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 5

2 Introduction 6
   2.1 STI interdisciplinary robot competition 6
   2.2 Team members 6

3 Project description 7
   3.1 Competition specifications 7
      3.1.1 Arena 7
      3.1.2 Bottles 9
      3.1.3 Goals 9
   3.2 Strategy options 9
      3.2.1 Non-selective storage 9
      3.2.2 Selective storage 9
   3.3 Selected solution 10

4 Project analysis 11
   4.1 List of needs 11
   4.2 Function specification 11
      4.2.1 External 11
      4.2.2 Internal 11
   4.3 Critical technical points 11
   4.4 Solutions identification 11
      4.4.1 Movement 11
      4.4.2 Object detection 13
      4.4.3 Bottle grasping 15
      4.4.4 Localisation 17
   4.5 Risk analysis 19
   4.6 Gantt diagram 19

5 Project design 20
   5.1 Robot design 20
   5.2 Hardware 20
      5.2.1 Motors 20
      5.2.2 Servomotors 20
      5.2.3 Cage 20
   5.3 Electronics 21
      5.3.1 Raspberry Pi 21
      5.3.2 Raspberry Pi camera 21
      5.3.3 PRismino 21
      5.3.4 Motor controller 23
      5.3.5 Compass 24
      5.3.6 5V regulator 24
      5.3.7 Power 25
      5.3.8 Custom PCBs 25
   5.4 Communication between modules 26
   5.5 Mechanical design 27
      5.5.1 Bought parts 27
      5.5.2 Custom parts 27
      5.5.3 3D printed parts 28
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6 Budget management</td>
<td>29</td>
</tr>
<tr>
<td>5.7 Software</td>
<td>29</td>
</tr>
<tr>
<td><strong>6 Testing</strong></td>
<td>31</td>
</tr>
<tr>
<td>6.1 Navigation</td>
<td>31</td>
</tr>
<tr>
<td>6.1.1 Obstacle avoidance</td>
<td>31</td>
</tr>
<tr>
<td>6.1.2 Finding the recycling area</td>
<td>31</td>
</tr>
<tr>
<td>6.2 Bottle grasping</td>
<td>32</td>
</tr>
<tr>
<td>6.2.1 Electronics interfacing</td>
<td>32</td>
</tr>
<tr>
<td>6.2.2 Programming language selection</td>
<td>32</td>
</tr>
<tr>
<td>6.2.3 Benchmarks</td>
<td>33</td>
</tr>
<tr>
<td>6.3 Simulation in Webots</td>
<td>33</td>
</tr>
<tr>
<td><strong>7 Results</strong></td>
<td>36</td>
</tr>
<tr>
<td><strong>8 Conclusion</strong></td>
<td>37</td>
</tr>
<tr>
<td><strong>References</strong></td>
<td>38</td>
</tr>
<tr>
<td><strong>A Gantt diagram</strong></td>
<td>39</td>
</tr>
<tr>
<td><strong>B Chassis drawings</strong></td>
<td>41</td>
</tr>
<tr>
<td>B.1 Base</td>
<td>42</td>
</tr>
<tr>
<td>B.2 Plow</td>
<td>43</td>
</tr>
<tr>
<td>B.3 Robot assembly</td>
<td>44</td>
</tr>
<tr>
<td><strong>C Source code</strong></td>
<td>45</td>
</tr>
<tr>
<td>C.1 Controller source code in Python</td>
<td>45</td>
</tr>
<tr>
<td>C.2 PRismino source code</td>
<td>50</td>
</tr>
<tr>
<td>C.3 Motor controller source code</td>
<td>81</td>
</tr>
</tbody>
</table>
1 Abstract

The STI competition is an interdisciplinary Master semester project in the 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 $8\times8m^2$ 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](image)

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

![Bottles](image)

**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

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Principle</th>
<th>Advantage</th>
<th>Disadvantage</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom chassis</td>
<td>Custom-made chassis</td>
<td>Can take any form needed</td>
<td>Takes times to make</td>
<td>Might not finish building on time, might not work</td>
</tr>
<tr>
<td>Wild Thumper</td>
<td>Powerful differential mobile base</td>
<td>Can navigate any terrain. Readily available with the control electronics, can start working on it immediately</td>
<td>Requires adaptation</td>
<td>Motors might break when load too heavy</td>
</tr>
<tr>
<td>Rover 5</td>
<td>Differential mobile base on tracks</td>
<td>Can navigate any terrain</td>
<td>Must be ordered</td>
<td>May not be powerful enough</td>
</tr>
<tr>
<td>Cartesian robot</td>
<td>Moves in x-y axis over the whole terrain</td>
<td>Can move over any terrain</td>
<td>Too big and heavy, takes time to put in place, doesn’t really go with the spirit of this competition</td>
<td>Too expensive</td>
</tr>
<tr>
<td>Quadcopter</td>
<td>Exploits the 3rd dimension</td>
<td>Can move over any terrain, can see everything from above</td>
<td>Not allowed</td>
<td>Getting disqualified</td>
</tr>
<tr>
<td>2-wheel differential robot</td>
<td>2 wheeled robot</td>
<td>Easy to control, only 2 motors</td>
<td>Needs a 3rd passive wheel</td>
<td>May not be powerful enough</td>
</tr>
<tr>
<td>Swedish wheels</td>
<td>Wheels on wheels. 4 wheels</td>
<td>Allows movement in any direction</td>
<td>Not as precise as conventional wheels, odometry very difficult. Cannot climb slopes</td>
<td>No net gain in locomotion compared to other methods</td>
</tr>
<tr>
<td>Hexapod</td>
<td>6 legged robot</td>
<td>Navigation through complex terrain</td>
<td>Hard to program, slow, lots of parts</td>
<td>Might take too long to implement</td>
</tr>
<tr>
<td>Quadruped</td>
<td>4 legged robot</td>
<td>Can move over any terrain</td>
<td>Statically unstable, difficult to control</td>
<td>Might take too long to implement</td>
</tr>
<tr>
<td>Biped</td>
<td>2 legged robot</td>
<td>Can access all terrains</td>
<td>Extremely difficult to program and control. Not statically stable</td>
<td>Too ambitious</td>
</tr>
<tr>
<td>Strategy</td>
<td>Description</td>
<td>Advantages</td>
<td>Disadvantages</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Synchrodrive</td>
<td>All wheels turn synchronously and the chassis doesn’t rotate</td>
<td>Nothing really</td>
<td>Requires a custom chassis and lots of moving parts</td>
<td></td>
</tr>
<tr>
<td>Large robot arm</td>
<td>A serial robot that is placed in the middle of the arena and can reach any place on it</td>
<td>Can reach any area with ease</td>
<td>Extremely expensive. Difficult to put in place. Heavy, potentially dangerous</td>
<td></td>
</tr>
<tr>
<td>Crawling</td>
<td>Chassis consisting of multiple sections with actuated motors in-between</td>
<td>Can access all terrains</td>
<td>Very difficult to control, expensive and time consuming to make</td>
<td></td>
</tr>
<tr>
<td>Hopping</td>
<td>Movement with a series of jumps</td>
<td>Can move over obstacles and large distances fast</td>
<td>Hard to control, to manufacture as there aren’t any commercial products</td>
<td></td>
</tr>
<tr>
<td>Hovercraft</td>
<td>Movement on an air cushion</td>
<td>Can move over any terrain</td>
<td>Cannot move up slopes, difficult to control, noise, needs a lot of power</td>
<td></td>
</tr>
</tbody>
</table>

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 behaviours.

### 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

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Advantage</th>
<th>Disadvantage</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ultrasound</td>
<td>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</td>
<td>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</td>
<td>False positive outputs due to a large operating angle, detecting an object other than the desired target.</td>
</tr>
<tr>
<td>Infrared</td>
<td>Detect infrared light from far distances over a large area. In real-time and detect movement.</td>
<td>Incapable of distinguishing between objects.</td>
<td>Strong infrared sources might be detected as obstacles.</td>
</tr>
<tr>
<td>Laser rangefinders</td>
<td>Better accuracy more quickly. Easy alignment by employing visible red laser beam. Detects of very small targets due to small measuring spot size</td>
<td>Suffer from laser noise, stray light, and speckle effects interference.</td>
<td>Detect an object other than the desired target.</td>
</tr>
<tr>
<td>Structured light</td>
<td>Can do 3D imaging using a simple and cheap algorithm.</td>
<td>Needs lots of processing power</td>
<td>Powerful computer needed.</td>
</tr>
<tr>
<td>Tactile sensors</td>
<td>Guaranteed obstacle detection. Allow physical interaction with objects</td>
<td>Must be close enough to the obstacle, cannot avoid without physical interaction.</td>
<td>Hit the obstacles while detecting</td>
</tr>
<tr>
<td>Color sensor</td>
<td>High speed, easy to use and relative intensity display.</td>
<td>Complex calibration and limited accuracy</td>
<td>Just detect objects with certain colours</td>
</tr>
<tr>
<td>Surface transducer</td>
<td>Less sensitive to surface condition</td>
<td>Low transduction efficiency</td>
<td>Need more time and detect an object other than the desired target</td>
</tr>
<tr>
<td>Camera</td>
<td>Cheaper, more informative and more compact</td>
<td>Limitation of its view fields</td>
<td>Might not detect the whole targeted space</td>
</tr>
</tbody>
</table>
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 grasp bottles, and several strategies are listed in table 3, including the principles, advantages, disadvantages and risks corresponding to the relative strategies.

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Description</th>
<th>Challenges</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stereovision</td>
<td>Can do 3D vision.</td>
<td>Complex, poor dynamic range</td>
<td>Powerful computer needed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and still not very reliable</td>
<td></td>
</tr>
<tr>
<td>Millimeter Wave Radar</td>
<td>Accurate, excellent image identification and</td>
<td>Too expensive.</td>
<td>More expensive than other technologies</td>
</tr>
<tr>
<td></td>
<td>resolution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strategy</td>
<td>Principle</td>
<td>Advantage</td>
<td>Disadvantage</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Robotic arm</td>
<td>Arm with a few DDLs mounted onto the robot</td>
<td>Allows picking up bottle in every position and in every terrain</td>
<td>Difficult mechanical realisation and time consuming programming</td>
</tr>
<tr>
<td>Clamp</td>
<td>Grabbing a bottle in front of the robot with a clamp</td>
<td>Depending on DDLs wanted, can be very easy to realise</td>
<td>Needs good precision in positioning to grab a bottle</td>
</tr>
<tr>
<td>Suction</td>
<td>Suction mechanism to hold bottles</td>
<td>Easy pick-up and release</td>
<td>Requires compressor, energy consuming, can be hard to position on bottle</td>
</tr>
<tr>
<td>Pushing</td>
<td>Bottle being rolled with the robots chassis</td>
<td>No extra mechanical parts</td>
<td>Can be hard to perform complex movements while keeping bottle pushed. Hard when bottle positioned close to an edge or corner</td>
</tr>
<tr>
<td>Storage bay for single bottle</td>
<td>Robot ingests bottle inside a storage bay</td>
<td>Easy mechanical implementation, carrying bottle around relatively easy</td>
<td>Must return to base for every single bottle, must be well aligned with the bottle</td>
</tr>
<tr>
<td>Storage bay for multiple bottles</td>
<td>Robot ingests bottle inside a storage bay, while being able to store a few of them</td>
<td>No time lost going back to the base each time</td>
<td>Robot ingests bottle inside a storage bay, hard to implement storage system, requires a bigger robot</td>
</tr>
<tr>
<td>Deployable cage</td>
<td>Deploying a cage to surround the object, and bring it back to the base. The bottle rolls on the floor</td>
<td>Very easy to implement, bottle position doesn’t have to be exact, can grab bottle in any position or orientation</td>
<td>Can’t bring bottle over rough terrain, only one bottle at a time</td>
</tr>
</tbody>
</table>
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 planned.

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>
<thead>
<tr>
<th>Strategy</th>
<th>Advantage</th>
<th>Disadvantage</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beacon-based positioning</td>
<td>Active beacons available on the terrain</td>
<td>External conditions may influence results and interfere with sensor values</td>
<td>Sensible to environmental conditions</td>
</tr>
<tr>
<td>Odometry or encoders</td>
<td>Easy to implement in software, integrated into motors, Very precise</td>
<td>Cumulative error. Absolute positioning still required</td>
<td>Wheel slip makes robot lost immediately</td>
</tr>
<tr>
<td>Inertial measurement unit</td>
<td>Cheap and easy to integrate, ready-made boards exist with Kalman filters that return the x-y position</td>
<td>Cumulative error. Absolute positioning still required</td>
<td>Fast accelerations might disturb the system</td>
</tr>
<tr>
<td>Global positioning system</td>
<td>Absolute position anywhere on earth</td>
<td>Cannot be used inside. Consumes a lot of power. Not precise (+/-3m)</td>
<td>Won’t work</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------</td>
<td>-------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Motion field and optic flow</td>
<td>Precise, fast</td>
<td>Cumulative error. Absolute positioning still required</td>
<td></td>
</tr>
<tr>
<td>RFID</td>
<td>Available. Absolute positioning</td>
<td>Not easy to detect, need to be right over it to detect it</td>
<td>Robot might not pass over a tag for a long time</td>
</tr>
<tr>
<td>Linear cameras</td>
<td>Easy to implement</td>
<td>Needs a powerful light source from a beacon. Might interfere with other surrounding lights</td>
<td></td>
</tr>
<tr>
<td>SLAM (camera)</td>
<td>Can be made to be very robust, use existing algorithms</td>
<td>Complex to use, heavy processing needed</td>
<td>Too slow or not enough time to optimal implementation</td>
</tr>
<tr>
<td>Markovian localisation (camera)</td>
<td>Particle filter localization. Algorithm estimates the position and orientation of a robot as it moves and senses the environment</td>
<td>Terrain changes (removed litter)</td>
<td>Might not converge to actual robot position</td>
</tr>
<tr>
<td>Monte Carlo localisation (camera)</td>
<td>Grid-based localization, which uses a histogram to represent the belief distribution</td>
<td>Terrain changes (removed litter)</td>
<td>Might not converge to actual robot position</td>
</tr>
<tr>
<td>SURF, SIFT... (camera)</td>
<td>Feature point detection</td>
<td>Terrain changes (removed litter). Needs a lot of computing power</td>
<td>Might not converge to actual robot position</td>
</tr>
<tr>
<td>Color blob based localisation (camera)</td>
<td>Detect colors on images and interpret. Active beacons around the arena can be used. Terrain features are of different color</td>
<td>Bottles are transparent. Computationally expensive</td>
<td>Might not converge to actual robot position</td>
</tr>
<tr>
<td>Kinect</td>
<td>3D imaging, depth information can be useful</td>
<td>Needs a powerful computer to process data and correlate to a virtual map</td>
<td>Too time consuming</td>
</tr>
<tr>
<td>Visual odometry (egomotion) (camera)</td>
<td>Existing algorithms</td>
<td>Needs a lot of computing power</td>
<td>Computationally too expensive, cumulative error</td>
</tr>
</tbody>
</table>

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

K. Kangur, M. Starein, C. Xie
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.

![Deployable bottle cage](image)

**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).
5 PROJECT DESIGN

Figure 5: Raspberry Pi on-board computer running 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.
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.

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 I2C lines which works just as well.

![Figure 9: MPU-9150 breakout board](image)

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 the 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
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.
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.

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

| 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

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

The 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.
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>
<thead>
<tr>
<th>Provider</th>
<th>Virtual budget (CHF)</th>
<th>Real budget (CHF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distrelec</td>
<td>0.00</td>
<td>1.40</td>
</tr>
<tr>
<td>Pololu</td>
<td>0.00</td>
<td>211.37</td>
</tr>
<tr>
<td>Robopoly</td>
<td>0.00</td>
<td>68.00</td>
</tr>
<tr>
<td>Conrad</td>
<td>0.00</td>
<td>159.10</td>
</tr>
<tr>
<td>Virtual</td>
<td>472.50</td>
<td>0.00</td>
</tr>
<tr>
<td>3D printed parts</td>
<td>186.01</td>
<td>0.00</td>
</tr>
<tr>
<td>DealExtreme</td>
<td>0.00</td>
<td>8.50</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>658.51</strong></td>
<td><strong>448.37</strong></td>
</tr>
</tbody>
</table>

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 navigation 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.](image)

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 lenses’ 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 tended to roll over the bottles, because of
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)*

[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*

[9] All about openCV *Creating a haar cascade classifier aka haar training*

[10] Computer Vision Software *FAQ: OpenCV Haartraining*


[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

K. Kangur, M. Starein, C. Xie
B Chassis drawings
Thickness: 2mm

Group 5

Base

K. Kangur, M. Starein, C. Xie
B.3 Robot assembly

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

C.1 Controller source code in Python

```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 script 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"
```
self.commThread.write('l' + chr(1))

print "Enable bottle detection"
self.commThread.write('0')

while True:
    try:
        self.stateSearching()
        # check for keyboard input
        if self.checkInput():
            break
        time.sleep(0.1)
    except (KeyboardInterrupt, SystemExit):
        print "User forced exit"
        break
    except Exception as e:
        # when an error occurs make sure to end the dependant threads
        print e
        self.stopThreads()
        break

# turn off the led lights on the robot
print "Turning light off"
self.commThread.write('l' + chr(0))

print "Disable bottle detection"
self.commThread.write('0')

print "Control loop exited"
self.stopThreads()

def stopThreads(self):
    print "Stopping threads"
    self.commThread.stop()
    self.commThread.join()
    self.cameraThread.stop()
    self.cameraThread.join()

def checkInput(self):
    # check for user input
    key = waitKey(10) & 0xff
    if key == 27:
        print "Manually stopped"
        return True
    elif key == ord('w'):
        self.commThread.write('s' + chr(self.maxSpeed) + chr(self.maxSpeed))
    elif key == ord('a'):
        self.commThread.write('s' + chr(-self.maxSpeed + 255) + chr(self.maxSpeed))
    elif key == ord('d'):
        self.commThread.write('s' + chr(self.maxSpeed) + chr(-self.maxSpeed + 255))
    elif key == ord('s'):
        self.commThread.write('s' + chr(-self.maxSpeed + 255) + chr(-self.maxSpeed + 255))
    elif key == ord('1'):
        self.commThread.write('u')
    elif key == ord('2'):
        self.commThread.write('d')
    elif key == ord('q') or key == 32:
        self.commThread.write('s' + chr(0) + chr(0))
elif key != 255:
    print key
return False

def stateSearching(self):
    data = self.cameraThread.checkDataQueue()
    if data:
        self.commThread.write("!" + chr(data["position"][0]) + chr(data["position"][1])
        )
        print "Bottle detected", data["position"]

robot = RobotControl("/dev/ttyACM0", 9600)
robot.connect()

Listing 1: Python main program
```python
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# create a socket to an external website
s.connect(('google.com', 0))
ip = s.getsockname()[0].split(".")
# transform list of strings to list of integers
ip = map(int, ip)
# send the external ip address via serial to the arduino board
self.write(chr(ip[0]) + chr(ip[1]) + chr(ip[2]) + chr(ip[3]))

def stop(self):
    print "Stopping communication thread"
    self.running.set()
```

Listing 2: Communication thread

```python
#!/usr/bin/env python
import io
import picamera
import threading
import cv2
import Queue

# types of elements to detect
OBSTACLE, BOTTLE = range(2)

class ImageProcessing(threading.Thread):
    def __init__(self, width, height, preview):
        super(ImageProcessing, self).__init__()
        self.obstacleThreshold = 100000
        self.preview = preview
        self.width = width
        self.height = height
        self.camera = picamera.PiCamera()
        self.camera.resolution = (self.width, self.height)
        # boost colors
        self.camera.saturation = 100
        #self.camera.shutter_speed = 10000
        #self.camera.awb_mode = u'off'
        #self.camera.exposure_mode = u'fixedfps'
        #self.camera.meter_mode = u'spot'
        #self.camera.exposure_compensation = 10
        #self.camera.framerate = 2
        #self.camera.sharpness = 0
        #self.camera.video_stabilization = True
        #self.configureCamera()
        # load the xml file for
        self.cascadeXml = cv2.CascadeClassifier('bottle.xml')
        # load the mask that hides the non important parts of the image
        self.mask = cv2.imread("mask.png", 0)
        self.stream = io.BytesIO()
        self.dataQueue = Queue.Queue()
```

K. Kangur, M. Starein, C. Xie
# if enabled show the preview window
if self.preview:
    cv2.namedWindow("Preview", flags=cv2.CV_WINDOW_AUTOSIZE)

# event to stop the thread
self.running = threading.Event()

def run(self):
    print "Image processing thread started"
    # make sure we're still running
    while not self.running.isSet():
        # take the picture
        self.camera.capture(self.stream, format='jpeg', use_video_port=True)
        # construct a numpy array from the stream
        data = np.fromstring(self.stream.getvalue(), dtype=np.uint8)
        self.stream.truncate()
        self.stream.seek(0)
        # "decode" the image from the array, preserving colour in BGR format
        self.img = cv2.imdecode(data, cv2.CV_LOAD_IMAGE_COLOR)
        # image that can be modified by the image processing for preview purposes
        if self.preview:
            self.previewImage = self.img
            #self.detectObstacles()
            self.detectBottles()
            if self.preview:
                try:
                    cv2.imshow("Preview", self.previewImage)
                except:
                    pass
        # destroy the preview window
        if self.preview:
            cv2.destroyAllWindows()
    def stop(self):
        print "Stopping image processing thread"
        self.running.set()

def detectBottles(self):
    # get a gray picture
    gry = cv2.cvtColor(self.img, cv2.COLOR_BGR2GRAY)
    # run the haar cascade algorithm (slow)
    haar = selfcascadeXml.detectMultiScale(gry, 1.1, 6)
    # update preview
    if self.preview:
        for (x,y,w,h) in haar:
            cv2.rectangle(self.previewImage, (x,y), (x+w, y+h), (255, 255, 255), 1)
    # normalised value between 0 and 1 of the first bottle position
    if len(haar):
        bottleX = (haar[0][0] + haar[0][2] / 2);
        bottleY = self.height - (haar[0][1] + haar[0][3] / 2);
        self.dataQueue.put({"type": BOTTLE, "position": (bottleX, bottleY)})
def detectObstacles(self):
    colorFloorL = np.array([8, 30, 30], np.uint8)
    colorFloorH = np.array([28, 255, 255], np.uint8)
    # convert to hue, saturation, value format
    hsv = cv2.cvtColor(self.img, cv2.COLOR_BGR2HSV)
    colorThreshold = cv2.inRange(hsv, colorFloorL, colorFloorH)
    colorThreshold = np.invert(colorThreshold)
    cv2.erode(colorThreshold, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7)),
              colorThreshold, (-1, -1), 1)
    np.bitwise_and(colorThreshold, self.mask, colorThreshold)
    # average values vertically and split screen in 4 horizontally
    sumVertical = np.sum(colorThreshold, axis = 0)
    splitHorizontal = np.array([np.sum(sumVertical[0:63]), np.sum(sumVertical[64:127]), np.sum(sumVertical[128:191]), np.sum(sumVertical[192:255])])
    # print "H", splitHorizontal
    if self.preview:
        # show obstacle position with the overlay on the image
        mask = np.invert(colorThreshold)
        np.bitwise_and(self.previewImage[:,:,0], mask, self.previewImage[:,:,0])
        np.bitwise_and(self.previewImage[:,:,1], mask, self.previewImage[:,:,1])
        np.bitwise_and(self.previewImage[:,:,2], mask, self.previewImage[:,:,2])
        pass
    peakX = splitHorizontal.argmax()
    # obstacle threshold has been reached, obstacle has been detected
    if splitHorizontal[peakX] > self.obstacleThreshold:
        # average values horizontally and split screen in 4 vertically
        sumHorizontal = np.sum(colorThreshold, axis = 1)
        splitVertical = np.array([np.sum(sumHorizontal[0:31]), np.sum(sumHorizontal[32:63]), np.sum(sumHorizontal[64:95]), np.sum(sumHorizontal[96:127])])
        #print "V", splitVertical
        peakY = splitVertical.argmax()
        self.dataQueue.put({"type": OBSTACLE, "position": (peakX, peakY)})

def checkDataQueue(self):
    if not self.dataQueue.empty():
        return self.dataQueue.get()
    else:
        return False

Listing 3: Image processing thread

C.2 PRismino source code

/*
  **********************************************************************************
  * Title: STI competition Arduino code for the Tokamak robot.
  * Date:    2014-06-06
  */

K. Kangur, M. Starein, C. Xie
```c
#include <Servo.h>
#include <prismino.h>
#include <Wire.h>
#include "robot.h"

Tokamak robot;

// transition structure
struct transition
{
    enum state_codes state_source;
    enum return_codes return_code;
    enum state_codes state_destination;
};

// state functions and codes must be in sync
return_codes (*state[])(void) = {
    stateSearching,
    stateFetchingBottle,
    stateLowerCage,
    stateGoHome,
    stateRaiseCage
};

struct transition state_transitions[] = {
    {STATE_SEARCHING, OK, STATE_FETCHING_BOTTLE},
    {STATE_SEARCHING, REPEAT, STATE_SEARCHING},
    {STATE_FETCHING_BOTTLE, OK, STATE_LOWER_CAGE},
    {STATE_FETCHING_BOTTLE, REPEAT, STATE_FETCHING_BOTTLE},
    {STATE_FETCHING_BOTTLE, FAIL, STATE_SEARCHING},
    {STATE_LOWER_CAGE, OK, STATE_GO_HOME},
    {STATE_GO_HOME, REPEAT, STATE_GO_HOME},
    {STATE_GO_HOME, OK, STATE_RAISE_CAGE},
    {STATE_RAISE_CAGE, OK, STATE_SEARCHING}
};

enum state_codes currentState;
enum return_codes returnCode;
comm_methods inputMethod;

// pointer to the current called function in the state machine
return_codes (*stateFunction)(void);

// other global variables
volatile uint8_t bottlePosition;
volatile uint8_t bottleDistance;
sides sideWall;

volatile uint32_t timeBottleLastSeen;
volatile uint32_t timeNextBottleCheck;
uint32_t timeCheckBattery;
volatile boolean booleanBottleUpdate;
boolean booleanBottleSecondTry;
boolean booleanFalseBottle;

// ############################################################ SETUP

void setup()
{
    // set pin output mode (sources current)
}
```
pinMode(LED, OUTPUT);
pinMode(PIN_LIGHTS, OUTPUT);

// enable button pull-up
pinMode(BTN, INPUT);
digitalWrite(BTN, HIGH);

// play a sound on boot, repeated sounds will indicate very low battery voltage
robot.playSound(ONEUP);

// initialise serial bus for communication with the Raspberry Pi
Serial.begin(9600);

// initialise the bus for communication with the computer via Bluetooth
#ifdef ENABLE_BLUETOOTH
Bluetooth.begin(9600);
#endif

// join i2c bus as master
Wire.begin();

// disable internal pull-ups to 5V as there are external 2K pull-ups to 3.3V
digitalWrite(SDA, LOW);
digitalWrite(SCL, LOW);

#ifdef ENABLE_COMPASS
// wait for the imu to boot
delay(500);
// put the HMC5883 IC into the correct operating mode
// open communication with HMC5883
Wire.beginTransmission(I2C_COMPASS_ADDRESS);
// select mode register
Wire.write(0x02);
// continuous measurement mode
Wire.write(0x00);
Wire.endTransmission();
#endif

// initialise global variables
bottlePosition = 0;
bottleDistance = 0;
timeCheckBattery = millis() + TIME_CHECK_BATTERY;
timeBottleLastSeen = 0;
sideWall = RIGHT;
booleanBottleUpdate = 0;

// reset the robot state
reset();
currentState = ENTRY_STATE;
}

// main loop
void loop()
{
  if(Serial.available())
  {
    processInput(USB);
  }
}
```c
#ifdef ENABLE_BLUETOOTH
if(Bluetooth.available())
{
    processInput(BLUETOOTH);
}
#endif

uint16_t irLeft, irCenterLeft, irCenterRight, irRight;
robot.readIrSensors(& irLeft, &irCenterLeft, &irCenterRight, &irRight);

Bluetooth.print(irLeft);
Bluetooth.print("\t");
Bluetooth.print(irCenterLeft);
Bluetooth.print("\t");
Bluetooth.print(irCenterRight);
Bluetooth.print("\t");
Bluetooth.println(irRight);

//Bluetooth.println(robot.getHeading());

// toggle robot running state via the button on the shield
if(!digitalRead(BTN))
{
    robot.playSound(POWERUP);
    robot.flags.running = !robot.flags.running;

    // robot has been stopped
    if(!robot.flags.running)
    {
        robot.stop();
        currentState = ENTRY_STATE;
    }

    // wait for button debounce
    delay(1000);
}
#endif

if(robot.flags.running)
{
    // current function to call according to the state machine
    stateFunction = state[currentState];
    // actually call the function
    returnCode = stateFunction();
    // fetch next state
    currentState = lookupTransitions(currentState, returnCode);
}

#include "state_transitions.h"

// returns the new state according to the current state and the return value
state_codes lookupTransitions(state_codes state, return_codes code)
{
```
```c
uint8_t i;
// default return state is the entry state
state_codes nextState = ENTRY_STATE;
// see if a state transition matches and switch to the next state
for(i = 0; i < sizeof(state_transitions) / sizeof(transition); i++)
{
  if(state_transitions[i].state_source == state && state_transitions[i].
    return_code == code)
  {
    nextState = state_transitions[i].state_destination;
    if(nextState != state_transitions[i].state_source)
    {
      robot.playSound(COIN);
      break;
    }
  }
  return nextState;
}

// ############################################################ ROBOT STATES

return_codes stateSearching()
{
  // check if a bottle has been seen and change state
  if(bottlePosition && millis() > timeNextBottleCheck)
  {
    robot.stop();
    booleanBottleSecondTry = 0;
    booleanFalseBottle = 1;
    return OK;
  }

  // make robot roam the arena set deviation to 0 to go straight when there are no
  obstacles
  robot.headTo(0);
  return REPEAT;
}

return_codes stateFetchingBottle()
{
  // check that the last time a bottle has been seen doesn’t exceed a limit
  if(millis() > timeBottleLastSeen)
  {
    timeNextBottleCheck = millis() + TIME_NEXT_BOTTLE_CHECK;
  }

  // a bottle hasn’t been seed since TIME_BOTTLE_SEEN_TIMEOUT milliseconds, it was
  probably a false positive
  if(booleanBottleSecondTry)
  {
    bottlePosition = 0;
    return FAIL;
  }

  // back up a little bit just to be sure it was a false positive
  booleanBottleSecondTry = 1;

  robot.setSpeed(-CONST_SPEED_BOTTLE, -CONST_SPEED_BOTTLE);
  delay(500);
  robot.stop();
  delay(1000);
```
// make the 0-255 bottle position value signed
int8_t deviation = 127 - bottlePosition;
static uint32_t timeBottleApproachingLastCheck = 0;

if(deviation > -CONST_DEVIATION_OK && deviation < CONST_DEVIATION_OK)
{
    // 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
    if(bottleDistance > CONST_BOTTLE_SIZE_LOWER_CAGE || millis() > timeBottleApproachingLastCheck)
    {
        // if the bottle magically appeared in front of the robot consider it a false positive
        if(booleanFalseBottle)
        {
            bottlePosition = 0;
            return FAIL;
        }
        // just to be sure move forwards for a little while
        robot.setSpeed(CONST_SPEED_BOTTLE, CONST_SPEED_BOTTLE);
        delay(500);
        robot.stop();
        return OK;
    }
    // approach the bottle only if the robot got an update on the bottle position
    if(booleanBottleUpdate)
    {
        robot.setSpeed(CONST_SPEED_BOTTLE, CONST_SPEED_BOTTLE);
        booleanBottleUpdate = 0;
        timeBottleApproachingLastCheck = millis() + TIME_GRAB_LOST_BOTTLE;
    }
    // head towards the bottle at a lower speed
    if(booleanBottleUpdate)
    {
        robot.turn(deviation, CONST_SPEED_BOTTLE);
        delay(CONST_SPEED_SET_DELAY);
        robot.stop();
        booleanBottleUpdate = 0;
        timeBottleApproachingLastCheck = millis() + TIME_GRAB_LOST_BOTTLE;
    }
    // the bottle was not perfectly detected in front of the robot the first time, so it’s probably a true bottle
    booleanFalseBottle = 0;
    return REPEAT;
}
return_codes stateLowerCage()
{
    // make sure the wheels are stopped
    robot.stop();
    robot.setCagePosition(CAGE_DOWN);
    robot.playSound(ONEUP);
    return OK;
```c
/*return_codes stateFindWall()
{
    uint16_t irLeft, irCenterLeft, irCenterRight, irRight;
    robot.readIrSensors(&irLeft, &irCenterLeft, &irCenterRight, &irRight);
    // get the heading angle between -180 and 180 degrees
    int16_t deviation = robot.getHeading() - CONST_HEADING_HOME;
    // must check of it will enter an infinite loop
    static uint32_t timeAntiRecheck = 0;
    static uint32_t timeTurnTime;
    if(
        millis() > timeAntiRecheck &&
        (irLeft > CONST_IR_OBSTACLE_SIDE_CAGE || irRight > CONST_IR_OBSTACLE_SIDE_CAGE)
    )
    {
        // if the robot is too close to a wall it's impossible for it to be a wall
        if(irLeft > CONST_IR_TOO_CLOSE || irRight > CONST_IR_TOO_CLOSE)
        {
            timeAntiRecheck = millis() + TIME_WALL_RECHECK;
            return REPEAT;
        }
        // see if this is an obstacle or a wall
        // 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
        if(irLeft > irRight)
        {
            // the "obstacle" was on the left, turn towards the left to check for a wall
            sideWall = LEFT;
            timeTurnTime = millis();
            robot.setSpeed(-CONST_SPEED_OBSTACLE, CONST_SPEED_OBSTACLE);
            // wait until the right sensor sees the obstacle
            while(analogRead(SENSOR_IR_RIGHT) < CONST_IR_OBSTACLE_SIDE_CAGE);
            // stop before checking the other IR sensor
            timeTurnTime = millis() - timeTurnTime;
            robot.setSpeed(0, 0);
            // check if the left sensor still sees the obstacle, if yes it's a wall
            if(analogRead(SENSOR_IR_LEFT) > CONST_IR_OBSTACLE_FAR)
            {
                return OK;
            }
        }
        // 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
        else
        {
            // the "obstacle" was on the right, turn towards the right to check for a wall
            sideWall = RIGHT;
            timeTurnTime = millis();
            robot.setSpeed(CONST_SPEED_OBSTACLE, -CONST_SPEED_OBSTACLE);
            // wait until the right sensor sees the obstacle
            while(analogRead(SENSOR_IR_LEFT) < CONST_IR_OBSTACLE_SIDE_CAGE);
        }
    }
}
```
// stop before checking the other IR sensor
    timeTurnTime = millis() - timeTurnTime;
    robot.setSpeed(0, 0);

    // check if the left sensor still sees the obstacle, if yes it's a wall
    if(analogRead(SENSOR_IR_RIGHT) > CONST_IR_OBSTACLE_FAR)
        return OK;
    }

    // false positive, turn back
    if(sideWall == LEFT)
        {       
            robot.setSpeed(CONST_SPEED_OBSTACLE, -CONST_SPEED_OBSTACLE);
            }
    else
        {       
            robot.setSpeed(-CONST_SPEED_OBSTACLE, CONST_SPEED_OBSTACLE);
            delay(timeTurnTime);
            }
    timeAntiRecheck = millis() + TIME_WALL_RECHECK;
  }

    // head towards the direction the compass indicates
    robot.headTo(deviation);

    return REPEAT;
*/

return_codes stateGoHome()
{
    return robot.goHome();

    /*uint16_t irLeft, irCenterLeft, irCenterRight, irRight;
    robot.readIrSensors(&irLeft, &irCenterLeft, &irCenterRight, &irRight);
    int16_t deviation = robot.getHeading() - CONST_HEADING_HOME;
    // at this point we know the robot is in front of the wall
    Serial.println(deviation);
    // turn towards the home heading within a margin
    Serial.println(deviation);

    if(deviation > CONST_DEVIATION_OK)
        {       
            robot.setSpeed(-CONST_SPEED_BOTTLE, CONST_SPEED_BOTTLE);
            }
    else if(deviation < -CONST_DEVIATION_OK)
        {       
            robot.setSpeed(CONST_SPEED_BOTTLE, -CONST_SPEED_BOTTLE);
            }
    else if(irLeft > irRight)
        {
            // wall is on the left, turn left a little bit for the next state
            robot.setSpeed(CONST_SPEED_MAX, 0);
            delay(TIME_TURN_FOLLOW_WALL);
            robot.setSpeed(0, 0);
            sideWall = LEFT;
            return OK;
        }
```c

else if(irRight > irLeft) {
    // wall is on the right, turn left a little bit for the next state
    robot.setSpeed(0, CONST_SPEED_MAX);
    delay(TIME_TURN_FOLLOW_WALL);
    robot.setSpeed(0, 0);
    sideWall = RIGHT;
    return OK;
}
else {
    robot.headTo(deviation);
}
return REPEAT;/*

return_codes stateFollowWall () {
    uint16_t irLeft, irCenterLeft, irCenterRight, irRight;
    robot.readIrSensors(& irLeft, & irCenterLeft, & irCenterRight, & irRight);
    // follow the left wall
    if(sideWall == LEFT ) {
        // a wall was detected on the right, it can only be the goal
        if(irRight > CONST_IR_OBSTACLE_SIDE_CAGE) {
            robot.stop();
            return OK;
        }
        // go straight with an offset to the left
        //robot.headTo(CONST_SPEED_MAX - 1);
        robot.followWall(& irLeft, LEFT);
    }
    else {
        // a wall was detected on the left, it can only be the goal
        if(irLeft > CONST_IR_OBSTACLE_SIDE_CAGE) {
            robot.stop();
            return OK;
        }
        // go straight with an offset to the right
        //robot.headTo(-(CONST_SPEED_MAX - 1));
        robot.followWall(& irRight, RIGHT);
    }
    return REPEAT;
}

return_codes stateRaiseCage () {
    robot.setCagePosition(CAGE_UP);
    // announce the glorious point it just probably got
    //robot.playSound(FLAGPOLE);
    if(robot.flags.wall == WALL_RIGHT)
```
C SOURCE CODE

```c
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    while(robot.getHeading() < 160 && robot.getHeading() > -160)
499    {
500        if(robot.flags.wall == WALL_LEFT)
501            robot.setSpeed(CONST_SPEED_MAX, -CONST_SPEED_MAX);
502        else
503            robot.setSpeed(-CONST_SPEED_MAX, CONST_SPEED_MAX);
504    }
505    robot.stop();
506    // reset the robot state before restarting
507    reset();
508    return OK;
509```

```c
510    }
511    void reset()
512    {
513        robot.flags.wall = NO_WALL;
514        bottlePosition = 0;
515        timeNextBottleCheck = millis() + TIME_NEXT_BOTTLE_CHECK;
516        booleanBottleSecondTry = 0;
517    }
518    // ############################################################ USER INPUT METHODS
519    void processInput(comm_methods method)
520    {
521        inputMethod = method;
522        digitalWrite(LED, HIGH);
523        switch(input())
524        {
525            case '0':
526                // force robot state to searching
527                output("Toggle robot\n");
528                bottlePosition = 0;
529                robot.flags.running = !robot.flags.running;
530                if(!robot.flags.running)
531                    robot.stop();
532                robot.playSound(POWERUP);
533                break;
534            case '1':
535                output("Set state: searching\n");
536                currentState = STATE_SEARCHING;
537                break;
538            case '2':
```

K. Kangur, M. Starein, C. Xie

59
```c
output("Set state: fetching bottle\n");
currentState = STATE_FETCHING_BOTTLE;
break;
case '3':
  output("Set state: go home\n");
currentState = STATE_LOWER_CAGE;
break;
case '4':
  output("Set state: follow wall\n");
currentState = STATE_GO_HOME;
break;
case '5':
  output("Set state: raise cage\n");
currentState = STATE_RAISE_CAGE;
break;
case 'B':
  // a bottle was seen
  bottlePosition = input();
bottleDistance = input();
timeBottleLastSeen = millis() + TIME_BOTTLE_SEEN_TIMEOUT;
booleanBottleUpdate = 1;
break;
case 'S':
  robot.setSpeed(input(), input());
break;
case 't':
  play((input() << 8) | input(), 500);
break;
case ',' :
  output("Lower cage\n");
  robot.setCagePosition(CAGE_DOWN);
break;
case '.' :
  output("Raise cage\n");
  robot.setCagePosition(CAGE_UP);
break;
case 'm':
  output("Turn lights on\n");
  robot.setLights(true);
break;
case 'n':
  output("Turn lights off\n");
  robot.setLights(false);
break;
case 'w':
  output("Go forwards\n");
  robot.setSpeed(CONST_SPEED_MAX, CONST_SPEED_MAX);
break;
case 'a':
  output("Go left\n");
  robot.setSpeed(-CONST_SPEED_MAX, CONST_SPEED_MAX);
break;
case 's':
  output("Go backwards\n");
  robot.setSpeed(-CONST_SPEED_MAX, -CONST_SPEED_MAX);
break;
case 'd':
  output("Go right\n");
  robot.setSpeed(CONST_SPEED_MAX, -CONST_SPEED_MAX);
break;
case 'q':
  output("Stop\n");
```

Listing 4: Arduino main program

#include <Servo.h>
#include <Wire.h>
#include <prismino.h>
#include "robot.h"
#include "pitch.h"
#include "sound.h"

Tokamak::Tokamak()
{
    // initialise default values
    this->flags.enableFrontLeds = false;
    this->flags.running = false;
    this->flags.cagePosition = CAGE_UP;
}

void Tokamak::setCagePosition(cage_positions position)
{
    this->servoRight.attach(S1);
    this->servoLeft.attach(S2);
}

char input()
{
    if(inputMethod == USB)
    {
        return Serial.read();
    }
    else if (inputMethod == BLUETOOTH)
    {
        return Bluetooth.read();
    }
}

char output(const char * data)
{
    if(inputMethod == USB)
    {
        Serial.print(data);
    }
    else if (inputMethod == BLUETOOTH)
    {
        Bluetooth.print(data);
    }
}

robot.stop();
break;
default:
    output("Command not recognised\n");
digitalWrite(LED, LOW);
uint8_t r, l;

if(position == CAGE_UP)
{
    // both sevomotors have 160 steps between up and down positions, so this is
    // allowed
    for(l = SERVO_LEFT_DOWN , r = SERVO_RIGHT_DOWN ; r < SERVO_RIGHT_UP ; r++, l--)
    {
        this->servoLeft.write(l);
        this->servoRight.write(r);
        delay(CONST_MAX_SERVO_SPEED);
    }
} else if(position == CAGE_DOWN)
{
    for(l = SERVO_LEFT_UP , r = SERVO_RIGHT_UP ; r > SERVO_RIGHT_DOWN ; r--, l++)
    {
        this->servoLeft.write(l);
        this->servoRight.write(r);
        delay(CONST_MAX_SERVO_SPEED);
    }
}

// always detach the motors so that they are free-running and do not consume power
// in resting positions
this->servoLeft.detach();
this->servoRight.detach();
this->flags.cagePosition = position;

void Tokamak::setSpeed(int8_t speedLeft, int8_t speedRight)
{
    // controller has been disabled
    #ifndef ENABLE_CONTROLLER
    return;
    #endif

    Wire.beginTransmission(I2C_MOTOR_CONTROLLER_ADDRESS);
    Wire.write("s");

    // make sure not to go over the maximum speed limit (100 or -100)
    if(speedLeft > CONST_SPEED_MAX)
    {
        Wire.write(CONST_SPEED_MAX);
    } else if(speedLeft < -CONST_SPEED_MAX)
    {
        Wire.write(-CONST_SPEED_MAX);
    } else
    {
        Wire.write(speedLeft);
    }

    if(speedRight > CONST_SPEED_MAX)
    {
        Wire.write(CONST_SPEED_MAX);
    }
else if (speedRight < -CONST_SPEED_MAX)
{
    Wire.write(-CONST_SPEED_MAX);
}
else
{
    Wire.write(speedRight);
}
Wire.endTransmission();

// a small delay is needed so that the speed could actually be applied to the
wheels
delay(CONST_SPEED_SET_DELAY);
}

void Tokamak::stop()
{
    this->setSpeed(0, 0);
}

void Tokamak::checkBattery()
{
    Wire.beginTransmission(I2C_MOTOR_CONTROLLER_ADDRESS);

    // casting needed for some reason
    uint8_t available = Wire.requestFrom((uint8_t)I2C_MOTOR_CONTROLLER_ADDRESS, (uint8_t)6);

    // read 6 bytes, 2 by 2, high byte first: voltage, left motor current, right motor
    current
    if(available == 6)
    {
        this->batteryVoltage = (Wire.read() << 8) | Wire.read();
        this->currentLeft = (Wire.read() << 8) | Wire.read();
        this->currentRight = (Wire.read() << 8) | Wire.read();
    }
    else
    {
        // inform i2c error
        play(TONE_I2C_ERROR, 500);
    }
    Wire.endTransmission();

    if(this->batteryVoltage < CONST_BATTERY_LOW)
    {
        play(TONE_BATTERY, 500);
    }

void Tokamak::setLights(boolean state)
{
    digitalWrite(PIN_LIGHTS, state);
}

int16_t Tokamak::getHeading()
{
    #ifndef ENABLE_COMPASS
    return 0;
    #endif
int16_t x = 0, y = 0, z = 0;

Wire.beginTransmission(I2C_COMPASS_ADDRESS);
// select register 3, X MSB register
Wire.write(0x03);
Wire.endTransmission();

Wire.requestFrom(I2C_COMPASS_ADDRESS, 6);
if(6 <= Wire.available())
{
    x = (Wire.read() << 8) | Wire.read();
    z = (Wire.read() << 8) | Wire.read();
    y = (Wire.read() << 8) | Wire.read();
}
else
{
    // inform i2c error
    play(TONE_I2C_ERROR, 500);
    return 0;
}

int16_t angle = atan2(x, z) * 180 / M_PI;
return angle;

void Tokamak::turn(int16_t deviation, int8_t speed)
{
    int8_t speedLeft = 0;
    int8_t speedRight = 0;

    // limit deviation to maximum allowed speed
    if(deviation > speed)
    {
        deviation = speed;
    }
    else if(deviation < -speed)
    {
        deviation = -speed;
    }
    if(deviation < -CONST_DEVIATION_OK)
    {
        speedLeft = speed;
        speedRight = -speed;
    }
    else if(deviation > CONST_DEVIATION_OK)
    {
        speedLeft = -speed;
        speedRight = speed;
    }
    else
    {
        speedLeft = 0;
        speedRight = 0;
    }
    this->setSpeed(speedLeft, speedRight);
}

void Tokamak::headTo(int16_t deviation, int8_t speed)
{
    navigation_avoidance(deviation);
}
```c
return;

/*int8_t speedLeft = 0;
int8_t speedRight = 0;

uint16_t irLeft, irCenterLeft, irCenterRight, irRight;
this->readIrSensors(&irLeft, &irCenterLeft, &irCenterRight, &irRight);

// these values are incremented over time and reset after every TIME_ANTI_LOOP
// seconds
static uint8_t antiLoopCountLeft = 0, antiLoopCountRight = 0;
static uint32_t timeAntiLoop = 0;

// every TIME_ANTI_LOOP seconds reset the anti-loop timer and wheel counters
if(millis() > timeAntiLoop)
{
    antiLoopCountLeft = 0;
    antiLoopCountRight = 0;
    timeAntiLoop = millis() + TIME_ANTI_LOOP;
}

// limit deviation to maximum allowed speed
if(deviation > speed)
{
    deviation = speed;
}
else if(deviation < -speed)
{
    deviation = -speed;
}

// check if the robot has entered an infinite loop
if(antiLoopCountLeft > CONST_LOOP_TURN_TIMES && antiLoopCountRight >
CONST_LOOP_TURN_TIMES)
{
    // if the cage is deployed reverse a bit
    if(this->flags.cagePosition == CAGE_UP)
    {
        this->setSpeed(-speed, -speed);
        delay(500);
    }

    // set speed immediately
    if(deviation > 0)
    {
        this->setSpeed(speed, -speed);
    }
    else
    {
        this->setSpeed(-speed, speed);
    }

    delay(TIME_ANTI_LOOP_TIMEOUT_TURN);

    // reset the counters
    antiLoopCountLeft = 0;
    antiLoopCountRight = 0;
}

// avoid obstacles using the IR sensors
else if(irCenterRight > CONST_IR_OBSTACLE_CENTER_CAGE)
{  
```
// turn on itself
speedLeft = speed;
speedRight = -speed;
antiLoopCountLeft++;
}
else if(irCenterLeft > CONST_IR_OBSTACLE_CENTER_CAGE)
{
    speedLeft = -speed;
    speedRight = speed;
    antiLoopCountRight++;
}
else if(irRight > CONST_IR_OBSTACLE_SIDE_CAGE)
{
    // block one wheel, reverse the other
    speedLeft = -speed;
    speedRight = 0;
    antiLoopCountLeft++;
}
else if(irLeft > CONST_IR_OBSTACLE_SIDE_CAGE)
{
    speedLeft = 0;
    speedRight = -speed;
    antiLoopCountRight++;
}
else if(deviation < -CONST_DEVIATION_OK)
{
    speedLeft = speed;
    speedRight = speed + deviation;
}
else if(deviation > CONST_DEVIATION_OK)
{
    speedLeft = speed - deviation;
    speedRight = speed;
}
else
{
    // simply go forwards
    speedLeft = speed;
    speedRight = speed;
}
this->setSpeed(speedLeft, speedRight);*/
}

void Tokamak::followWall(uint16_t *sensorValue, sides side, int8_t speed)
{
    if(side == RIGHT && *sensorValue < CONST_IR_WALL_FOLLOW_MIN)
    {
        this->setSpeed(speed, speed >> 2);
    }
    else if(side == RIGHT && *sensorValue > CONST_IR_WALL_FOLLOW_MAX)
    {
        this->setSpeed(speed >> 2, speed);
    }
    else if(side == LEFT && *sensorValue < CONST_IR_WALL_FOLLOW_MIN)
    {
        this->setSpeed(speed >> 2, speed);
    }
    else if(side == LEFT && *sensorValue > CONST_IR_WALL_FOLLOW_MAX)
    {
        this->setSpeed(speed, speed >> 2);
    }
}
void Tokamak::readIrSensors(uint16_t *irLeft, uint16_t *irCenterLeft, uint16_t *irCenterRight, uint16_t *irRight)
{
    *irLeft = analogRead(SENSOR_IR_LEFT);
    *irCenterLeft = analogRead(SENSOR_IR_CENTER_LEFT);
    *irCenterRight = analogRead(SENSOR_IR_CENTER_RIGHT);
    *irRight = analogRead(SENSOR_IR_RIGHT);
}

// a sound is an array of notes

note notesCoin[] = {
    {B5, 100},
    {E6, 200}
};

sound soundCoin = {sizeof(notesCoin) / sizeof(note), notesCoin};

note notesPowerUp[] = {
    {G3, 50},
    {B4, 50},
    {D4, 50},
    {G4, 50},
    {B5, 50},
    {A4b, 50},
    {C4, 50},
    {E4b, 50},
    {A5b, 50},
    {C5, 50},
    {B5b, 50},
    {D5, 50}
};

sound soundPowerUp = {sizeof(notesPowerUp) / sizeof(note), notesPowerUp};

note notesOneUp[] = {
    {E4, 100},
    {G4, 100},
    {E5, 100},
    {C5, 100},
    {D5, 100},
    {G5, 100}
};

sound soundOneUp = {sizeof(notesOneUp) / sizeof(note), notesOneUp};

note notesFlagpoleFanfare[] = {
    {G2, 100},
    {C3, 100},
    {E3, 100},
    {G3, 100},
    {C4, 100},
    {E4, 100},
    {G4, 300},
    {E4, 300},
    {A2b, 100},
    {C3, 100},
    {E3b, 100},
sound soundFlagpoleFanfare = {sizeof(notesFlagpoleFanfare) / sizeof(note),
    notesFlagpoleFanfare};

// function that actually plays the sounds
void Tokamak::playSound(sounds theSound)
{
    #ifndef ENABLE_SOUND
        return;
    #endif

    sound soundPtr;

    // pointer to the sound object
    switch (theSound)
    {
        case COIN:
            soundPtr = soundCoin;
            break;
        case POWERUP:
            soundPtr = soundPowerUp;
            break;
        case ONEUP:
            soundPtr = soundOneUp;
            break;
        case FLAGPOLE:
            soundPtr = soundFlagpoleFanfare;
            break;
    }

    uint16_t frequency, duration;
    uint8_t length = soundPtr.length;
    for(uint8_t i = 0; i < length; i++)
    {
        // get data from program memory
        frequency = soundPtr.notes[i].pitch;
        duration = soundPtr.notes[i].duration;

        // if the next note is the same then make a short pause
        uint8_t pause = 0;
        if(i < length - 1 && frequency == soundPtr.notes[i + 1].pitch)
        {
            // 5 millisecond pause
            pause = 5;
        }
// play the right pitch for the determined duration
play(frequency, duration - pause);

// play is not a blocking function so one has to manually set a delay
delay(duration);
}

void Tokamak::resetReturnBase(void)
{
  this->flags.wall = NO_WALL;
}

return_codes Tokamak::goHome(int16_t direction_home)
{
  //tell RASP to stop looking for bottles -> accelerates image processing
  //tell RASP to start looking for beacon
  getHeading();
gotoHeading(direction_home);
  uint16_t irFarLeft = analogRead(SENSOR_IR_LEFT);
  uint16_t irFarRight = analogRead(SENSOR_IR_RIGHT);

  if(this->flags.wall == NO_WALL)
  {
    this->flags.wall = detectWall();
gotoHeading(direction_home);
    if(this->flags.wall != NO_WALL)
    {
      //Serial.println("wall detected");
      this->setSpeed(0,0);
    }
  }
  else
  {
    if(this->flags.wall == WALL_LEFT)
    {
      // Serial.println("WALL LEFT");
      if(irFarRight > CONST_IR_OBSTACLE_SIDE_CAGE)
      {
        this->setSpeed(0,0);
        play(440, 500);
        return OK;
      }
      else navigation_avoidance(-6);
    }
    else if(this->flags.wall == WALL_RIGHT)
    {
      // Serial.println("WALL RIGHT");
      if(irFarLeft > CONST_IR_OBSTACLE_SIDE_CAGE)
      {
        this->setSpeed(0,0);
        play(440, 500);
        return OK;
      }
      else navigation_avoidance(6);
    }
    else if(this->flags.wall != WALL_LEFT &&
             this->flags.wall != WALL_RIGHT &&
             this->flags.wall != NO_WALL)
    {
      //Serial.println("Error: Wall not detected");
      this->setSpeed(0,0);
      play(440, 500);
      return OK;
    }
    else
    {
      //Serial.println("WALL RIGHT");
      if(irFarLeft > CONST_IR_OBSTACLE_SIDE_CAGE)
      {
        this->setSpeed(0,0);
        play(440, 500);
        return OK;
      }
      else navigation_avoidance(6);
    }
  }
  return REPEAT;
  //if(beacon_detected) //stop, open cage, wallSide = 0;
}
uint8_t Tokamak::detectWall(void)
{
    uint16_t irFarLeft = analogRead(SENSOR_IR_LEFT);
    uint16_t irFarRight = analogRead(SENSOR_IR_RIGHT);
    static uint32_t timeAntiRecheck;

    if((irFarLeft < 100 && irFarRight < CONST_IR_OBSTACLE_SIDE_CAGE) || (irFarLeft < CONST_IR_OBSTACLE_SIDE_CAGE && irFarRight < 100))
    {
        return NO_WALL; // no wall detected
        timeAntiRecheck = millis() + TIME_WALL_RECHECK;
    }
    else
    {
        if(millis()<timeAntiRecheck) return NO_WALL;
        if (irFarLeft > irFarRight)
        {
            if(irFarLeft < 200)
            {
                this->setSpeed(-100,+100);
                delay(TIME_WALL_CHECK_TURN);
                this->stop();
                irFarLeft = analogRead(SENSOR_IR_LEFT);
                irFarRight = analogRead(SENSOR_IR_RIGHT);
                delay(50); //wait for IR
                if (irFarLeft > 100 && irFarRight > 100)
                {
                    this->setSpeed(+100,-100);
                    delay(TIME_WALL_CHECK_TURN+500);
                    //Serial.println("WALL LEFT");
                    return WALL_LEFT; // wall detected on the left
                }
                else
                {
                    this->setSpeed(+100,-100);
                    delay(TIME_WALL_CHECK_TURN);
                    timeAntiRecheck = millis() + TIME_WALL_RECHECK;
                    return NO_WALL;
                }
            }
            else
            {
                //incease the robot wants to turn on itself, but an obstacle is too near
                // and would destroy the deployed cage
                this->setSpeed(-100,-100);
                delay(1000);
            }
        }
        else
        {
            if(irFarRight < 200)
            {
                this->setSpeed(+100,-100);
                delay(TIME_WALL_CHECK_TURN);
                this->stop();
                irFarLeft = analogRead(SENSOR_IR_LEFT);
                irFarRight = analogRead(SENSOR_IR_RIGHT);
                delay(50); //wait for IR
                if (irFarLeft > 100 && irFarRight > 100)
                {
```c
{ 
  this->setSpeed(-100,+100);
  delay(TIME_WALL_CHECK_TURN+500);
  return WALL_RIGHT; // wall detected on the left
} else 
{ 
  this->setSpeed(-100,+100);
  delay(TIME_WALL_CHECK_TURN);
  timeAntiRecheck = millis() + TIME_WALL_RECHECK;
  return NO_WALL;
} 
}
else 
{ 
  //incase the robot wants to turn on itself , but an obstacle is too near and 
  //would destroy the deployed cage 
  this->setSpeed(-100,-100);
  delay(1000);
}
}

void Tokamak::gotoHeading(int16_t direction_home) 
{
  int16_t currentHeading = this->getHeading();
  int16_t error;
  error = currentHeading - direction_home;
  // in order to avoid the -180 -> 180 jump
  if(error > 180){
    error = error - 360 ;
  } else if(error < -180){
    error = error + 360 ;
  }
  //reduce error so that the navigation function can use it
  if(error>100) error = 100;
  else if(error<-100) error = -100;
  this->navigation_avoidance(-error);
}

void Tokamak::navigation_avoidance(int16_t error)
{
  uint16_t irLeft, irRight, irFarLeft, irFarRight;
  int8_t speedL,speedR;
  static int8_t countL,countR;
  irFarLeft = analogRead(SENSOR_IR_LEFT);
  irFarRight = analogRead(SENSOR_IR_RIGHT);
  irLeft = analogRead(SENSOR_IR_CENTER_LEFT);
  irRight = analogRead(SENSOR_IR_CENTER_RIGHT);
  static uint32_t timeAntiLoop = 0;
  if(millis() > timeAntiLoop)
  {
```
// every 3 seconds reset the anti-loop timer and wheel counters
countL = 0;
countR = 0;
timeAntiLoop = millis() + TIME_ANTI_LOOP;
}
if(countL > 3 && countR > 3) // this condition is to not get stuck
{
    if (this->flags.cagePosition == CAGE_DOWN) // reverse
        { 
            speedL = -CONST_SPEED_OBSTACLE;
speedR = -CONST_SPEED_OBSTACLE;
            this->setSpeed(speedL, speedR);
delay(500);
        }
    if(error > 0)
        {
            speedL = CONST_SPEED_OBSTACLE;
speedR = -CONST_SPEED_OBSTACLE;
        }
    else
        {
            speedL = -CONST_SPEED_OBSTACLE;
speedR = CONST_SPEED_OBSTACLE;
        }
    // set speed immediately
    this->setSpeed(speedL, speedR);
    // turn for 1 second
delay(1000);
    // reset the counters
    countL = 0;
countR = 0;
}
// avoid obstacles using the IR sensors
else if(irRight > CONST_IR_OBSTACLE_CENTER_CAGE)
{
    // turn on itself
    speedL = CONST_SPEED_OBSTACLE;
speedR = -CONST_SPEED_OBSTACLE;
countL++;
}
else if(irLeft > CONST_IR_OBSTACLE_CENTER_CAGE)
{
    speedL = -CONST_SPEED_OBSTACLE;
speedR = CONST_SPEED_OBSTACLE;
countR++;
}
else if(irFarRight > CONST_IR_OBSTACLE_SIDE_CAGE)
{
    // block one wheel, reverse the other
    speedL = -CONST_SPEED_OBSTACLE;
speedR = 0;
countL++;
}
else if(irFarLeft > CONST_IR_OBSTACLE_SIDE_CAGE)
{
    speedL = 0;
speedR = -CONST_SPEED_OBSTACLE;
countR++;
}
else
{

if ( abs(error)<5 )
{
    // under an error of 5 units go straight
    speedL = CONST_SPEED_MAX;
    speedR = CONST_SPEED_MAX;
}
else
{
    if(error>0)
    {
        if(error == 100 && irFarRight > 100)
        {
            speedL = CONST_SPEED_OBSTACLE;
            speedR = -CONST_SPEED_OBSTACLE;
        }
        else
        {
            speedL = CONST_SPEED_OBSTACLE;
            speedR = (100 - abs(error)); //* CONST_SPEED_OBSTACLE / 100 );
        }
        //Serial.write("turn right\t");
    }
    else
    {
        if(abs(error) == 100 && irFarLeft > 100)
        {
            speedL = -CONST_SPEED_OBSTACLE;
            speedR = CONST_SPEED_OBSTACLE;
        }
        else
        {
            speedL = (100-abs(error)); //* CONST_SPEED_OBSTACLE / 100 );
            speedR = CONST_SPEED_OBSTACLE;
        }
        //Serial.write("turn left\t");
    }
    this->setSpeed(speedL, speedR);
}

Listing 5: Robot class

/*******************************************************/

* Title: STI competition Arduino code for the Tokamak robot
* Date: 2014-06-06
*
*******************************************************/

#ifndef _PRISMINO_MASTER
#define _PRISMINO_MASTER

// comment this line when testing without the motor controller
#define ENABLE_CONTROLLER

// comment this line to disable all bluetooth functionalities
#define ENABLE_BLUETOOTH

// comment this line to turn off sounds
#define ENABLE_SOUND

// comment this line to disable compass

K. Kangur, M. Starein, C. Xie 73
#define ENABLE_COMPASS
#define Bluetooth Serial1
#define I2C_MOTOR_CONTROLLER_ADDRESS 0x04
#define I2C_COMPASS_ADDRESS 0x1E
#define PIN_LIGHTS 9
#define SENSOR_IR_LEFT 0
#define SENSOR_IR_CENTER_LEFT 1
#define SENSOR_IR_CENTER_RIGHT 2
#define SENSOR_IR_RIGHT 3
#define SERVO_LEFT_UP 10
#define SERVO_LEFT_DOWN 170
#define SERVO_RIGHT_UP SERVO_LEFT_DOWN
#define SERVO_RIGHT_DOWN SERVO_LEFT_UP
#define TONE_I2C_ERROR 3300
#define TONE_BEEP 440
#define TONE_BATTERY 1100
#define TONE_CHECK_BATTERY 1000
#define TIME_ANTI_LOOP 3000
#define TIME_ANTI_LOOP_TIMEOUT_TURN 1000
#define TIME_BOTTLE_SEEN_TIMEOUT 2000
#define TIME_BOTTLE_RECHECK 4000
#define TIME_CHECK_BATTERY 1000
#define TIME_ANTI_LOOP_TIMEOUT_TURN 1000
#define TIME_WALL_RECHECK 4000
#define TIME_TURN_FOLLOW_WALL 1500
#define TIME_TURN_AROUND 1000
#define TIME_WALL_CHECK_TURN 700
#define TIME_GRAB_LOST_BOTTLE 500
#define CONST_SPEED_MAX 100
#define CONST_SPEED_OBSTACLE 100
#define CONST_SPEED_BOTTLE 80
#define CONST_SPEED_SET_DELAY 50
#define CONST_BOTTLE_SIZE_LOWER_CAGE 90
#define CONST_DEVIATION_OK 15
#define CONST_DEVIATION_OK_STRICT 5
#define CONST_LOOP_TURN_TIMES 3
// in milliseconds, the pause time between 1 degree change
#define CONST_MAX_SERVO_SPEED 10
// calibrated value, about 7.1V
#define CONST_BATTERY_LOW 470

// IR sensor value under which an obstacle is detected
#define CONST_IR_OBSTACLE_SIDE 300
#define CONST_IR_OBSTACLE_CENTER 310

// IR sensor values to use when the cage is deployed
#define CONST_IR_OBSTACLE_SIDE_CAGE 160
#define CONST_IR_OBSTACLE_CENTER_CAGE 140

// value at which there an obstacle far away
#define CONST_IR_NO_OBSTACLE 100
#define CONST_IR_TOO_CLOSE 250

// when following a wall the sensor should remain between these values
#define CONST_IR_WALL_FOLLOW_MIN 160
#define CONST_IR_WALL_FOLLOW_MAX 200

#define CONST_IR_OBSTACLE_FAR 100

// direction towards which to go when going home
#define CONST_HEADING_HOME 50

enum comm_methods
{
    USB,
    BLUETOOTH
};

enum cage_positions
{
    CAGE_UP,
    CAGE_DOWN
};

enum state_codes
{
    STATE_SEARCHING,
    STATE_FETCHING_BOTTLE,
    STATE_LOWER_CAGE,
    STATE_GO_HOME,
    STATE_RAISE_CAGE
};

enum return_codes
{
    OK,
    FAIL,
    REPEAT
};

enum sounds
{
    COIN,
    POWERUP,
    ONEUP,
    FLAGPOLE
};

enum sides
C  SOURCE CODE

Listing 6: Robot header

```c
enum state_wall
{
    NO_WALL,
    WALL_LEFT,
    WALL_RIGHT
};

#define ENTRY_STATE STATE_SEARCHING

enum state_wall
{
    NO_WALL,
    WALL_LEFT,
    WALL_RIGHT
};

#ifdef ENTRY_STATE
#define ENTRY_STATE
#endif

class Tokamak
{
public:
    struct Flags
    {
        uint8_t enableFrontLeds :1;
        uint8_t cagePosition :1;
        uint8_t running: 1;
        uint8_t wall: 2;
    };

    Servo servoLeft;
    Servo servoRight;
    uint16_t batteryVoltage;
    uint16_t currentLeft;
    uint16_t currentRight;

    char readInput(comm_methods);
    void sendOutput(comm_methods, const char*);

    Flags flags;

    Tokamak();
    void setCagePosition(cage_positions);
    void setSpeed(int8_t, int8_t);
    void turn(int16_t, int8_t);
    void stop(void);
    void setLights(boolean);
    void checkBattery(void);
    int16_t getHeading();
    // default speed is the maximum
    void headTo(int16_t, int8_t = CONST_SPEED_MAX);
    void readIrSensors(uint16_t*, uint16_t*, uint16_t*, uint16_t*);
    void playSound(sounds);

    void followWall(uint16_t*, sides, int8_t = CONST_SPEED_MAX);
    void navigation_avoidance(int16_t);
    void gotoHeading(int16_t direction_home = CONST_HEADING_HOME);
    void resetReturnBase(void);
    return_codes goHome(int16_t = CONST_HEADING_HOME);
    uint8_t detectWall(void);
};

#endif
```

```c
#ifndef _sound_h
#define _sound_h

K. Kangur, M. Starein, C. Xie
```
// define a note with a pitch and a duration
struct note
{
  uint16_t pitch;
  uint16_t duration;
};

// define a sound that has a length (number of notes) and a list of notes
struct sound
{
  uint8_t length;
  note *notes;
};

#endif

Listing 7: Sound header

// Frequencies for equal-tempered scale, A4 = 440Hz
// Reference: http://www.phy.mtu.edu/~suits/notefreqs.html
#ifndef _pitch_h
#define _pitch_h

// Scientific name
#define C0 16
#define C0s 17
#define D0b 17
#define D0 18
#define D0s 19
#define E0b 19
#define E0 21
#define F0 22
#define F0s 23
#define G0b 23
#define G0 25
#define G0s 26
#define A0b 26
#define A0 28
#define A0s 29
#define B0b 29
#define B0 31
#define B0s 32
#define C1 33
#define C1s 35
#define D1b 35
#define D1 37
#define D1s 39
#define E1b 39
#define E1 41
#define F1 44
#define F1s 46
#define G1b 46
#define G1 49
#define G1s 52
#define A1b 52
#define A1 55
#define A1s 58
#define B1b 58
#define B1 62
#define C2 65
#define C2s 69
#define D2b 69

K. Kangur, M. Starein, C. Xie
#define D2 73
#define D2s 78
#define E2b 78
#define E2 82
#define F2 87
#define F2s 93
#define G2b 93
#define G2 98
#define G2s 104
#define A2b 104
#define A2 110
#define A2s 117
#define B2b 117
#define B2 123
#define C3 131
#define C3s 139
#define D3b 139
#define D3 147
#define D3s 156
#define E3b 156
#define E3 165
#define F3 175
#define F3s 185
#define G3b 185
#define G3 196
#define G3s 208
#define A3b 208
#define A3 220
#define A3s 233
#define B3b 233
#define B3 247
#define C4 262
#define C4s 277
#define D4 294
#define D4s 311
#define E4b 311
#define E4 330
#define F4 349
#define F4s 370
#define G4b 370
#define G4 392
#define G4s 415
#define A4b 415
#define A4 440
#define A4s 466
#define B4b 466
#define B4 494
#define C5 523
#define C5s 554
#define D5b 554
#define D5 587
#define D5s 622
#define E5b 622
#define E5 659
#define F5 698
#define F5s 740
#define G5b 740
#define G5 784
#define G5s 831
#define A5b 831
#define A5 880
```c
#define A5s   932
#define B5b   932
#define B5    988
#define C6     1047
#define C6s    1109
#define D6b    1109
#define D6     1175
#define D6s    1245
#define E6     1245
#define E6b    1319
#define F6     1397
#define F6s    1480
#define G6b    1480
#define G6     1568
#define G6s    1661
#define A6b    1661
#define A6     1760
#define A6s    1865
#define B6b    1865
#define B6     1976
#define C7     2093
#define C7s    2217
#define D7b    2217
#define D7     2349
#define D7s    2489
#define E7     2489
#define E7b    2637
#define F7     2794
#define F7s    2960
#define G7b    2960
#define G7     3136
#define G7s    3322
#define A7b    3322
#define A7     3520
#define A7s    3729
#define B7b    3729
#define B7     3951
#define C8     4186
#define C8s    4435
#define D8b    4435
#define D8     4699
#define D8s    4978

// Piano key number
#define P01    28
#define P02    29
#define P03    31
#define P04    33
#define P05    35
#define P06    37
#define P07    39
#define P08    41
#define P09    44
#define P10    46
#define P11    49
#define P12    52
#define P13    55
#define P14    58
#define P15    62
#define P16    65
#define P17    69
#define P18    73
```
```c
#define P19 78
#define P20 82
#define P21 87
#define P22 93
#define P23 98
#define P24 104
#define P25 110
#define P26 117
#define P27 123
#define P28 131
#define P29 139
#define P30 147
#define P31 156
#define P32 165
#define P33 175
#define P34 185
#define P35 196
#define P36 208
#define P37 220
#define P38 233
#define P39 247
#define P40 262
#define P41 277
#define P42 294
#define P43 311
#define P44 330
#define P45 349
#define P46 370
#define P47 392
#define P48 415
#define P49 440
#define P50 466
#define P51 494
#define P52 523
#define P53 554
#define P54 587
#define P55 622
#define P56 659
#define P57 698
#define P58 740
#define P59 784
#define P60 831
#define P61 880
#define P62 932
#define P63 988
#define P64 1047
#define P65 1109
#define P66 1175
#define P67 1245
#define P68 1319
#define P69 1397
#define P70 1480
#define P71 1568
#define P72 1661
#define P73 1760
#define P74 1865
#define P75 1976
#define P76 2093
#define P77 2217
#define P78 2349
#define P79 2489
#define P80 2637
```
C.3 Motor controller source code

```c
#include <wildthumper.h>
#include <Wire.h>
#define I2C_ADDRESS 4
#define TIME_CHECK_BATTERY 500
#define CONST_BATTERY_LEVEL_GOOD 470
#define PIN_BACK_LIGHT 12
#define TIME_BACK_LIGHT_BLINK 50
#define TIME_BACK_LIGHT_PAUSE 500
// must be an even number
#define CONST_BACK_LIGHT_BLINK_TIMES 6

uint32_t backLightNextBlink = 0;
uint32_t backLightBlinkTimes = CONST_BACK_LIGHT_BLINK_TIMES;
WildThumper controller;

void setup()
{
  pinMode(LED, OUTPUT);
  digitalWrite(LED, HIGH);
  pinMode(PIN_BACK_LIGHT, OUTPUT);
  Wire.begin(I2C_ADDRESS);
  // disable internal pull-ups
  digitalWrite(SDA, LOW);
  digitalWrite(SCL, LOW);
  Wire.onReceive(receiveEvent);
```
wire.onRequest(requestEvent);
}

void loop()
{
    // check the battery every now and then
    if(millis() > timeCheckBattery + TIME_CHECK_BATTERY)
    {
        // show the battery state on the on-board LED
        if(controller.battery() < CONST_BATTERY_LEVEL_GOOD)
        {
            batteryGood = 0;
            digitalWrite(LED, LOW);
        }
        timeCheckBattery = millis() + TIME_CHECK_BATTERY;
    }
    if(millis() > backLightNextBlink && batteryGood)
    {
        if(backLightBlinkTimes)
        {
            backLightBlinkTimes --;
            digitalWrite(PIN_BACK_LIGHT, !digitalRead(PIN_BACK_LIGHT));
            backLightNextBlink = millis() + TIME_BACK_LIGHT_BLINK;
        }
        else
        {
            backLightBlinkTimes = CONST_BACK_LIGHT_BLINK_TIMES;
            backLightNextBlink = millis() + TIME_BACK_LIGHT_PAUSE;
        }
    }
    else if(millis() > backLightNextBlink && !batteryGood)
    {
        // if the battery has gone under the threshold level blink continuously
        digitalWrite(PIN_BACK_LIGHT, !digitalRead(PIN_BACK_LIGHT));
        backLightNextBlink = millis() + TIME_BACK_LIGHT_BLINK;
    }
}
C SOURCE CODE

Listing 9: Motor controller source code

K. Kangur, M. Starein, C. Xie
PORTD &= ~(1 << 6) | (1 << 5) | (1 << 3);

// do not allow higher or lower values than 100 or -100
if (_speedLeft < 0)
{
    speedLeft = _speedLeft < -100 ? -100 : _speedLeft;
} else
{
    speedLeft = _speedLeft > 100 ? 100 : _speedLeft;
}

if (_speedRight < 0)
{
    speedRight = _speedRight < -100 ? -100 : _speedRight;
} else
{
    speedRight = _speedRight > 100 ? 100 : _speedRight;
}

// set compare interrupt
uint16_t temp;

// PWM compare value between 0 and 255
temp = (long) 255 * (speedLeft > 0 ? speedLeft : -speedLeft) / 100;

OCR2A = temp & 0xff;

temp = (long) 255 * (speedRight > 0 ? speedRight : -speedRight) / 100;

OCR2B = temp & 0xff;

// reset timer
TCNT2 = 0;

// set prescaler to 64 (enable timer), do not set a lower prescaler
// it won’t work because of hardware restrictions (power transistors do not
// commute fast enough)
// this gives 16MHz/256/64 = 976.5625Hz, documentation says maximum frequency is
// 24kHz
TCCR2B |= (1 << CS22);

// interrupt vectors for pin toggling
ISR(TIMER2_COMPA_vect)
{
    if (speedLeft > 0)
    {
        PORTD &= ~(1 << 3);
    } else if (speedLeft < 0)
    {
        PORTB &= ~(1 << 3);
    }
}

ISR(TIMER2_COMPB_vect)
{
    if (speedRight > 0)
    {
        PORTD &= ~(1 << 5);
    }
```c
ISR(TIMER2_OVF_vect)
{
    if(speedLeft > 0)
    {
        PORTD |= (1 << 3);
    } else if(speedLeft < 0)
    {
        PORTB |= (1 << 3);
    }
    if(speedRight > 0)
    {
        PORTD |= (1 << 5);
    } else if(speedRight < 0)
    {
        PORTD |= (1 << 6);
    }
}

uint16_t WildThumper::battery(void)
{
    return analogRead(PIN_BATTERY);
}

uint8_t WildThumper::currentLeft(void)
{
    return analogRead(PIN_CURRENTL);
}

uint8_t WildThumper::currentRight(void)
{
    return analogRead(PIN_CURRENTR);
}

uint8_t WildThumper::currentTotal(void)
{
    return this->currentLeft() + this->currentRight();
}
```

Listing 10: Motor controller library
CSOURCE CODE

Listing 11: Motor controller library header