
SCALE 1 : 1



Thickness: 2mm

| | | | | | | |
|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | FINISH: | | DEBUR AND BREAK SHARP EDGES | | DO NOT SCALE DRAWING | REVISION |
| | | | | | Group 5 | |
| | NAME | SIGNATURE | DATE | | TITLE: | |
| DRAWN | Karl Kangur | | 2014-04-03 | | Plow | |
| CHK'D | | | | | | |
| APPV'D | | | | | | |
| MFG | | | | | | |
| Q.A | | | | MATERIAL: Al, 2mm | DWG NO. P02 | A4 |
| | | | | | | |
| | | | | WEIGHT: | SCALE:1:2 | SHEET 1 OF 1 |

### B.3 Robot assembly




| ITEM NO. | PART NUMBER | DESCRIPTION | Défaut/QTY. |
|---|---|---|---|
| 1 | base | Robot chassis | 1 |
| 2 | battery | Battery pack (7.2V, 3000mAh) | 1 |
| 3 | motor | WildThumper motor, 34:1 | 4 |
| 4 | controller | WildThumper motor controller | 1 |
| 5 | raspberrypi | On-board computer | 1 |
| 6 | assembly | Raspberry Pi camera | 1 |
| 7 | wheel | WildThumper wheel | 4 |
| 8 | plow | Front plow | 1 |
| 9 | servomotor | Standard servomotor | 2 |
| 10 | cage | Deployable bottle cage | 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | FINISH: | | DEBUR AND BREAK SHARP EDGES | | DO NOT SCALE DRAWING | REVISION |
| | | | | | Group 5 | |
| | NAME | SIGNATURE | DATE | | TITLE: | |
| DRAWN | Karl Kangur | | 2014-04-04 | | Fei | |
| CHK'D | | | | | | |
| APPV'D | | | | | | |
| MFG | | | | | | |
| Q.A | | | | MATERIAL: | DWG NO. A01 | A4 |
| | | | | | | |
| | | | | WEIGHT: | SCALE:1:5 | SHEET 1 OF 1 |

# C Source code

## C.1 Controller source code in Python

```
#!/usr/bin/env python
import time
import sys
from cv2 import waitKey

import comm
import camera

# types of elements to detect

class RobotControl:
  def __init__(self, port, baudrate):
    # configuration for the serial connection to the robot
    self.port = port
    self.baudrate = baudrate

  def connect(self):
        # start serial communication thread
    try:
      print "Connecting to robot"
      self.commThread = comm.CommThread(self.port, self.baudrate)
      self.commThread.start()
    except:
      print "Could not connect to robot"
      return

    preview = False
    # run the code with or without the preview, by default it's disabled
    if len(sys.argv) > 1:
      if sys.argv[1]:
        preview = True

    # start the camera thread
    try:
      print "Starting camera"
      self.cameraThread = camera.ImageProcessing(256, 128, preview)
      self.cameraThread.start()
    except:
      print "Could not start camera"
      return

    # start the control loop
    self.control()

  def control(self):
    # wait for the camera and communication threads to start
    while not self.commThread.isAlive() and not self.cameraThread.isAlive():
      time.sleep(0.1)
      pass

    print "Starting robot control loop"

    # play 880hz sound to indicate scropt start when working without a screen
    print "Beep"
    self.commThread.write('t' + chr(880 >> 8) + chr(880 & 0xff))

    # turn on the led lights on the robot
    print "Turning light on"
```

```
59      self.commThread.write('l' + chr(1))
60
61      print "Enable bottle detection"
62      self.commThread.write('0')
63
64      while True:
65        try:
66          self.stateSearching()
67
68          # check for keyboard input
69          if self.checkInput():
70            break
71
72          time.sleep(0.1)
73        except (KeyboardInterrupt, SystemExit):
74          print "User forced exit"
75          break
76        except Exception as e:
77          # when an error occurs make sure to end the dependant threads
78          print e
79          self.stopThreads()
80          break
81
82      # turn off the led lights on the robot
83      print "Turning light off"
84      self.commThread.write('l' + chr(0))
85
86      print "Disable bottle detection"
87      self.commThread.write('0')
88
89      print "Control loop exited"
90      self.stopThreads()
91
92    def stopThreads(self):
93      print "Stopping threads"
94      self.commThread.stop()
95      self.commThread.join()
96      self.cameraThread.stop()
97      self.cameraThread.join()
98
99    def checkInput(self):
100     # check for user input
101     key = waitKey(10) & 0xff
102
103     if key == 27:
104       print "Manually stopped"
105       return True
106     elif key == ord('w'):
107       self.commThread.write('s' + chr(self.maxSpeed) + chr(self.maxSpeed))
108     elif key == ord('a'):
109       self.commThread.write('s' + chr(-self.maxSpeed + 255) + chr(self.maxSpeed))
110     elif key == ord('d'):
111       self.commThread.write('s' + chr(self.maxSpeed) + chr(-self.maxSpeed + 255))
112     elif key == ord('s'):
113       self.commThread.write('s' + chr(-self.maxSpeed + 255) + chr(-self.maxSpeed +
              255))
114     elif key == ord('1'):
115       self.commThread.write('u')
116     elif key == ord('2'):
117       self.commThread.write('d')
118     elif key == ord('q') or key == 32:
119       self.commThread.write('s' + chr(0) + chr(0))
```

```
120     elif key != 255:
121       print key
122
123     return False
124
125   def stateSearching(self):
126     data = self.cameraThread.checkDataQueue()
127     if data:
128       self.commThread.write("!" + chr(data["position"][0] + chr(data["position"][1])
              )
129       print "Bottle detected", data["position"]
130
131 robot = RobotControl("/dev/ttyACM0", 9600)
132 robot.connect()
```

**Listing 1:** Python main program

```
1  #!/usr/bin/env python
2  import sys
3  import serial
4  import threading
5  import time
6  import socket
7
8  class CommThread(threading.Thread):
9      def __init__(self, port, baudrate):
10         # script crashes without this
11         super(CommThread, self).__init__()
12         # save handle
13         self.comm = serial.Serial(port=port, baudrate=baudrate, timeout=1)
14         # an event that can be used for anything, here used to stop the thread
15         self.running = threading.Event()
16
17     def write(self, data):
18         # send data to the device
19         try:
20             return self.comm.write(data)
21         except:
22             return False
23
24     def setSpeed(self, speedLeft, speedRight):
25         # must be unsigned value
26         if speedLeft < 0:
27             speedLeft += 255
28
29         if speedRight < 0:
30             speedRight += 255
31
32         return self.write('s' + chr(speedLeft) + chr(speedRight))
33
34     def run(self):
35         print "Communication thread started"
36         # make sure we're still running
37         while not self.running.isSet():
38             # see if there is something in the incoming buffer
39             if self.comm.inWaiting():
40                 self.process(self.comm.readline())
41
42             # wait for a bit or it will consume all the CPU
43             time.sleep(0.2)
44
45     def process(self, data):
46         if data == 'p':
```

```
47              s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
48              # create a socket to an external website
49              s.connect(('google.com', 0))
50              ip = s.getsockname()[0].split(".")
51              # transform list of strings to list of integers
52              ip = map(int, ip)
53              # send the external ip address via serial to the arduino board
54              self.write(chr(ip[0]) + chr(ip[1]) + chr(ip[2]) + chr(ip[3]))
55
56      def stop(self):
57          print "Stopping communication thread"
58          self.running.set()
```

**Listing 2:** Communication thread

```
1  #!/usr/bin/env python
2  import io
3  import picamera
4  import threading
5  import cv2
6  import numpy as np
7  import Queue
8
9  # types of elements to detect
10 OBSTACLE, BOTTLE = range(2)
11
12 class ImageProcessing(threading.Thread):
13   def __init__(self, width, height, preview):
14     super(ImageProcessing, self).__init__()
15
16     self.obstacleThreshold = 100000
17
18     self.preview = preview
19
20     self.width = width
21     self.height = height
22
23     self.camera = picamera.PiCamera()
24
25     self.camera.resolution = (self.width, self.height)
26
27     # boost colors
28     self.camera.saturation = 100
29
30     #self.camera.shutter_speed = 10000
31     #self.camera.awb_mode = u'off'
32     #self.camera.exposure_mode = u'fixedfps'
33     #self.camera.meter_mode = u'spot'
34     #self.camera.exposure_compensation = 10
35     #self.camera.framerate = 2
36     #self.camera.sharpness = 0
37     #self.camera.video_stabilization = True
38
39     #self.configureCamera()
40
41     # load the xml file for
42     self.cascadeXml = cv2.CascadeClassifier('bottle.xml')
43
44     # load the mask that hides the non important parts of the image
45     self.mask = cv2.imread("mask.png", 0)
46
47     self.stream = io.BytesIO()
48     self.dataQueue = Queue.Queue()
```

```
49
50      # if enabled show the preview window
51      if self.preview:
52        cv2.namedWindow("Preview", flags=cv2.cv.CV_WINDOW_AUTOSIZE)
53
54      # event to stop the thread
55      self.running = threading.Event()
56
57    def run(self):
58      print "Image processing thread started"
59      # make sure we're still running
60      while not self.running.isSet():
61        # take the picture
62        self.camera.capture(self.stream, format='jpeg', use_video_port=True)
63
64        # construct a numpy array from the stream
65        data = np.fromstring(self.stream.getvalue(), dtype=np.uint8)
66
67        self.stream.truncate()
68        self.stream.seek(0)
69
70        # "decode" the image from the array, preserving colour in BGR format
71        self.img = cv2.imdecode(data, cv2.CV_LOAD_IMAGE_COLOR)
72
73        # image that can be modified by the image processing for preview purposes
74        if self.preview:
75          self.previewImage = self.img
76
77        #self.detectObstacles()
78        self.detectBottles()
79
80        if self.preview:
81          try:
82            cv2.imshow("Preview", self.previewImage)
83          except:
84            pass
85
86      # destroy the preview window
87      if self.preview:
88        cv2.destroyAllWindows()
89
90    def stop(self):
91      print "Stopping image processing thread"
92      self.running.set()
93
94    def detectBottles(self):
95      # get a gray picture
96      gry = cv2.cvtColor(self.img, cv2.COLOR_BGR2GRAY)
97
98      # run the haar cascade algorithm (slow)
99      haar = self.cascadeXml.detectMultiScale(gry, 1.1, 6)
100
101     # update preview
102     if self.preview:
103       for (x,y,w,h) in haar:
104         cv2.rectangle(self.previewImage, (x,y), (x+w, y+h), (255, 255, 255), 1)
105
106     # normalised value between 0 and 1 of the first bottle position
107     if len(haar):
108       bottleX = (haar[0][0] + haar[0][2] / 2);
109       bottleY = self.height - (haar[0][1] + haar[0][3] / 2) ;
110       self.dataQueue.put({"type": BOTTLE, "position": (bottleX, bottleY)})
```

```
111
112   def detectObstacles(self):
113     colorFloorL = np.array([8, 30, 30], np.uint8)
114     colorFloorH = np.array([28, 255, 255], np.uint8)
115
116     # convert to hue, saturation, value format
117     hsv = cv2.cvtColor(self.img, cv2.COLOR_BGR2HSV)
118
119     colorThreshold = cv2.inRange(hsv, colorFloorL, colorFloorH)
120     colorThreshold = np.invert(colorThreshold)
121
122     cv2.erode(colorThreshold, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7)),
          colorThreshold, (-1, -1), 1)
123     np.bitwise_and(colorThreshold, self.mask, colorThreshold)
124
125     # average values vertically and split screen in 4 horizontally
126     sumVertical = np.sum(colorThreshold, axis = 0)
127     splitHorizontal = np.array([np.sum(sumVertical[0:63]), np.sum(sumVertical
          [64:127]), np.sum(sumVertical[128:191]), np.sum(sumVertical[192:255])])
128     # print "H", splitHorizontal
129
130     if self.preview:
131       # show obstacle position with the overlay on the image
132       mask = np.invert(colorThreshold)
133       np.bitwise_and(self.previewImage[:,:,0], mask, self.previewImage[:,:,0])
134       np.bitwise_and(self.previewImage[:,:,1], mask, self.previewImage[:,:,1])
135       np.bitwise_and(self.previewImage[:,:,2], mask, self.previewImage[:,:,2])
136       pass
137
138     peakX = splitHorizontal.argmax()
139
140     # obstacle threshold has been reached, obstacle has been detected
141     if splitHorizontal[peakX] > self.obstacleThreshold:
142
143       # average values horizontally and split screen in 4 vertically
144       sumHorizontal = np.sum(colorThreshold, axis = 1)
145       splitVertical = np.array([np.sum(sumHorizontal[0:31]), np.sum(sumHorizontal
            [32:63]), np.sum(sumHorizontal[64:95]), np.sum(sumHorizontal[96:127])])
146       #print "V", splitVertical
147
148       peakY = splitVertical.argmax()
149
150       self.dataQueue.put({"type": OBSTACLE, "position": (peakX, peakY)})
151
152   def checkDataQueue(self):
153     if not self.dataQueue.empty():
154       return self.dataQueue.get()
155     else:
156       return False
```

**Listing 3:** Image processing thread

## C.2 PRismino source code

```
1  /*
       ********************************************************************************
2   *
3   * Title:       STI competition Arduino code for the Tokamak robot.
4   * Date:        2014-06-06
5   *
```

```
6  ********************************************************************************
       */
7  #include <Servo.h>
8  #include <prismino.h>
9  #include <Wire.h>
10 #include "robot.h"
11
12 Tokamak robot;
13
14 // transition structure
15 struct transition
16 {
17     enum state_codes state_source;
18     enum return_codes return_code;
19     enum state_codes state_destination;
20 };
21
22 // state functions and codes must be in sync
23 return_codes (*state[])(void) = {
24   stateSearching,
25   stateFetchingBottle,
26   stateLowerCage,
27   stateGoHome,
28   stateRaiseCage
29 };
30
31 struct transition state_transitions[] = {
32   {STATE_SEARCHING,       OK,     STATE_FETCHING_BOTTLE},
33   {STATE_SEARCHING,       REPEAT, STATE_SEARCHING},
34   {STATE_FETCHING_BOTTLE, OK,     STATE_LOWER_CAGE},
35   {STATE_FETCHING_BOTTLE, REPEAT, STATE_FETCHING_BOTTLE},
36   {STATE_FETCHING_BOTTLE, FAIL,   STATE_SEARCHING},
37   {STATE_LOWER_CAGE,      OK,     STATE_GO_HOME},
38   {STATE_GO_HOME,         REPEAT, STATE_GO_HOME},
39   {STATE_GO_HOME,         OK,     STATE_RAISE_CAGE},
40   {STATE_RAISE_CAGE,      OK,     STATE_SEARCHING}
41 };
42
43 enum state_codes currentState;
44 enum return_codes returnCode;
45 comm_methods inputMethod;
46
47 // pointer to the current called function in the state machine
48 return_codes (*stateFunction)(void);
49
50 // other global variables
51 volatile uint8_t bottlePosition;
52 volatile uint8_t bottleDistance;
53 sides sideWall;
54
55 volatile uint32_t timeBottleLastSeen;
56 volatile uint32_t timeNextBottleCheck;
57 uint32_t timeCheckBattery;
58 volatile boolean booleanBottleUpdate;
59 boolean booleanBottleSecondTry;
60 boolean booleanFalseBottle;
61
62 // ############################################################ SETUP
63
64 void setup()
65 {
66   // set pin output mode (sources current)
```

```
67    pinMode(LED, OUTPUT);
68    pinMode(PIN_LIGHTS, OUTPUT);
69
70    // enable button pull-up
71    pinMode(BTN, INPUT);
72    digitalWrite(BTN, HIGH);
73
74    // play a sound on boot, repeated sounds will indicate very low battery voltage
75    robot.playSound(ONEUP);
76
77    // initialise serial bus for communication with the Raspberry Pi
78    Serial.begin(9600);
79
80    // initialise the bus for communication with the computer via Bluetooth
81    #ifdef ENABLE_BLUETOOTH
82    Bluetooth.begin(9600);
83    #endif
84
85    // join i2c bus as master
86    Wire.begin();
87    // disable internal pull-ups to 5V as there are external 2K pull-ups to 3.3V
88    digitalWrite(SDA, LOW);
89    digitalWrite(SCL, LOW);
90
91    #ifdef ENABLE_COMPASS
92    // wait for the imu to boot
93    delay(500);
94    // put the HMC5883 IC into the correct operating mode
95    // open communication with HMC5883
96    Wire.beginTransmission(I2C_COMPASS_ADDRESS);
97    // select mode register
98    Wire.write(0x02);
99    // continuous measurement mode
100   Wire.write(0x00);
101   Wire.endTransmission();
102   #endif
103
104   // initialise global variables
105   bottlePosition = 0;
106   bottleDistance = 0;
107
108   timeCheckBattery = millis() + TIME_CHECK_BATTERY;
109   timeBottleLastSeen = 0;
110
111   sideWall = RIGHT;
112   booleanBottleUpdate = 0;
113
114   // reset the robot state
115   reset();
116
117   currentState = ENTRY_STATE;
118 }
119
120 // ############################################################## MAIN LOOP
121
122 void loop()
123 {
124   if(Serial.available())
125   {
126     processInput(USB);
127   }
128
```

```
129   #ifdef ENABLE_BLUETOOTH
130   if(Bluetooth.available())
131   {
132     processInput(BLUETOOTH);
133   }
134   #endif
135
136   uint16_t irLeft, irCenterLeft, irCenterRight, irRight;
137   robot.readIrSensors(&irLeft, &irCenterLeft, &irCenterRight, &irRight);
138
139   Bluetooth.print(irLeft);
140   Bluetooth.print("\t");
141   Bluetooth.print(irCenterLeft);
142   Bluetooth.print("\t");
143   Bluetooth.print(irCenterRight);
144   Bluetooth.print("\t");
145   Bluetooth.println(irRight);
146
147   //Bluetooth.println(robot.getHeading());
148
149   // toggle robot running state via the button on the shield
150   if(!digitalRead(BTN))
151   {
152     robot.playSound(POWERUP);
153     robot.flags.running = !robot.flags.running;
154
155     // robot has been stopped
156     if(!robot.flags.running)
157     {
158       robot.stop();
159       // reset robot state to ENTRY_STATE
160       currentState = ENTRY_STATE;
161     }
162
163     // wait for button debounce
164     delay(1000);
165   }
166
167   #ifdef ENABLE_CONTROLLER
168   // check battery level and make a beep if it's too low
169   if(millis() > timeCheckBattery)
170   {
171     robot.checkBattery();
172   }
173   #endif
174
175   if(robot.flags.running)
176   {
177     // current function to call according to the state machine
178     stateFunction = state[currentState];
179     // actually call the function
180     returnCode = stateFunction();
181     // fetch next state
182     currentState = lookupTransitions(currentState, returnCode);
183   }
184 }
185
186 // ############################################################ STATE TRANSITIONS
187
188 // returns the new state according to the current state and the return value
189 state_codes lookupTransitions(state_codes state, return_codes code)
190 {
```

```
191   uint8_t i;
192   // default return state is the entry state
193   state_codes nextState = ENTRY_STATE;
194   // see if a state transition matches and switch to the next state
195   for(i = 0; i < sizeof(state_transitions) / sizeof(transition); i++)
196   {
197     if(state_transitions[i].state_source == state && state_transitions[i].
          return_code == code)
198     {
199       nextState = state_transitions[i].state_destination;
200       if(nextState != state_transitions[i].state_source)
201       {
202         robot.playSound(COIN);
203       }
204       break;
205     }
206   }
207   return nextState;
208 }
209
210 // ############################################################ ROBOT STATES
211
212 return_codes stateSearching()
213 {
214   // check if a bottle has been seen and change state
215   if(bottlePosition && millis() > timeNextBottleCheck)
216   {
217     robot.stop();
218     booleanBottleSecondTry = 0;
219     booleanFalseBottle = 1;
220     return OK;
221   }
222
223   // make robot roam the arena set deviation to 0 to go straight when there are no
        obstacles
224   robot.headTo(0);
225
226   return REPEAT;
227 }
228
229 return_codes stateFetchingBottle()
230 {
231   // check that the last time a bottle has been seen doesn't exceed a limit
232   if(millis() > timeBottleLastSeen)
233   {
234     timeNextBottleCheck = millis() + TIME_NEXT_BOTTLE_CHECK;
235
236     // a bottle hasn't been seed since TIME_BOTTLE_SEEN_TIMEOUT milliseconds, it was
            probably a false positive
237     if(booleanBottleSecondTry)
238     {
239       bottlePosition = 0;
240       return FAIL;
241     }
242
243     // back up a little bit just to be sure it was a false positive
244     booleanBottleSecondTry = 1;
245
246     robot.setSpeed(-CONST_SPEED_BOTTLE, -CONST_SPEED_BOTTLE);
247     delay(500);
248     robot.stop();
249     delay(1000);
```

```
250   }
251
252   // make the 0-255 bottle position value signed
253   int8_t deviation = 127 - bottlePosition;
254   static uint32_t timeBottleApproachingLastCheck = 0;
255
256   if(deviation > -CONST_DEVIATION_OK && deviation < CONST_DEVIATION_OK)
257   {
258     // if the bottle is close enough or it was lost while approaching from it (the
          bottle is right next to the robot), lower the cage
259     if(bottleDistance > CONST_BOTTLE_SIZE_LOWER_CAGE || millis() >
          timeBottleApproachingLastCheck)
260     {
261       // if the bottle magically appeared in front of the robot consider it a false
            positive
262       if(booleanFalseBottle)
263       {
264         bottlePosition = 0;
265         return FAIL;
266       }
267
268       // just to be sure move forwards for a little while
269       robot.setSpeed(CONST_SPEED_BOTTLE, CONST_SPEED_BOTTLE);
270       delay(500);
271       robot.stop();
272
273       return OK;
274     }
275
276     // approach the bottle only if the robot got an update on the bottle position
277     if(booleanBottleUpdate)
278     {
279       robot.setSpeed(CONST_SPEED_BOTTLE, CONST_SPEED_BOTTLE);
280       booleanBottleUpdate = 0;
281       timeBottleApproachingLastCheck = millis() + TIME_GRAB_LOST_BOTTLE;
282     }
283   }
284
285   // head towards the bottle at a lower speed
286   if(booleanBottleUpdate)
287   {
288     robot.turn(deviation, CONST_SPEED_BOTTLE);
289     delay(CONST_SPEED_SET_DELAY);
290     robot.stop();
291     booleanBottleUpdate = 0;
292     timeBottleApproachingLastCheck = millis() + TIME_GRAB_LOST_BOTTLE;
293   }
294
295   // the bottle was not perfectly detected in front of the robot the first time, so
        it's probably a true bottle
296   booleanFalseBottle = 0;
297
298   return REPEAT;
299 }
300
301 return_codes stateLowerCage()
302 {
303   // make sure the wheels are stopped
304   robot.stop();
305   robot.setCagePosition(CAGE_DOWN);
306   robot.playSound(ONEUP);
307   return OK;
```

```
308 }
309
310 /*return_codes stateFindWall()
311 {
312   uint16_t irLeft, irCenterLeft, irCenterRight, irRight;
313   robot.readIrSensors(&irLeft, &irCenterLeft, &irCenterRight, &irRight);
314
315   // get the heading angle between -180 and 180 degrees
316   int16_t deviation = robot.getHeading() - CONST_HEADING_HOME;
317
318   // must check of it will enter an infinite loop
319   static uint32_t timeAntiRecheck = 0;
320   static uint32_t timeTurnTime;
321
322   if(
323     millis() > timeAntiRecheck &&
324     (irLeft > CONST_IR_OBSTACLE_SIDE_CAGE || irRight > CONST_IR_OBSTACLE_SIDE_CAGE)
325   )
326   {
327     // if the robot is too close to a wall it's impossible for it to be a wall
328     if(irLeft > CONST_IR_TOO_CLOSE || irRight > CONST_IR_TOO_CLOSE)
329     {
330       timeAntiRecheck = millis() + TIME_WALL_RECHECK;
331       return REPEAT;
332     }
333
334     // see if this is an obstacle or a wall
335     // the obstacle was on the left, turn left and check if the robot can see it
          with its right side sensor, if at this point the left is still detecting the
           obstacle it's a wall
336     if(irLeft > irRight)
337     {
338       // the "obstacle" was on the left, turn towards the left to check for a wall
339       sideWall = LEFT;
340       timeTurnTime = millis();
341       robot.setSpeed(-CONST_SPEED_OBSTACLE, CONST_SPEED_OBSTACLE);
342
343       // wait until the right sensor sees the obstacle
344       while(analogRead(SENSOR_IR_RIGHT) < CONST_IR_OBSTACLE_SIDE_CAGE);
345
346       // stop before checking the other IR sensor
347       timeTurnTime = millis() - timeTurnTime;
348       robot.setSpeed(0, 0);
349
350       // check if the left sensor still sees the obstacle, if yes it's a wall
351       if(analogRead(SENSOR_IR_LEFT) > CONST_IR_OBSTACLE_FAR)
352       {
353         return OK;
354       }
355     }
356     // the obstacle was on the right, turn right and check if the robot can see it
          with its left side sensor, if at this point the right is still detecting the
           obstacle it's a wall
357     else
358     {
359       // the "obstacle" was on the right, turn towards the right to check for a wall
360       sideWall = RIGHT;
361       timeTurnTime = millis();
362       robot.setSpeed(CONST_SPEED_OBSTACLE, -CONST_SPEED_OBSTACLE);
363
364       // wait until the right sensor sees the obstacle
365       while(analogRead(SENSOR_IR_LEFT) < CONST_IR_OBSTACLE_SIDE_CAGE);
```

```
366
367       // stop before checking the other IR sensor
368       timeTurnTime = millis() - timeTurnTime;
369       robot.setSpeed(0, 0);
370
371       // check if the left sensor still sees the obstacle, if yes it's a wall
372       if(analogRead(SENSOR_IR_RIGHT) > CONST_IR_OBSTACLE_FAR)
373       {
374         return OK;
375       }
376     }
377
378     // false positive, turn back
379     if(sideWall == LEFT)
380     {
381       robot.setSpeed(CONST_SPEED_OBSTACLE, -CONST_SPEED_OBSTACLE);
382     }
383     else
384     {
385       robot.setSpeed(-CONST_SPEED_OBSTACLE, CONST_SPEED_OBSTACLE);
386     }
387     delay(timeTurnTime);
388
389     timeAntiRecheck = millis() + TIME_WALL_RECHECK;
390   }
391
392   // head towards the direction the compass indicates
393   robot.headTo(deviation);
394
395   return REPEAT;
396 }*/
397
398 return_codes stateGoHome()
399 {
400   return robot.goHome();
401
402   /*uint16_t irLeft, irCenterLeft, irCenterRight, irRight;
403   robot.readIrSensors(&irLeft, &irCenterLeft, &irCenterRight, &irRight);
404
405   int16_t deviation = robot.getHeading() - CONST_HEADING_HOME;
406
407   // at this point we know the robot is in front of the wall
408
409   // turn towards the home heading within a margin
410   Serial.println(deviation);
411
412   if(deviation > CONST_DEVIATION_OK)
413   {
414     robot.setSpeed(-CONST_SPEED_BOTTLE, CONST_SPEED_BOTTLE);
415   }
416   else if(deviation < -CONST_DEVIATION_OK)
417   {
418     robot.setSpeed(CONST_SPEED_BOTTLE, -CONST_SPEED_BOTTLE);
419   }
420   else if(irLeft > irRight)
421   {
422     // wall is on the left, turn left a little bit for the next state
423     robot.setSpeed(CONST_SPEED_MAX, 0);
424     delay(TIME_TURN_FOLLOW_WALL);
425     robot.setSpeed(0, 0);
426     sideWall = LEFT;
427     return OK;
```

```
428   }
429   else if(irRight > irLeft)
430   {
431     // wall is on the right, turn left a little bit for the next state
432     robot.setSpeed(0, CONST_SPEED_MAX);
433     delay(TIME_TURN_FOLLOW_WALL);
434     robot.setSpeed(0, 0);
435     sideWall = RIGHT;
436     return OK;
437   }
438   else
439   {
440     robot.headTo(deviation);
441   }
442
443   return REPEAT;*/
444 }
445
446 return_codes stateFollowWall()
447 {
448   uint16_t irLeft, irCenterLeft, irCenterRight, irRight;
449   robot.readIrSensors(&irLeft, &irCenterLeft, &irCenterRight, &irRight);
450
451   // follow the left wall
452   if(sideWall == LEFT)
453   {
454     // a wall was detected on the right, it can only be the goal
455     if(irRight > CONST_IR_OBSTACLE_SIDE_CAGE)
456     {
457       robot.stop();
458       return OK;
459     }
460
461     // go straight with an offset to the left
462     //robot.headTo(CONST_SPEED_MAX - 1);
463     robot.followWall(&irLeft, LEFT);
464   }
465   else
466   {
467     // a wall was detected on the left, it can only be the goal
468     if(irLeft > CONST_IR_OBSTACLE_SIDE_CAGE)
469     {
470       robot.stop();
471       return OK;
472     }
473
474     // go straight with an offset to the right
475     //robot.headTo(-(CONST_SPEED_MAX - 1));
476     robot.followWall(&irRight, RIGHT);
477   }
478
479   return REPEAT;
480 }
481
482 return_codes stateRaiseCage()
483 {
484   robot.setCagePosition(CAGE_UP);
485
486   // announce the glorious point it just probably got
487   //robot.playSound(FLAGPOLE);
488
489   if(robot.flags.wall == WALL_RIGHT)
```

```
490   {
491     robot.setSpeed(-CONST_SPEED_MAX, -(CONST_SPEED_MAX-10));
492   }
493   else
494   {
495     robot.setSpeed(-(CONST_SPEED_MAX-10), -CONST_SPEED_MAX);
496   }
497   delay(1000);
498
499   while(robot.getHeading() < 160 && robot.getHeading() > -160)
500   {
501     if(robot.flags.wall == WALL_LEFT)
502     {
503       robot.setSpeed(CONST_SPEED_MAX, -CONST_SPEED_MAX);
504     }
505     else
506     {
507       robot.setSpeed(-CONST_SPEED_MAX, CONST_SPEED_MAX);
508     }
509   }
510
511   robot.stop();
512
513   // reset the robot state before restarting
514   reset();
515
516   return OK;
517 }
518
519 void reset()
520 {
521   robot.flags.wall = NO_WALL;
522   bottlePosition = 0;
523   timeNextBottleCheck = millis() + TIME_NEXT_BOTTLE_CHECK;
524   booleanBottleSecondTry = 0;
525 }
526
527 // ############################################################ USER INPUT METHODS
528
529 void processInput(comm_methods method)
530 {
531   inputMethod = method;
532   digitalWrite(LED, HIGH);
533   switch(input())
534   {
535   case '0':
536     // force robot state to searching
537     output("Toggle robot\n");
538     bottlePosition = 0;
539     robot.flags.running = !robot.flags.running;
540
541     if(!robot.flags.running)
542     {
543       robot.stop();
544     }
545     robot.playSound(POWERUP);
546     break;
547   case '1':
548     output("Set state: searching\n");
549     currentState = STATE_SEARCHING;
550     break;
551   case '2':
```

```
552       output("Set state: fetching bottle\n");
553       currentState = STATE_FETCHING_BOTTLE;
554       break;
555     case '3':
556       output("Set state: go home\n");
557       currentState = STATE_LOWER_CAGE;
558       break;
559     case '4':
560       output("Set state: follow wall\n");
561       currentState = STATE_GO_HOME;
562       break;
563     case '5':
564       output("Set state: raise cage\n");
565       currentState = STATE_RAISE_CAGE;
566       break;
567     case 'B':
568       // a bottle was seen
569       bottlePosition = input();
570       bottleDistance = input();
571       timeBottleLastSeen = millis() + TIME_BOTTLE_SEEN_TIMEOUT;
572       booleanBottleUpdate = 1;
573       break;
574     case 'S':
575       robot.setSpeed(input(), input());
576       break;
577     case 't' :
578       play((input() << 8) | input(), 500);
579       break;
580     case '.' :
581       output("Lower cage\n");
582       robot.setCagePosition(CAGE_DOWN);
583       break;
584     case ',' :
585       output("Raise cage\n");
586       robot.setCagePosition(CAGE_UP);
587       break;
588     case 'm' :
589       output("Turn lights on\n");
590       robot.setLights(true);
591       break;
592     case 'n' :
593       output("Turn lights off\n");
594       robot.setLights(false);
595       break;
596     case 'w':
597       output("Go forwards\n");
598       robot.setSpeed(CONST_SPEED_MAX, CONST_SPEED_MAX);
599       break;
600     case 'a':
601       output("Go left\n");
602       robot.setSpeed(-CONST_SPEED_MAX, CONST_SPEED_MAX);
603       break;
604     case 's':
605       output("Go backwards\n");
606       robot.setSpeed(-CONST_SPEED_MAX, -CONST_SPEED_MAX);
607       break;
608     case 'd':
609       output("Go right\n");
610       robot.setSpeed(CONST_SPEED_MAX, -CONST_SPEED_MAX);
611       break;
612     case 'q':
613       output("Stop\n");
```

```
614     robot.stop();
615     break;
616   default:
617     output("Command not recognised\n");
618   }
619   digitalWrite(LED, LOW);
620 }
621 
622 char input()
623 {
624   if(inputMethod == USB)
625   {
626     return Serial.read();
627   }
628   else if(inputMethod == BLUETOOTH)
629   {
630     return Bluetooth.read();
631   }
632 }
633 
634 char output(const char* data)
635 {
636   if(inputMethod == USB)
637   {
638     Serial.print(data);
639   }
640   else if(inputMethod == BLUETOOTH)
641   {
642     Bluetooth.print(data);
643   }
644 }
```

**Listing 4:** Arduino main program

```
1  /*
       ********************************************************************************
2   *
3   * Title:       STI competition Arduino code for the Tokamak robot
4   * Date:        2014-06-06
5   *
6   ********************************************************************************
        */
7  #include <Servo.h>
8  #include <Wire.h>
9  #include <prismino.h>
10 #include "robot.h"
11 #include "pitch.h"
12 #include "sound.h"
13 
14 Tokamak::Tokamak()
15 {
16   // initialise default values
17   this->flags.enableFrontLeds = false;
18   this->flags.running = false;
19   this->flags.cagePosition = CAGE_UP;
20 }
21 
22 void Tokamak::setCagePosition(cage_positions position)
23 {
24   this->servoRight.attach(S1);
25   this->servoLeft.attach(S2);
26 
```

```
27    uint8_t r, l;
28
29    if(position == CAGE_UP)
30    {
31      // both sevomotors have 160 steps between up and down positions, so this is
          allowed
32      for(l = SERVO_LEFT_DOWN, r = SERVO_RIGHT_DOWN; r < SERVO_RIGHT_UP; r++, l--)
33      {
34        this->servoLeft.write(l);
35        this->servoRight.write(r);
36
37        delay(CONST_MAX_SERVO_SPEED);
38      }
39    }
40    else if(position == CAGE_DOWN)
41    {
42      for(l = SERVO_LEFT_UP, r = SERVO_RIGHT_UP; r > SERVO_RIGHT_DOWN; r--, l++)
43      {
44        this->servoLeft.write(l);
45        this->servoRight.write(r);
46
47        delay(CONST_MAX_SERVO_SPEED);
48      }
49
50    }
51
52    // always detach the motors so that they are free-running and do not consume power
        in resting positions
53    this->servoLeft.detach();
54    this->servoRight.detach();
55
56    this->flags.cagePosition = position;
57  }
58
59  void Tokamak::setSpeed(int8_t speedLeft, int8_t speedRight)
60  {
61    // controller has been disabled
62    #ifndef ENABLE_CONTROLLER
63    return;
64    #endif
65
66    Wire.beginTransmission(I2C_MOTOR_CONTROLLER_ADDRESS);
67    Wire.write("s");
68
69    // make sure not to go over the maximum speed limit (100 or -100)
70    if(speedLeft > CONST_SPEED_MAX)
71    {
72      Wire.write(CONST_SPEED_MAX);
73    }
74    else if(speedLeft < -CONST_SPEED_MAX)
75    {
76      Wire.write(-CONST_SPEED_MAX);
77    }
78    else
79    {
80      Wire.write(speedLeft);
81    }
82
83    if(speedRight > CONST_SPEED_MAX)
84    {
85      Wire.write(CONST_SPEED_MAX);
86    }
```

```
87    else if(speedRight < -CONST_SPEED_MAX)
88    {
89      Wire.write(-CONST_SPEED_MAX);
90    }
91    else
92    {
93      Wire.write(speedRight);
94    }
95
96    Wire.endTransmission();
97
98    // a small delay is needed so that the speed could actually be applied to the
          wheels
99    delay(CONST_SPEED_SET_DELAY);
100 }
101
102 void Tokamak::stop()
103 {
104   this->setSpeed(0, 0);
105 }
106
107 void Tokamak::checkBattery()
108 {
109   Wire.beginTransmission(I2C_MOTOR_CONTROLLER_ADDRESS);
110
111   // casting needed for some reason
112   uint8_t available = Wire.requestFrom((uint8_t)I2C_MOTOR_CONTROLLER_ADDRESS, (
          uint8_t)6);
113
114   // read 6 bytes, 2 by 2, high byte first: voltage, left motor current, right motor
          current
115   if(available == 6)
116   {
117     this->batteryVoltage = (Wire.read() << 8) | Wire.read();
118     this->currentLeft = (Wire.read() << 8) | Wire.read();
119     this->currentRight = (Wire.read() << 8) | Wire.read();
120   }
121   else
122   {
123     // inform i2c error
124     play(TONE_I2C_ERROR, 500);
125   }
126
127   Wire.endTransmission();
128
129   if(this->batteryVoltage < CONST_BATTERY_LOW)
130   {
131     play(TONE_BATTERY, 500);
132   }
133 }
134
135 void Tokamak::setLights(boolean state)
136 {
137   digitalWrite(PIN_LIGHTS, state);
138 }
139
140 int16_t Tokamak::getHeading()
141 {
142   #ifndef ENABLE_COMPASS
143   return 0;
144   #endif
145
```

```
146   int16_t x = 0, y = 0, z = 0;
147
148   Wire.beginTransmission(I2C_COMPASS_ADDRESS);
149   // select register 3, X MSB register
150   Wire.write(0x03);
151   Wire.endTransmission();
152
153   Wire.requestFrom(I2C_COMPASS_ADDRESS, 6);
154   if(6 <= Wire.available())
155   {
156     x = (Wire.read() << 8) | Wire.read();
157     z = (Wire.read() << 8) | Wire.read();
158     y = (Wire.read() << 8) | Wire.read();
159   }
160   else
161   {
162     // inform i2c error
163     play(TONE_I2C_ERROR, 500);
164     return 0;
165   }
166
167   int16_t angle = atan2(x, z) * 180 / M_PI;
168   return angle;
169 }
170
171 void Tokamak::turn(int16_t deviation, int8_t speed)
172 {
173   int8_t speedLeft = 0;
174   int8_t speedRight = 0;
175
176   // limit deviation to maximum allowed speed
177   if(deviation > speed)
178   {
179     deviation = speed;
180   }
181   else if(deviation < -speed)
182   {
183     deviation = -speed;
184   }
185
186   if(deviation < -CONST_DEVIATION_OK)
187   {
188     speedLeft = speed;
189     speedRight = -speed;
190   }
191   else if(deviation > CONST_DEVIATION_OK)
192   {
193     speedLeft = -speed;
194     speedRight = speed;
195   }
196   else
197   {
198     speedLeft = 0;
199     speedRight = 0;
200   }
201
202   this->setSpeed(speedLeft, speedRight);
203 }
204
205 void Tokamak::headTo(int16_t deviation, int8_t speed)
206 {
207   navigation_avoidance(deviation);
```

```
208   return;
209
210   /*int8_t speedLeft = 0;
211   int8_t speedRight = 0;
212
213   uint16_t irLeft, irCenterLeft, irCenterRight, irRight;
214   this->readIrSensors(&irLeft, &irCenterLeft, &irCenterRight, &irRight);
215
216   // these values are incremented over time and reset after every TIME_ANTI_LOOP
          seconds
217   static uint8_t antiLoopCountLeft = 0, antiLoopCountRight = 0;
218   static uint32_t timeAntiLoop = 0;
219
220   // every TIME_ANTI_LOOP seconds reset the anti-loop timer and wheel counters
221   if(millis() > timeAntiLoop)
222   {
223     // every TIME_ANTI_LOOP seconds reset the anti-loop timer and wheel counters
224     antiLoopCountLeft = 0;
225     antiLoopCountRight = 0;
226     timeAntiLoop = millis() + TIME_ANTI_LOOP;
227   }
228
229   // limit deviation to maximum allowed speed
230   if(deviation > speed)
231   {
232     deviation = speed;
233   }
234   else if(deviation < -speed)
235   {
236     deviation = -speed;
237   }
238
239   // check if the robot has entered an infinite loop
240   if(antiLoopCountLeft > CONST_LOOP_TURN_TIMES && antiLoopCountRight >
          CONST_LOOP_TURN_TIMES)
241   {
242     // if the cage is deployed reverse a bit
243     if(this->flags.cagePosition == CAGE_UP)
244     {
245       this->setSpeed(-speed, -speed);
246       delay(500);
247     }
248
249     // set speed immediately
250     if(deviation > 0)
251     {
252       this->setSpeed(speed, -speed);
253     }
254     else
255     {
256       this->setSpeed(-speed, speed);
257     }
258
259     delay(TIME_ANTI_LOOP_TIMEOUT_TURN);
260
261     // reset the counters
262     antiLoopCountLeft = 0;
263     antiLoopCountRight = 0;
264   }
265   // avoid obstacles using the IR sensors
266   else if(irCenterRight > CONST_IR_OBSTACLE_CENTER_CAGE)
267   {
```

```
268     // turn on itself
269     speedLeft = speed;
270     speedRight = -speed;
271     antiLoopCountLeft++;
272   }
273   else if(irCenterLeft > CONST_IR_OBSTACLE_CENTER_CAGE)
274   {
275     speedLeft = -speed;
276     speedRight = speed;
277     antiLoopCountRight++;
278   }
279   else if(irRight > CONST_IR_OBSTACLE_SIDE_CAGE)
280   {
281     // block one wheel, reverse the other
282     speedLeft = -speed;
283     speedRight = 0;
284     antiLoopCountLeft++;
285   }
286   else if(irLeft > CONST_IR_OBSTACLE_SIDE_CAGE)
287   {
288     speedLeft = 0;
289     speedRight = -speed;
290     antiLoopCountRight++;
291   }
292   else if(deviation < -CONST_DEVIATION_OK)
293   {
294     speedLeft = speed;
295     speedRight = speed + deviation;
296   }
297   else if(deviation > CONST_DEVIATION_OK)
298   {
299     speedLeft = speed - deviation;
300     speedRight = speed;
301   }
302   else
303   {
304     // simply go forwards
305     speedLeft = speed;
306     speedRight = speed;
307   }
308
309   this->setSpeed(speedLeft, speedRight);*/
310 }
311
312 void Tokamak::followWall(uint16_t *sensorValue, sides side, int8_t speed)
313 {
314   if(side == RIGHT && *sensorValue < CONST_IR_WALL_FOLLOW_MIN)
315   {
316     this->setSpeed(speed, speed >> 2);
317   }
318   else if(side == RIGHT && *sensorValue > CONST_IR_WALL_FOLLOW_MAX)
319   {
320     this->setSpeed(speed >> 2, speed);
321   }
322   else if(side == LEFT && *sensorValue < CONST_IR_WALL_FOLLOW_MIN)
323   {
324     this->setSpeed(speed >> 2, speed);
325   }
326   else if(side == LEFT && *sensorValue > CONST_IR_WALL_FOLLOW_MAX)
327   {
328     this->setSpeed(speed, speed >> 2);
329   }
```

```
330 }
331
332 void Tokamak::readIrSensors(uint16_t *irLeft, uint16_t *irCenterLeft, uint16_t *
        irCenterRight, uint16_t *irRight)
333 {
334   *irLeft = analogRead(SENSOR_IR_LEFT);
335   *irCenterLeft = analogRead(SENSOR_IR_CENTER_LEFT);
336   *irCenterRight = analogRead(SENSOR_IR_CENTER_RIGHT);
337   *irRight = analogRead(SENSOR_IR_RIGHT);
338 }
339
340 // a sound is an array of notes
341 note notesCoin[] = {
342   {B5, 100},
343   {E6, 200}
344 };
345
346 sound soundCoin = {sizeof(notesCoin) / sizeof(note), notesCoin};
347
348 note notesPowerUp[] = {
349   {G3, 50},
350   {B4, 50},
351   {D4, 50},
352   {G4, 50},
353   {B5, 50},
354   {A4b, 50},
355   {C4, 50},
356   {E4b, 50},
357   {A5b, 50},
358   {C5, 50},
359   {B4b, 50},
360   {D4, 50},
361   {F4, 50},
362   {B5b, 50},
363   {D5, 50}
364 };
365
366 sound soundPowerUp = {sizeof(notesPowerUp) / sizeof(note), notesPowerUp};
367
368 note notesOneUp[] = {
369   {E4, 100},
370   {G4, 100},
371   {E5, 100},
372   {C5, 100},
373   {D5, 100},
374   {G5, 100}
375 };
376
377 sound soundOneUp = {sizeof(notesOneUp) / sizeof(note), notesOneUp};
378
379 note notesFlagpoleFanfare[] = {
380   {G2, 100},
381   {C3, 100},
382   {E3, 100},
383   {G3, 100},
384   {C4, 100},
385   {E4, 100},
386   {G4, 300},
387   {E4, 300},
388   {A2b, 100},
389   {C3, 100},
390   {E3b, 100},
```

```
391   {A3b, 100},
392   {C4, 100},
393   {E4b, 100},
394   {A4b, 300},
395   {E4b, 300},
396   {B2b, 100},
397   {D3, 100},
398   {F3, 100},
399   {B3b, 100},
400   {D4, 100},
401   {F4, 100},
402   {B4b, 300},
403   {B4b, 100},
404   {B4b, 100},
405   {B4b, 100},
406   {C5, 600},
407 };
408
409 sound soundFlagpoleFanfare = {sizeof(notesFlagpoleFanfare) / sizeof(note),
        notesFlagpoleFanfare};
410
411 // function that actually plays the sounds
412 void Tokamak::playSound(sounds theSound)
413 {
414   #ifndef ENABLE_SOUND
415   return;
416   #endif
417
418   sound soundPtr;
419
420   // pointer to the sound object
421   switch(theSound)
422   {
423     case COIN:
424       soundPtr = soundCoin;
425       break;
426     case POWERUP:
427       soundPtr = soundPowerUp;
428       break;
429     case ONEUP:
430       soundPtr = soundOneUp;
431       break;
432     case FLAGPOLE:
433       soundPtr = soundFlagpoleFanfare;
434       break;
435   }
436
437   uint16_t frequency, duration;
438   uint8_t length = soundPtr.length;
439   for(uint8_t i = 0; i < length; i++)
440   {
441     // get data from program memory
442     frequency = soundPtr.notes[i].pitch;
443     duration = soundPtr.notes[i].duration;
444
445     // if the next note is the same then make a short pause
446     uint8_t pause = 0;
447     if(i < length - 1 && frequency == soundPtr.notes[i + 1].pitch)
448     {
449       // 5 millisecond pause
450       pause = 5;
451     }
```

```
452
453     // play the right pitch for the determined duration
454     play(frequency, duration - pause);
455
456     // play is not a blocking function so one has to manually set a delay
457     delay(duration);
458   }
459 }
460
461 void Tokamak::resetReturnBase(void)
462 {
463   this->flags.wall = NO_WALL;
464 }
465
466 return_codes Tokamak::goHome(int16_t direction_home)
467 {
468   //tell RASP to stop looking for bottles -> accelerates image processing
469   //tell RASP to start looking for beacon
470
471   //getHeading();
472   //gotoHeading(direction_home);
473   uint16_t irFarLeft = analogRead(SENSOR_IR_LEFT);
474   uint16_t irFarRight = analogRead(SENSOR_IR_RIGHT);
475
476   if(this->flags.wall == NO_WALL)
477   {
478     this->flags.wall = detectWall();
479     gotoHeading(direction_home);
480     if(this->flags.wall != NO_WALL)
481     {
482       //Serial.println("wall detected");
483       this->setSpeed(0,0);
484     }
485   }
486   else
487   {
488     if(this->flags.wall == WALL_LEFT)
489     {
490 //      Serial.println("WALL LEFT");
491       if(irFarRight > CONST_IR_OBSTACLE_SIDE_CAGE)
492       {
493         this->setSpeed(0,0);
494         play(440, 500);
495         return OK;
496       }
497       else navigation_avoidance(-6);
498     }
499     else if(this->flags.wall == WALL_RIGHT)
500     {
501 //      Serial.println("WALL RIGHT");
502       if(irFarLeft > CONST_IR_OBSTACLE_SIDE_CAGE)
503       {
504         this->setSpeed(0,0);
505         play(440, 500);
506         return OK;
507       }
508       else navigation_avoidance(6);
509     }
510   }
511   return REPEAT;
512   //if(beacon_detected) //stop, open cage, wallSide = 0;
513 }
```

```
514
515
516 uint8_t Tokamak::detectWall(void)
517 {
518   uint16_t irFarLeft = analogRead(SENSOR_IR_LEFT);
519   uint16_t irFarRight = analogRead(SENSOR_IR_RIGHT);
520   static uint32_t timeAntiRecheck;
521
522   if((irFarLeft < 100 && irFarRight < CONST_IR_OBSTACLE_SIDE_CAGE) || (irFarLeft <
        CONST_IR_OBSTACLE_SIDE_CAGE && irFarRight < 100) )
523   {
524     return NO_WALL; // no wall detected
525     timeAntiRecheck = millis() + TIME_WALL_RECHECK;
526   }
527   else
528   {
529     if(millis()<timeAntiRecheck) return NO_WALL;
530     if (irFarLeft > irFarRight)
531     {
532       if(irFarLeft < 200)
533       {
534           this->setSpeed(-100,+100);
535           delay(TIME_WALL_CHECK_TURN);
536           this->stop();
537           irFarLeft = analogRead(SENSOR_IR_LEFT);
538           irFarRight = analogRead(SENSOR_IR_RIGHT);
539           delay(50); //wait for IR
540           if (irFarLeft > 100 && irFarRight > 100)
541           {
542             this->setSpeed(+100,-100);
543             delay(TIME_WALL_CHECK_TURN+500);
544             //Serial.println("WALL LEFT");
545             return WALL_LEFT; // wall detected on the left
546           }
547           else
548           {
549             this->setSpeed(+100,-100);
550             delay(TIME_WALL_CHECK_TURN);
551             timeAntiRecheck = millis() + TIME_WALL_RECHECK;
552             return NO_WALL;
553           }
554         }
555         else
556         {
557           //incase the robot wants to turn on itself, but an obstacle is too near
                and would destroy the deployed cage
558           this->setSpeed(-100,-100);
559           delay(1000);
560         }
561       }
562     else
563     {
564       if(irFarRight < 200)
565       {
566         this->setSpeed(+100,-100);
567         delay(TIME_WALL_CHECK_TURN);
568         this->stop();
569         irFarLeft = analogRead(SENSOR_IR_LEFT);
570         irFarRight = analogRead(SENSOR_IR_RIGHT);
571         delay(50); //wait for IR
572
573         if (irFarLeft > 100 && irFarRight > 100)
```

```
574         {
575           this->setSpeed(-100,+100);
576           delay(TIME_WALL_CHECK_TURN+500);
577           return WALL_RIGHT; // wall detected on the left
578         }
579         else
580         {
581           this->setSpeed(-100,+100);
582           delay(TIME_WALL_CHECK_TURN);
583           timeAntiRecheck = millis() + TIME_WALL_RECHECK;
584           return NO_WALL;
585         }
586       }
587       else
588       {
589         //incase the robot wants to turn on itself, but an obstacle is too near and
              would destroy the deployed cage
590         this->setSpeed(-100,-100);
591         delay(1000);
592       }
593     }
594   }
595 }
596
597 void Tokamak::gotoHeading(int16_t direction_home)
598 {
599
600   int16_t currentHeading = this->getHeading();
601   int16_t error;
602
603   error = currentHeading - direction_home;
604
605   // in order to avoid the -180 -> 180 jump
606   if(error > 180){
607     error = error - 360 ;
608   }
609   else if(error < -180){
610     error = error + 360 ;
611   }
612
613   //reduce error so that the navigation function can use it
614   if(error>100) error = 100;
615   else if(error<-100) error = -100;
616
617   this->navigation_avoidance(-error);
618 }
619
620 void Tokamak::navigation_avoidance(int16_t error)
621 {
622   uint16_t irLeft, irRight, irFarLeft, irFarRight;
623   int8_t speedL,speedR;
624   static int8_t countL,countR;
625
626   irFarLeft = analogRead(SENSOR_IR_LEFT);
627   irFarRight = analogRead(SENSOR_IR_RIGHT);
628   irLeft = analogRead(SENSOR_IR_CENTER_LEFT);
629   irRight = analogRead(SENSOR_IR_CENTER_RIGHT);
630
631   static uint32_t timeAntiLoop = 0;
632
633   if(millis() > timeAntiLoop)
634   {
```

```
635     // every 3 seconds reset the anti-loop timer and wheel counters
636     countL = 0;
637     countR = 0;
638     timeAntiLoop = millis() + TIME_ANTI_LOOP;
639   }
640   if(countL > 3 && countR > 3) // this condition is to not get stuck
641   {
642
643     if (this->flags.cagePosition == CAGE_DOWN) //reverse
644     {
645       speedL = -CONST_SPEED_OBSTACLE;
646       speedR = -CONST_SPEED_OBSTACLE;
647       this->setSpeed(speedL, speedR);
648       delay(500);
649     }
650     if(error>0)
651     {
652       speedL = CONST_SPEED_OBSTACLE;
653       speedR = -CONST_SPEED_OBSTACLE;
654     }
655     else
656      {
657       speedL = -CONST_SPEED_OBSTACLE;
658       speedR = CONST_SPEED_OBSTACLE;
659     }
660     // set speed immediately
661     this->setSpeed(speedL, speedR);
662     // turn for 1 second
663     delay(1000);
664     // reset the counters
665     countL = 0;
666     countR = 0;
667   }
668   // avoid obstacles using the IR sensors
669   else if(irRight > CONST_IR_OBSTACLE_CENTER_CAGE)
670   {
671     // turn on itself
672     speedL = CONST_SPEED_OBSTACLE;
673     speedR = -CONST_SPEED_OBSTACLE;
674     countL++;
675   }
676   else if(irLeft > CONST_IR_OBSTACLE_CENTER_CAGE)
677   {
678     speedL = -CONST_SPEED_OBSTACLE;
679     speedR = CONST_SPEED_OBSTACLE;
680     countR++;
681   }
682   else if(irFarRight > CONST_IR_OBSTACLE_SIDE_CAGE)
683   {
684     // block one wheel, reverse the other
685     speedL = -CONST_SPEED_OBSTACLE;
686     speedR = 0;
687     countL++;
688   }
689   else if(irFarLeft > CONST_IR_OBSTACLE_SIDE_CAGE)
690   {
691     speedL = 0;
692     speedR = -CONST_SPEED_OBSTACLE;
693     countR++;
694   }
695   else
696   {
```

```
697     if( abs(error)<5 )
698     {
699       // under an error of 5 units go straight
700       speedL = CONST_SPEED_MAX;
701       speedR = CONST_SPEED_MAX;
702     }
703     else
704     {
705       if(error>0)
706       {
707         if(error == 100 && irFarRight > 100)
708         {
709             speedL = CONST_SPEED_OBSTACLE;
710             speedR = -CONST_SPEED_OBSTACLE;
711         }
712         else
713         {
714           speedL = CONST_SPEED_OBSTACLE;
715           speedR =(100 - abs(error));//* CONST_SPEED_OBSTACLE / 100 );
716         }
717         //Serial.write("turn right\t");
718       }
719       else
720       {
721          if(abs(error) == 100 && irFarLeft > 100)
722         {
723             speedL = -CONST_SPEED_OBSTACLE;
724             speedR = CONST_SPEED_OBSTACLE;
725         }
726         else
727         {
728           speedL = (100-abs(error)); //* CONST_SPEED_OBSTACLE / 100 );
729           speedR = CONST_SPEED_OBSTACLE;
730         }
731         //Serial.write("turn left\t");
732       }
733     }
734   }
735
736   this->setSpeed(speedL, speedR);
737 }
```

**Listing 5:** Robot class

```
1  /*
       ********************************************************************************
2   *
3   * Title:      STI competition Arduino code for the Tokamak robot
4   * Date:       2014-06-06
5   *
6   ********************************************************************************
       */
7  #ifndef _PRISMINO_MASTER
8  #define _PRISMINO_MASTER
9
10 // comment this line when testing without the motor controller
11 #define ENABLE_CONTROLLER
12 // comment this line to disable all bluetooth functionalities
13 #define ENABLE_BLUETOOTH
14 // comment this line to turn off sounds
15 #define ENABLE_SOUND
16 // comment this line to disable compass
```

```
17 #define ENABLE_COMPASS
18
19 #define Bluetooth Serial1
20
21 #define I2C_MOTOR_CONTROLLER_ADDRESS 0x04
22 #define I2C_COMPASS_ADDRESS 0x1E
23
24 #define PIN_LIGHTS 9
25
26 #define SENSOR_IR_LEFT 0
27 #define SENSOR_IR_CENTER_LEFT 1
28 #define SENSOR_IR_CENTER_RIGHT 2
29 #define SENSOR_IR_RIGHT 3
30
31 #define SERVO_LEFT_UP 10
32 // set up position to 5 when the mast is installed
33 //#define SERVO_LEFT_UP 50
34 #define SERVO_LEFT_DOWN 170
35 #define SERVO_RIGHT_UP SERVO_LEFT_DOWN
36 #define SERVO_RIGHT_DOWN SERVO_LEFT_UP
37
38 #define TONE_I2C_ERROR 3300
39 #define TONE_BEEP 440
40 #define TONE_BATTERY 1100
41
42 #define TIME_CHECK_BATTERY 1000
43 #define TIME_ANTI_LOOP 3000
44 #define TIME_ANTI_LOOP_TIMEOUT_TURN 1000
45 // if a bottle hasn't been detected during this time consider it a false positive
46 #define TIME_BOTTLE_SEEN_TIMEOUT 2000
47 // after a false positive the robot won't check for a bottle for this amount of time
48 #define TIME_NEXT_BOTTLE_CHECK 1000
49 // when an obstacle has been seen instead of a wall in the STATE_FIND_WALL state do
       not check for a wall for this amount of time
50 #define TIME_WALL_RECHECK 4000
51 #define TIME_TURN_FOLLOW_WALL 1500
52 #define TIME_TURN_AROUND 1000
53
54 #define TIME_WALL_CHECK_TURN 700
55
56 #define TIME_GRAB_LOST_BOTTLE 500
57
58 // in percent the maximum allowed setting speed
59 #define CONST_SPEED_MAX 100
60 // in percent the maximum speed while avoiding obstacles
61 #define CONST_SPEED_OBSTACLE 100
62 // in percent the speed of approach when a bottle was seen, 60 is not enough with
       low-power 172:1 and 7.2V battery
63 #define CONST_SPEED_BOTTLE 80
64 // after every setSpeed() wait this long before continuing
65 #define CONST_SPEED_SET_DELAY 50
66
67 // this is the threshold when to lower the cage, the constant indicated the minimum
       box size the bottle must be detected in
68 #define CONST_BOTTLE_SIZE_LOWER_CAGE 90
69 // acceptable error when heading towards a direction (goes straight forward)
70 #define CONST_DEVIATION_OK 15
71 #define CONST_DEVIATION_OK_STRICT 5
72 // number of times the robot must change direction without going forwards for it to
       be considered stuck
73 #define CONST_LOOP_TURN_TIMES 3
74
```

```
75 // in milliseconds, the pause time between 1 degree change
76 #define CONST_MAX_SERVO_SPEED 10
77 // calibrated value, about 7.1V
78 #define CONST_BATTERY_LOW 470
79
80 // IR sensor value under which an obstacle is detected
81 #define CONST_IR_OBSTACLE_SIDE 300
82 #define CONST_IR_OBSTACLE_CENTER 310
83 // IR sensor values to use when the cage is deployed
84 #define CONST_IR_OBSTACLE_SIDE_CAGE 160
85 #define CONST_IR_OBSTACLE_CENTER_CAGE 140
86 // value at which there an obstacle far away
87 #define CONST_IR_NO_OBSTACLE 100
88 #define CONST_IR_TOO_CLOSE 250
89
90 // when following a wall the sensor should remain between these values
91 #define CONST_IR_WALL_FOLLOW_MIN 160
92 #define CONST_IR_WALL_FOLLOW_MAX 200
93
94 #define CONST_IR_OBSTACLE_FAR 100
95
96 // direction towards which to go when going home
97 #define CONST_HEADING_HOME 50
98
99 enum comm_methods
100 {
101   USB,
102   BLUETOOTH
103 };
104
105 enum cage_positions
106 {
107   CAGE_UP,
108   CAGE_DOWN
109 };
110
111 enum state_codes
112 {
113   STATE_SEARCHING,
114   STATE_FETCHING_BOTTLE,
115   STATE_LOWER_CAGE,
116   STATE_GO_HOME,
117   STATE_RAISE_CAGE
118 };
119
120 enum return_codes
121 {
122   OK,
123   FAIL,
124   REPEAT
125 };
126
127 enum sounds
128 {
129   COIN,
130   POWERUP,
131   ONEUP,
132   FLAGPOLE
133 };
134
135 enum sides
136 {
```

```
137   LEFT,
138   RIGHT
139 };
140
141 enum state_wall
142 {
143   NO_WALL,
144   WALL_LEFT,
145   WALL_RIGHT
146 };
147
148 //#define ENTRY_STATE STATE_SEARCHING
149 #define ENTRY_STATE STATE_SEARCHING
150
151 class Tokamak
152 {
153   public:
154   struct Flags
155   {
156     uint8_t enableFrontLeds :1;
157     uint8_t cagePosition :1;
158     uint8_t running: 1;
159     uint8_t wall: 2;
160   };
161
162   Servo servoLeft;
163   Servo servoRight;
164   uint16_t batteryVoltage;
165   uint16_t currentLeft;
166   uint16_t currentRight;
167
168   char readInput(comm_methods);
169   void sendOutput(comm_methods, const char*);
170
171   Flags flags;
172
173   Tokamak();
174   void setCagePosition(cage_positions);
175   void setSpeed(int8_t, int8_t);
176   void turn(int16_t, int8_t);
177   void stop(void);
178   void setLights(boolean);
179   void checkBattery(void);
180   int16_t getHeading();
181   // default speed is the maximum
182   void headTo(int16_t, int8_t = CONST_SPEED_MAX);
183   void readIrSensors(uint16_t*, uint16_t*, uint16_t*, uint16_t*);
184   void playSound(sounds);
185
186   void followWall(uint16_t*, sides, int8_t = CONST_SPEED_MAX);
187   void navigation_avoidance(int16_t);
188   void gotoHeading(int16_t direction_home = CONST_HEADING_HOME);
189   void resetReturnBase(void);
190   return_codes goHome(int16_t = CONST_HEADING_HOME);
191   uint8_t detectWall(void);
192 };
193
194 #endif
```

**Listing 6:** Robot header

```
1 #ifndef _sound_h
2 #define _sound_h
```

```
3
4  // define a note with a pitch and a duration
5  struct note
6  {
7    uint16_t pitch;
8    uint16_t duration;
9  };
10
11 // define a sound that has a length (number of notes) and a list of notes
12 struct sound
13 {
14   uint8_t length;
15   note *notes;
16 };
17
18 #endif
```

**Listing 7:** Sound header

```
1  // Frequencies for equal-tempered scale, A4 = 440Hz
2  // Reference: http://www.phy.mtu.edu/~suits/notefreqs.html
3
4  #ifndef _pitch_h
5  #define _pitch_h
6
7  // Scientific name
8  #define C0  16
9  #define C0s 17
10 #define D0b 17
11 #define D0  18
12 #define D0s 19
13 #define E0b 19
14 #define E0  21
15 #define F0  22
16 #define F0s 23
17 #define G0b 23
18 #define G0  25
19 #define G0s 26
20 #define A0b 26
21 #define A0  28
22 #define A0s 29
23 #define B0b 29
24 #define B0  31
25 #define C1  33
26 #define C1s 35
27 #define D1b 35
28 #define D1  37
29 #define D1s 39
30 #define E1b 39
31 #define E1  41
32 #define F1  44
33 #define F1s 46
34 #define G1b 46
35 #define G1  49
36 #define G1s 52
37 #define A1b 52
38 #define A1  55
39 #define A1s 58
40 #define B1b 58
41 #define B1  62
42 #define C2  65
43 #define C2s 69
44 #define D2b 69
```

```
45 #define D2  73
46 #define D2s 78
47 #define E2b 78
48 #define E2  82
49 #define F2  87
50 #define F2s 93
51 #define G2b 93
52 #define G2  98
53 #define G2s 104
54 #define A2b 104
55 #define A2  110
56 #define A2s 117
57 #define B2b 117
58 #define B2  123
59 #define C3  131
60 #define C3s 139
61 #define D3b 139
62 #define D3  147
63 #define D3s 156
64 #define E3b 156
65 #define E3  165
66 #define F3  175
67 #define F3s 185
68 #define G3b 185
69 #define G3  196
70 #define G3s 208
71 #define A3b 208
72 #define A3  220
73 #define A3s 233
74 #define B3b 233
75 #define B3  247
76 #define C4  262
77 #define C4s 277
78 #define D4b 277
79 #define D4  294
80 #define D4s 311
81 #define E4b 311
82 #define E4  330
83 #define F4  349
84 #define F4s 370
85 #define G4b 370
86 #define G4  392
87 #define G4s 415
88 #define A4b 415
89 #define A4  440
90 #define A4s 466
91 #define B4b 466
92 #define B4  494
93 #define C5  523
94 #define C5s 554
95 #define D5b 554
96 #define D5  587
97 #define D5s 622
98 #define E5b 622
99 #define E5  659
100 #define F5  698
101 #define F5s 740
102 #define G5b 740
103 #define G5  784
104 #define G5s 831
105 #define A5b 831
106 #define A5  880
```

```
107 #define A5s 932
108 #define B5b 932
109 #define B5  988
110 #define C6  1047
111 #define C6s 1109
112 #define D6b 1109
113 #define D6  1175
114 #define D6s 1245
115 #define E6b 1245
116 #define E6  1319
117 #define F6  1397
118 #define F6s 1480
119 #define G6b 1480
120 #define G6  1568
121 #define G6s 1661
122 #define A6b 1661
123 #define A6  1760
124 #define A6s 1865
125 #define B6b 1865
126 #define B6  1976
127 #define C7  2093
128 #define C7s 2217
129 #define D7b 2217
130 #define D7  2349
131 #define D7s 2489
132 #define E7b 2489
133 #define E7  2637
134 #define F7  2794
135 #define F7s 2960
136 #define G7b 2960
137 #define G7  3136
138 #define G7s 3322
139 #define A7b 3322
140 #define A7  3520
141 #define A7s 3729
142 #define B7b 3729
143 #define B7  3951
144 #define C8  4186
145 #define C8s 4435
146 #define D8b 4435
147 #define D8  4699
148 #define D8s 4978
149
150 // Piano key number
151 #define P01 28
152 #define P02 29
153 #define P03 31
154 #define P04 33
155 #define P05 35
156 #define P06 37
157 #define P07 39
158 #define P08 41
159 #define P09 44
160 #define P10 46
161 #define P11 49
162 #define P12 52
163 #define P13 55
164 #define P14 58
165 #define P15 62
166 #define P16 65
167 #define P17 69
168 #define P18 73
```

```
169 #define P19 78
170 #define P20 82
171 #define P21  87
172 #define P22 93
173 #define P23 98
174 #define P24 104
175 #define P25 110
176 #define P26 117
177 #define P27 123
178 #define P28 131
179 #define P29 139
180 #define P30 147
181 #define P31 156
182 #define P32 165
183 #define P33 175
184 #define P34 185
185 #define P35 196
186 #define P36 208
187 #define P37 220
188 #define P38 233
189 #define P39 247
190 #define P40 262
191 #define P41 277
192 #define P42 294
193 #define P43 311
194 #define P44 330
195 #define P45 349
196 #define P46 370
197 #define P47 392
198 #define P48 415
199 #define P49 440
200 #define P50 466
201 #define P51 494
202 #define P52 523
203 #define P53 554
204 #define P54 587
205 #define P55 622
206 #define P56 659
207 #define P57 698
208 #define P58 740
209 #define P59 784
210 #define P60 831
211 #define P61 880
212 #define P62 932
213 #define P63 988
214 #define P64 1047
215 #define P65 1109
216 #define P66 1175
217 #define P67 1245
218 #define P68 1319
219 #define P69 1397
220 #define P70 1480
221 #define P71 1568
222 #define P72 1661
223 #define P73 1760
224 #define P74 1865
225 #define P75 1976
226 #define P76 2093
227 #define P77 2217
228 #define P78 2349
229 #define P79 2489
230 #define P80 2637
```

```
231 #define P81 2794
232 #define P82 2960
233 #define P83 3136
234 #define P84 3322
235 #define P85 3520
236 #define P86 3729
237 #define P87 3951
238 #define P88 4186
239
240 #endif
```

**Listing 8:** Pitch header

## C.3 Motor controller source code

```
1  /*
       ********************************************************************************
2   *
3   * Title:       WildThumper Motor Controller sketch
4   * Date:        2014-05-27
5   *
6   ********************************************************************************
       */
7  #include <wildthumper.h>
8  #include <Wire.h>
9
10 #define I2C_ADDRESS 4
11
12 #define TIME_CHECK_BATTERY 500
13 #define CONST_BATTERY_LEVEL_GOOD 470
14
15 #define PIN_BACK_LIGHT 12
16 #define TIME_BACK_LIGHT_BLINK 50
17 #define TIME_BACK_LIGHT_PASUE 500
18
19 // must be an even number
20 #define CONST_BACK_LIGHT_BLINK_TIMES 6
21
22 uint32_t backLightNextBlink = 0;
23 uint32_t backLightBlinkTimes = CONST_BACK_LIGHT_BLINK_TIMES;
24
25 WildThumper controller;
26
27 uint8_t batteryGood = 1;
28 uint8_t buffer[6];
29 uint16_t input;
30 uint32_t timeCheckBattery = 0;
31
32 void setup()
33 {
34   pinMode(LED, OUTPUT);
35   digitalWrite(LED, HIGH);
36
37   pinMode(PIN_BACK_LIGHT, OUTPUT);
38
39   Wire.begin(I2C_ADDRESS);
40   // disable internal pull-ups
41   digitalWrite(SDA, LOW);
42   digitalWrite(SCL, LOW);
43
44   Wire.onReceive(receiveEvent);
```

```
45    Wire.onRequest(requestEvent);
46  }
47
48  void loop()
49  {
50    // check the battery every now and then
51    if(millis() > timeCheckBattery + TIME_CHECK_BATTERY)
52    {
53      // show the battery state on the on-board LED
54      if(controller.battery() < CONST_BATTERY_LEVEL_GOOD)
55      {
56        batteryGood = 0;
57        digitalWrite(LED, LOW);
58      }
59      timeCheckBattery = millis() + TIME_CHECK_BATTERY;
60    }
61
62    if(millis() > backLightNextBlink && batteryGood)
63    {
64      if(backLightBlinkTimes)
65      {
66        backLightBlinkTimes--;
67        digitalWrite(PIN_BACK_LIGHT, !digitalRead(PIN_BACK_LIGHT));
68        backLightNextBlink = millis() + TIME_BACK_LIGHT_BLINK;
69      }
70      else
71      {
72        backLightBlinkTimes = CONST_BACK_LIGHT_BLINK_TIMES;
73        backLightNextBlink = millis() + TIME_BACK_LIGHT_PASUE;
74      }
75    }
76    else if(millis() > backLightNextBlink && !batteryGood)
77    {
78      // if the battery has gone under the threshold level blink continuously
79      digitalWrite(PIN_BACK_LIGHT, !digitalRead(PIN_BACK_LIGHT));
80      backLightNextBlink = millis() + TIME_BACK_LIGHT_BLINK;
81    }
82  }
83
84  void requestEvent()
85  {
86    input = controller.battery();
87    buffer[0] = input >> 8;
88    buffer[1] = input & 0xff;
89
90    input = controller.currentLeft();
91    buffer[2] = input >> 8;
92    buffer[3] = input & 0xff;
93
94    input = controller.currentRight();
95    buffer[4] = input >> 8;
96    buffer[5] = input & 0xff;
97
98    Wire.write(buffer, 6);
99  }
100
101 void receiveEvent(int howMany)
102 {
103   static int8_t speedLeft = 0;
104   static int8_t speedRight = 0;
105
106   while(Wire.available())
```

```
107   {
108     char c = Wire.read();
109     if(c == 's')
110     {
111       speedLeft = Wire.read();
112       speedRight = Wire.read();
113       controller.setSpeed(speedLeft, speedRight);
114     }
115   }
116 }
```

**Listing 9:** Motor controller source code

```
1  /*
       ********************************************************************************
2   *
3   * Title:        WildThumper Motor Controller Library v1.0
4   * File:         wildthumper.cpp
5   * Date:         2014-05-27
6   * Author:       Karl Kangur <karl.kangur@epfl.ch>
7   *
8   ********************************************************************************
        */
9  #include <wildthumper.h>
10
11 static volatile int8_t speedLeft, speedRight;
12
13 // initialisation routine setting up the timer, enabling pins and interrupt vectors
14 WildThumper::WildThumper(void)
15 {
16   // h-bridge uses timer/counter 2 (8-bit), channels A and B
17   // stop timer, set port operations to normal and waveform generation mode to Fast
         PWM (mode 3)
18   // counter top value is 0xff (255) which gives 16MHz/256 = 62.5kHz, a prescaler is
         required
19   TCCR2A = (1 << WGM21) | (1 << WGM20);
20   // stop the timer set by Arduino (stops the default PWM)
21   TCCR2B = 0;
22
23   // set h-bridge control ports to output mode
24   DDRB |= (1 << 3);
25   DDRD |= (1 << 6) | (1 << 5) | (1 << 3);
26
27   // enable interrupt vectors
28   TIMSK2 = (1 << OCIE2A) | (1 << OCIE2B) | (1 << TOIE2);
29   // enable overflow and compare interrupts for channels A and B
30   TIFR2 = (1 << OCF2A) | (1 << OCF2B) | (1 << TOV2);
31
32   // just in case enable interrupts if not already done by Arduino
33   asm("sei");
34 }
35
36 // sets pwm for h-bridge
37 // pin mapping reference for the Atmega168: http://arduino.cc/en/Hacking/
       PinMapping168
38 void WildThumper::setSpeed(int8_t _speedLeft, int8_t _speedRight)
39 {
40   // stop the PWM by clearing the prescaler
41   TCCR2B = 0;
42
43   // reset the h-bridge by clearing all port values
44   PORTB &= ~(1 << 3);
```

```
45    PORTD &= ~((1 << 6) | (1 << 5) | (1 << 3));
46
47    // do not allow higher or lower values than 100 or -100
48    if(_speedLeft < 0)
49    {
50      speedLeft = _speedLeft < -100 ? -100 : _speedLeft;
51    }
52    else
53    {
54      speedLeft = _speedLeft > 100 ? 100 : _speedLeft;
55    }
56
57    if(_speedRight < 0)
58    {
59      speedRight = _speedRight < -100 ? -100 : _speedRight;
60    }
61    else
62    {
63      speedRight = _speedRight > 100 ? 100 : _speedRight;
64    }
65
66    // set compare interrupt
67    uint16_t temp;
68
69    // PWM compare value between 0 and 255
70    temp = (long) 255 * (speedLeft > 0 ? speedLeft : -speedLeft) / 100;
71
72    OCR2A = temp & 0xff;
73
74    temp = (long) 255 * (speedRight > 0 ? speedRight : -speedRight) / 100;
75
76    OCR2B = temp & 0xff;
77
78    // reset timer
79    TCNT2 = 0;
80
81    // set prescaler to 64 (enable timer), do not set a lower prescaler
82    // it won't work because of hardware restrictions (power transistors do not
          commute fast enough)
83    // this gives 16MHz/256/64 = 976.5625Hz, documentation says maximum frequency is
          24kHz
84    TCCR2B |= (1 << CS22);
85  }
86
87  // interrupt vectors for pin toggling
88  ISR(TIMER2_COMPA_vect)
89  {
90    if(speedLeft > 0)
91    {
92      PORTD &= ~(1 << 3);
93    }
94    else if(speedLeft < 0)
95    {
96      PORTB &= ~(1 << 3);
97    }
98  }
99
100 ISR(TIMER2_COMPB_vect)
101 {
102   if(speedRight > 0)
103   {
104     PORTD &= ~(1 << 5);
```

```
105   }
106   else if(speedRight < 0)
107   {
108     PORTD &= ~(1 << 6);
109   }
110 }
111
112 ISR(TIMER2_OVF_vect)
113 {
114   if(speedLeft > 0)
115   {
116     PORTD |= (1 << 3);
117   }
118   else if(speedLeft < 0)
119   {
120     PORTB |= (1 << 3);
121   }
122
123   if(speedRight > 0)
124   {
125     PORTD |= (1 << 5);
126   }
127   else if(speedRight < 0)
128   {
129     PORTD |= (1 << 6);
130   }
131 }
132
133 uint16_t WildThumper::battery(void)
134 {
135     return analogRead(PIN_BATTERY);
136 }
137
138 uint8_t WildThumper::currentLeft(void)
139 {
140     return analogRead(PIN_CURRENTL);
141 }
142
143 uint8_t WildThumper::currentRight(void)
144 {
145     return analogRead(PIN_CURRENTR);
146 }
147
148 uint8_t WildThumper::currentTotal(void)
149 {
150     return this->currentLeft() + this->currentRight();
151 }
```

**Listing 10:** Motor controller library

```
1  /*
       ********************************************************************************
2   *
3   * Title:        WildThumper Motor Controller Library v1.0
4   * File:         wildthumper.h
5   * Date:         2014-05-27
6   * Author:       Karl Kangur <karl.kangur@epfl.ch>
7   *
8   ********************************************************************************
        */
9  #include <Arduino.h>
10
```

```
11 #ifndef _wildthumper_h
12 #define _wildthumper_h
13
14 // Servo output definitions
15 #define S0  2
16 #define S1  4
17 #define S2  7
18 #define S3  8
19 #define S4  9
20 #define S5 10
21 #define S6 12
22
23 // Other constant definitions
24 #define PIN_BATTERY   0 // Analog input 00
25 #define PIN_CURRENTL  6 // Analog input 06
26 #define PIN_CURRENTR  7 // Analog input 07
27 #define LED           13
28
29 void setSpeed(int8_t, int8_t);
30
31 class WildThumper
32 {
33     public:
34     WildThumper(void);
35     void setSpeed(int8_t, int8_t);
36
37     uint16_t battery(void);
38     uint8_t currentLeft(void);
39     uint8_t currentRight(void);
40     uint8_t currentTotal(void);
41     uint8_t charging(void);
42 };
43
44 #endif
```

**Listing 11:** Motor controller library header