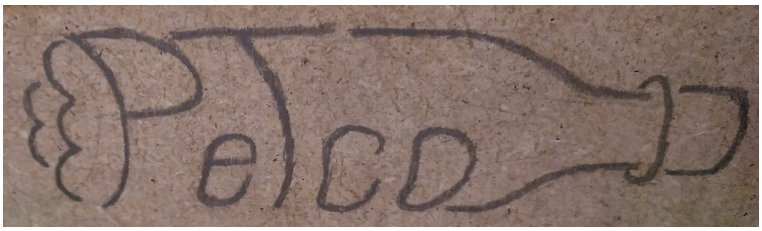


STI Interdisciplinary Robot Competition

Report of team PeTco (Group 4)

Auteurs:
Cédric Athanasiadès
Alexis Schizas
Devakumar Thammisetty

Project Supervisor:
Alessandro Crespi

Team coach:
Özdemir Can Kara

Spring semester 2019

EPFL

June 16, 2019

Contents

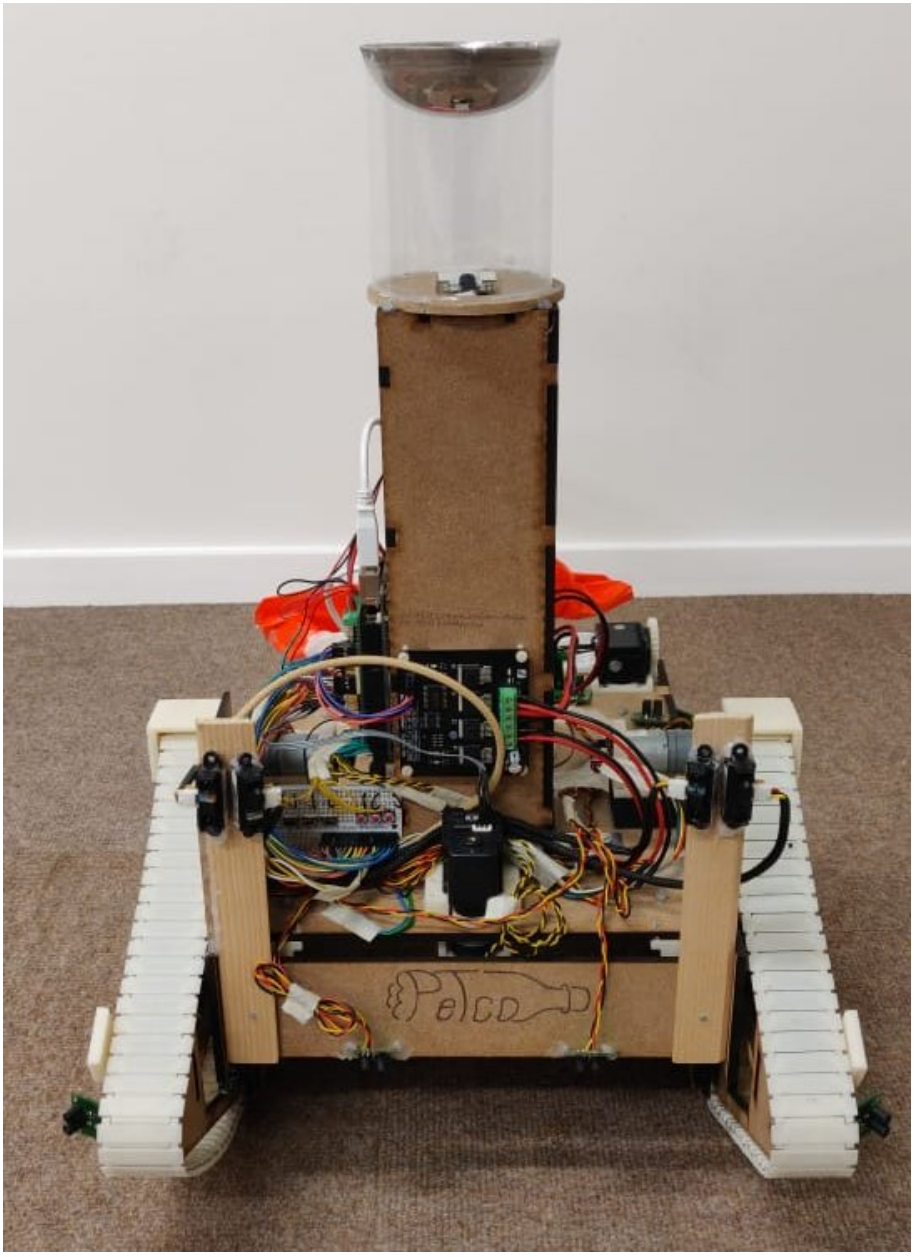


Figure 1: Our beautiful robot

1 Introduction

The goal of this project was to design an autonomous robot that picks up and brings back pet bottles placed at random on an arena in a defined lapse time. The arena is presented below on figure 2.
This was done over the course of a semester, by a team of three students from different master sections. Our team was composed of :

Alexis Schizas, bachelor in micro-engineering and in the first year of the robotics master, worked mainly on drawing the CAD files for the robot and manufacturing it.
Cédric Athanasiadès, bachelor in micro-engineering and in the first year of the robotics master. Worked mostly on the electronics, Arduino code and the building of the robot.
Devakumar Thammisetty, bachelor in Aerospace Engineering from Indian Institute of Technology, Bombay and presently in the first year of Mechanical Engineering master. Worked with Raspberry Pi, mostly in the software section including bottle detection localization and Navigation.

Figure 2: The full mounted arena

2 Rules & objectives

The goal of the competition is to win a maximum of points during a time lapse of ten minutes. For this, the robot must capture bottles and retrieve them in the *Home* zone. Given the zones where the bottles are, the values can change as it is more difficult to reach such areas. The zone delimited by rocks and the elevated platform contain each 6 bottles that are worth 40 points each. The grass zone contains also 6 bottles and they are worth 20 points and finally the remaining area has 15 bottles at 10 points. In order to have the maximum of points the bottles must be retrieved in the yellow zone. If they are released in the green square, the points are divided by 2. If at the end of the time the robot contains bottles, they are worth a quarter of there value (recapitulation

of figure 3).
The robot must be able to detect bottles and avoid obstacles (walls as well as bricks and eventually rocks).
There is no maximal weight but the dimensions of the robot must be under a square meter.

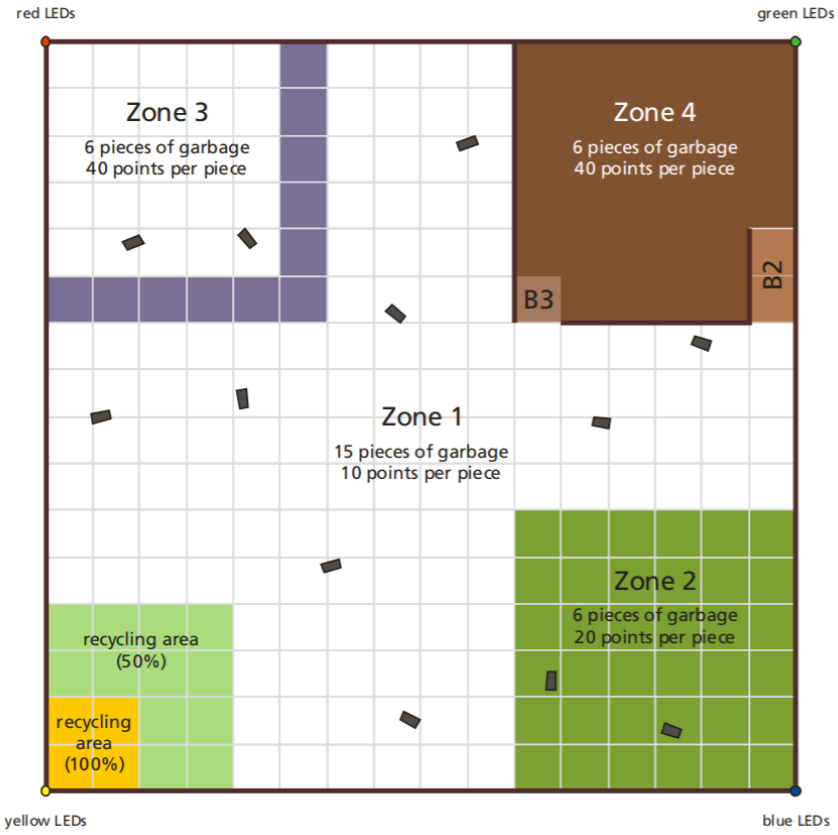


Figure 3: The arena schematics

3 Concepts

During the first month, we review a lot of designs based on very different approaches and tried to define the pros and cons of each ideas. When an idea was promising, we went into more details to check the viability. We decided early on that our robot should take more than one bottle as we figured that the speed would not be fast enough in between obstacles to retrieve many bottles if it had to go back and forth to the recycling area.
After discussing on all our alternatives, we had only two concepts remaining.

One of them was the one implemented and the other can be seen in the drawing below (figure 4).

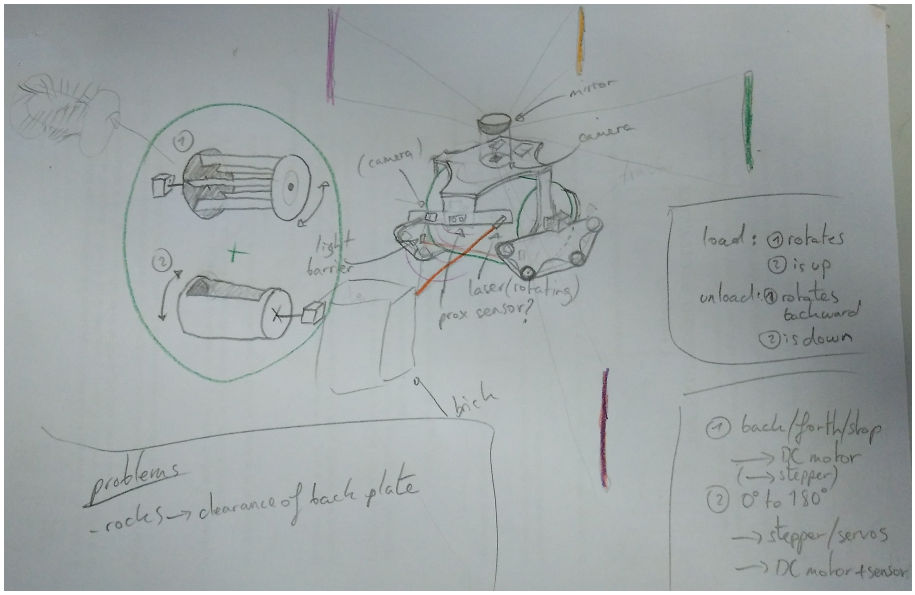


Figure 4: One of the last design we hesitated to build.

In this design the robot would have two cylinders, one inside the other. The external one would be continuously rolling to catch bottles and the second contains the bottles. At the end, the robot would change the position of the inside cylinder and turn the external one in the opposite direction in order to empty them. Although the concept seemed good, there were some issues as how to make sure the bottles are captured and which zones would be accessible as the clearance was very low and the cylinder quite big.
We also had one with 4 clamps on each side of the robot that could pick up to four bottles, but the design had a lot of flows and was much more complicated to build.

4 Different solutions examined

In order to have an objective view on our choices, we separated the design in a few categories where we explored our options and compared the pros and cons.

4.1 Locomotion

4.1.1 Wheels

Advantages

- Simple to use, reliable and they offer low friction.
- Possibility to have high speed.

Disadvantages

- Hard to adjust friction if it is too high/low.
- We can't manufacture the rubber ourselves. We have to find and buy something that has the desired shape.
- The diameter defines the size of the obstacles we can pass.

If wheels were to be chosen, the question of how many wheels and which one needs to be directional is a difficult choice as most combinations have advantages and inconvenient. For example, having four wheels with two directional is very stable but more complex to drive and if the four are directional there can be a lot of friction for turning if differential speeds are used. In our robot the best choice would probably be to have only two wheels as it is the most simple to control and it allows the robot to turn easily on itself without too much friction. The disadvantage is that it requires to have other support in front and back to ensure stability and then it becomes more difficult to pass over rocks or climb the platform.

4.1.2 Caterpillar tracks

Advantages

- Sturdy, better on rough terrain (rocks).
- Easy to adjust friction by changing/adding something on the track surface.

Disadvantages

- Long and expensive to manufacture by 3D printing.
- Does not allow very high speed.

4.1.3 Swedish wheels

Advantages

- Allows the robot to translate horizontally, increases mobility.
- Probably allows good speeds as well.

Disadvantages

- Difficult to manufacture ourselves, not a huge choice of models available for purchase.
- Probably bad on rough terrain (rocks).
- Relatively expensive compared to standard wheels

4.2 Bottle capture

4.2.1 Grabbing bottles (pincers or other)

Advantages

- Possibility to have a large container on top of the robot (easy to empty, large capacity).
- No issues with crossing the rocks, going on grass or up the ramp.

Disadvantages

- Very reliant on good vision/detection.
- Complex, many active components.

4.2.2 Any form of rolling brush/wheel in front of/under the robot

Advantages

- Simple to control since you can just let it roll, relatively large room for errors, reliable.
- Can be implemented with a large container, allowing for the capture of multiple bottles.

Disadvantages

- Needs to be close to the ground, could pose problems for the rocks.
- The whole robot needs to be designed around this feature.

4.2.3 Capturing the bottles underneath the robot, rolling against the ground

Advantages

- Probably the easiest solution there is.
- Very easy to release the bottles.

Disadvantages

- May cause problems when going up the ramp/rolling on grass, probably impossible to cross the rocks.

4.3 Localization

A couple different options were reviewed to localize ourselves in the arena.
The easiest one to implement is to use the beacons as it require only a camera and it gives exact position in all the arena. This option is declined into two possibilities, the first is to see every beacon at once with a mirror that reflects at 360 degrees. The second is to follow one of the beacon but this method is vulnerable as it has only one reference. Furthermore, it it more difficult to estimate the distance from the light and the robot needs to know its orientation.
The other solutions were based on mapping with ether IR sensors or a small 2D LIDAR range scanner. This would have required too much work as we don't have much experience on mapping and it is a very complex task.
We decided to use a spherical mirror as it had the main advantage that the bottle detection could be done at the same time. Other mirror shape could have been used for even better results, but it would have required more complex calculation and to make it by CNC turning.

4.4 Obstacle avoidance

To detect the brick or walls, there is not a lot of options as they don't have particular property and it needs to be detected at long range. The main sensors evaluated were : SHARP geometric IR sensors, reflection IR sensors, ultrasonic sensors and vision based on a Raspberry Pi camera.
The use of ultrasonic sensors was very unreliable when the obstacles was not exactly perpendicular to the emitter and so was discarded. As we already used a camera for localization we wanted to avoid to need a second one as it makes the communication more complex and so creates more risks. We decided then that we would use IR sensors.

4.5 Bottle detection

At first we wanted to use a second camera placed in front of the robot to detect bottles but as the spherical mirror idea was getting concreted, we found it more and more attractive to use the view for the bottles as well and it simplified a lot the electronics and processing as we only needed one Raspberry Pi.
A backup solution was to find bottles relying only one IR or ultrasonic sensors but after trying this option in the early stage, it proved to be very unreliable.

5 Chosen solutions description

5.1 Locomotion : tracks

We decided to go with caterpillar tracks because we wanted our robot to be able to cross the rocks. It also had the advantage of being able to rotate on itself pretty easily. While it was certainly possible to do all this with wheels as well, we felt like caterpillar tracks weren't a bad solution either, and there

was some challenge involved in actually coming up with the track design and manufacturing them, instead of just buying wheels on the internet.

5.2 Bottle capture & release

For our bottle capture mechanism, we liked the idea of having the bottles being captured under the robot, but we wanted to be able to cross the rocks. Instead of going for a continuously rotating brush type of mechanism, we decided to have a guided metal rod underneath the robot that could push bottles in a bag. This rod has two positions, one that is close enough to the ground to be able to push the bottles inside the bag, and another one that is high enough so that it doesn't touch the rocks when crossing them.
The bag in which the bottles are pushed is fixed to a MDF door frame. This frame is also actuated, and can be moved inside the robot to completely close the bag and avoid touching the rocks. It can also be tilted backwards 90 degrees, so that the bottles fall out.

6 Robot mechanical design

6.1 Tracks

Our caterpillar tracks and wheels were all 3D printed. The material used was ABS. To make the transmission between the wheels and the tracks reliable and as simple as possible, we chose a design where the tracks have a cylindrical base that overlaps with a cylindrical slit in the wheel. We left just enough give to make sure that mounting would be easy, but not too much to not make the whole track too loose. The links were assembled with iron wire.

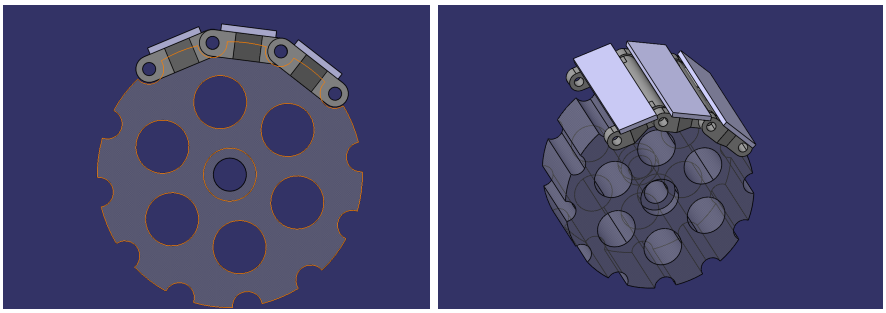


Figure 5: Wheel and track design

6.2 Main structure

For the structure, we almost exclusively used 5mm thick MDF (medium density fiberboard) panels. They are extremely simple to cut, are very cheap and low

weight. The main structure is composed of a top panel on which all our electronics and motors are mounted, and two side panels that constitute our bottle capture mechanism. Mounted on these side panels come our tracks, which are held in place by two other pieces of MDF on each side.

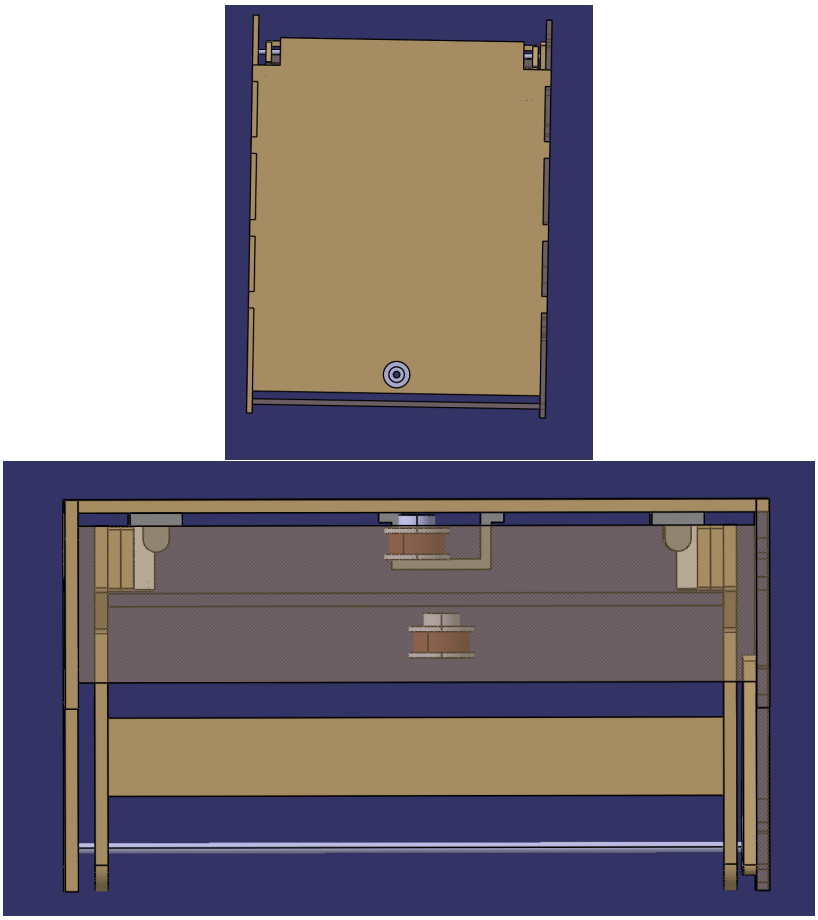


Figure 6: Main structure

The figure above shows the main structure, while the figure below shows the structure on which the tracks and wheels are mounted.

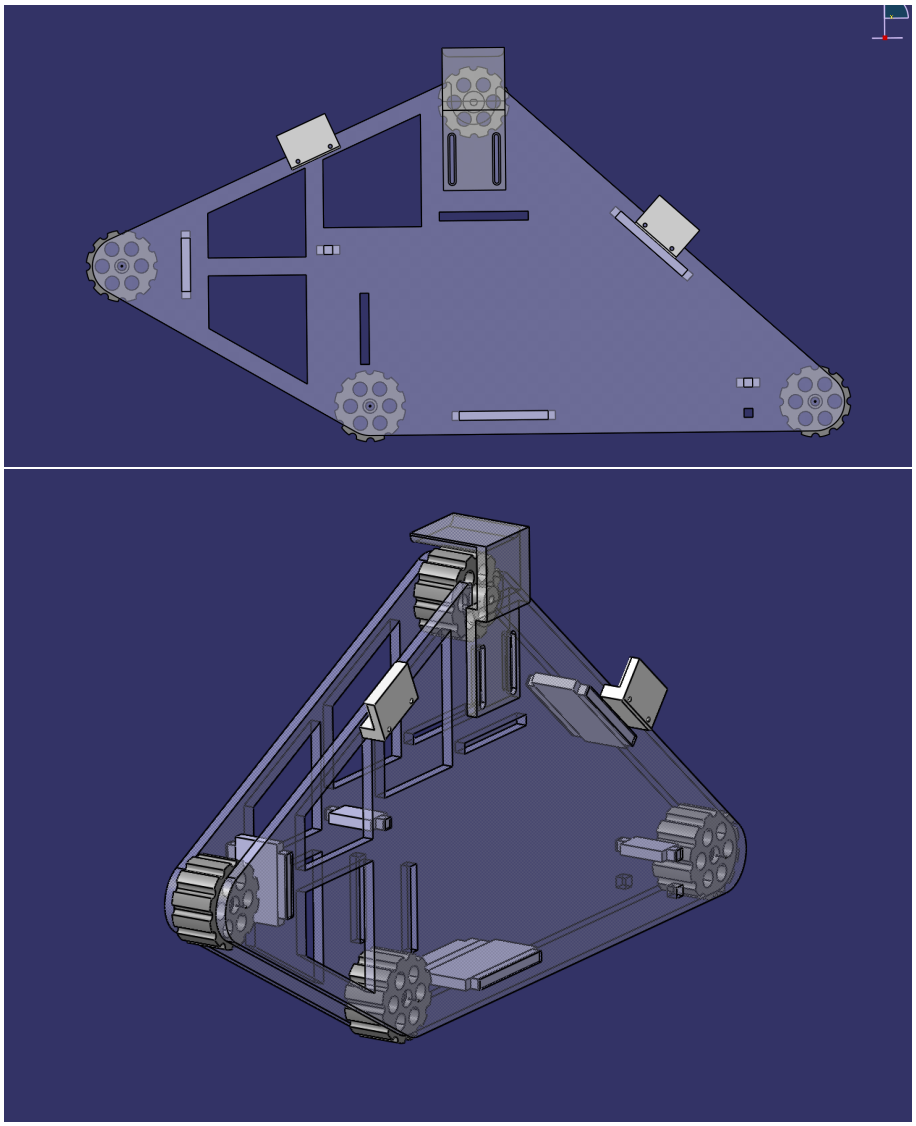


Figure 7: Caption

As you can see on figure 7, three elements were added on top. The purpose of these elements was to prevent the tracks from falling off when the robot is turning on a rough surface with a lot of friction. The central piece was adjustable and allowed us to press the track down against the actuated wheel, to make it impossible to skip a track. The two small pieces on the sides served as guides to avoid the track sliding out. These three pieces were 3D printed and added later, when we realized we needed some extra guidance for the tracks.
The two MDF panels holding the wheels together where held together by screws.

To be sure that there was enough rigidity, we placed a few MDF pieces in between the panels to help them hold together, as can be seen just below.

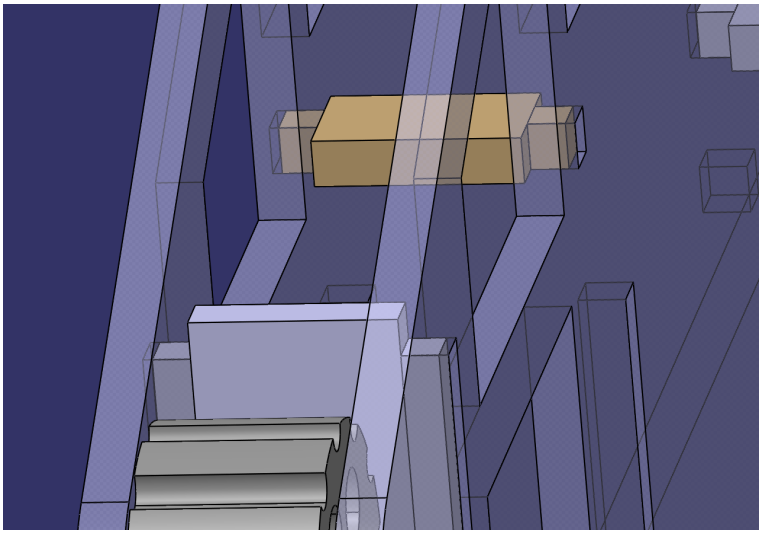


Figure 8: Rigidity elements

6.3 Bottle capture mechanism

Our bottle capture mechanism works by guiding an aluminum rod in a sort of circular motion, allowing it to pass above bottles when needed, and be at about half the height of a bottle when we want to push said bottle in the bag. To guide the rod, we designed two MDF pieces that, when put together, created the path that we wanted the rod to take. We wanted the motion of the rod to be controlled by a single motor, so we added a small pivoting piece that allows the rod to go back up just by changing the direction of the motor when the rod reaches the end of the mechanism. Said piece was 3D printed in ABS to get less friction.

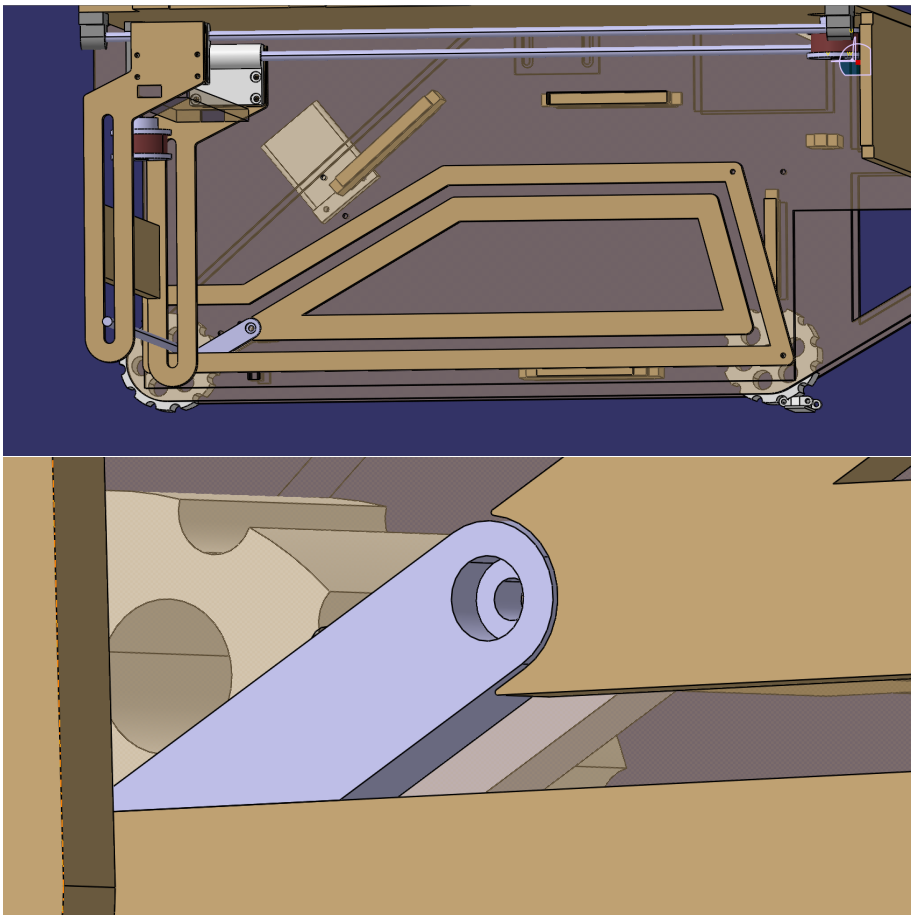


Figure 9: The bottle capture mechanism

6.4 Sensors placement

We have 2 kind of sensors that serve four different purposes. The first one are IR geometric sensors that have a very narrow field of view and the second one are reflective IR sensors that have an angle of view of around 50 degrees. By combining them, we are able to avoid obstacles as well as capture bottles efficiently.

- **Obstacles avoidance in front**
  To differentiate the obstacles from the bottles, the first kind of sensors are useful as they are very directional. Because the walls and bricks are higher than the bottles height even when standing, we placed the sensors between the top of the bricks and the top of a bottle standing. This avoid to detect bottles in front as obstacles.

- **Obstacles avoidance on the sides**
  Three sensors of the second category were places on each side higher than a bottle lying to be able to pass close to bottles. The orientation is done in order to cover the front and back to avoid hitting bricks that are not enough in front to be detected when we turn. And one was also placed on the middle and perpendicular to the robot so that when it is detected, the robot does not turn too much. Those sensors are better for this task as they cover a greater area.

- **Bottles detection**
  Two sensors are placed facing the ground in front of the robot in order to make the robot go straight to a detected bottle.

- **Mechanism trigger**
  As the bottle retrieve mechanism needs to be triggered when a bottle is captured, one reflection sensor is placed just before the entrance of the mechanism. The field of view is restrained in order to be more directional.

During the testing phase we tried several sensors positions before settling on this one. At first more sensors were placed on the sides but it proved not very useful so we replaced them in the front to detect bottles and we kept this solution which seemed to be very efficient during the capture.

6.5 Type and placement of the spherical mirror

During the building phase, we had some trouble finding our spherical mirror. We tried first to use a brilliant Christmas ball but during one of the test, it fell and shattered so we had to find another option. One idea was to use a chromed lamp bulb but the fragility would have been to much a risk. Finally, we found a ladle in metal with a good spherical shape and very reflective (figure 10).
The latest was placed high enough in order to see all beacons from most places on the arena. The height allowed us to see well the surrounding of the robot and detect bottle even close to the robot sides.

Figure 10: The different types of mirror reviewed

6.6 Hardware choices

We choose to use most parts from the catalog as it suited our needs and was usually the cheapest option.
After computing the necessary torque for the servos, the dynamixels were powerful enough and in addition it had a "wheel" mode that was very convenient for our retrieve mechanism. Using only dynamixels for all servos needed, simplified also the programming part. the compatibility with the Arduone was also very practical.
The motors were a critical point of our robot and the friction was one of our main concerns. Due to an error in the catalog we ended up to use LP (Low power) 75:1 motors instead of the HP (high power) ones. We had to do some testing to see if the latest were powerful enough to drive the tracks. With a good controller they proved to be sufficient although we would probably have benefited from a more powerful motor type.

7 Electronics

7.1 Custom IR sensors

The circuit is implemented with an operational amplifier in a transimpedance fashion.
A IR emitting diode (wavelength of 940nm) is wired through a 100 Ohm resistor in order to produce enough light for detection. At first this resistor was designed to be 30O Ohm but this didn't achieve the requirements wanted. A feedback resistor of 1 MOhm was chosen to have a good response range up to 30cm and saturates only below a couple centimeter.
To reduce the power consumption, we could have kept the 300 Ohm for emission and increase the feedback resistor as the noise induced was relatively low.
To reduce the interference due to the direct absorption of light between the diodes, we put around the diodes some thermal gain. This permits us also to limit the angle of view and reduce the cross talk. The light barrier that triggers the bottle retrieve mechanism has a longer piece in order to be very directional but it decrease the response.

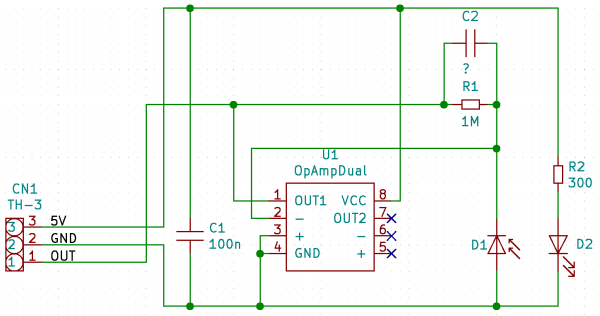


Figure 11: Circuit diagram of the custom IR sensor with emitting diode

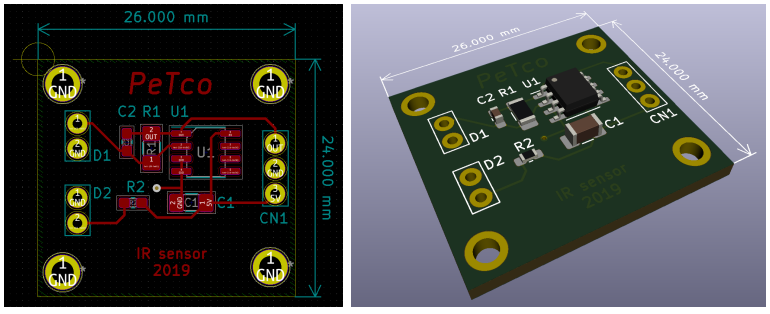


Figure 12: PCB of the custom IR sensor designed with KiCad and its 3D view

Figure 13: PCB of the custom IR sensor once received and soldiered (We can see well the infrared light as the quality of the smartphone camera is poor)

7.2 Main circuit

To avoid the poor connections that come with male breadboard prototyping cables, we decided to use an other board where we could soldier in an easy way the external components and connections needed.

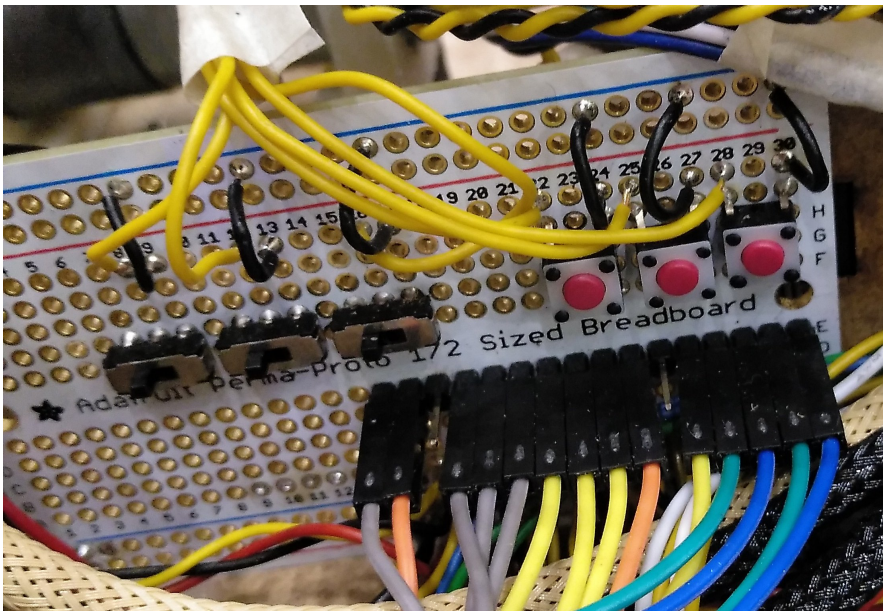
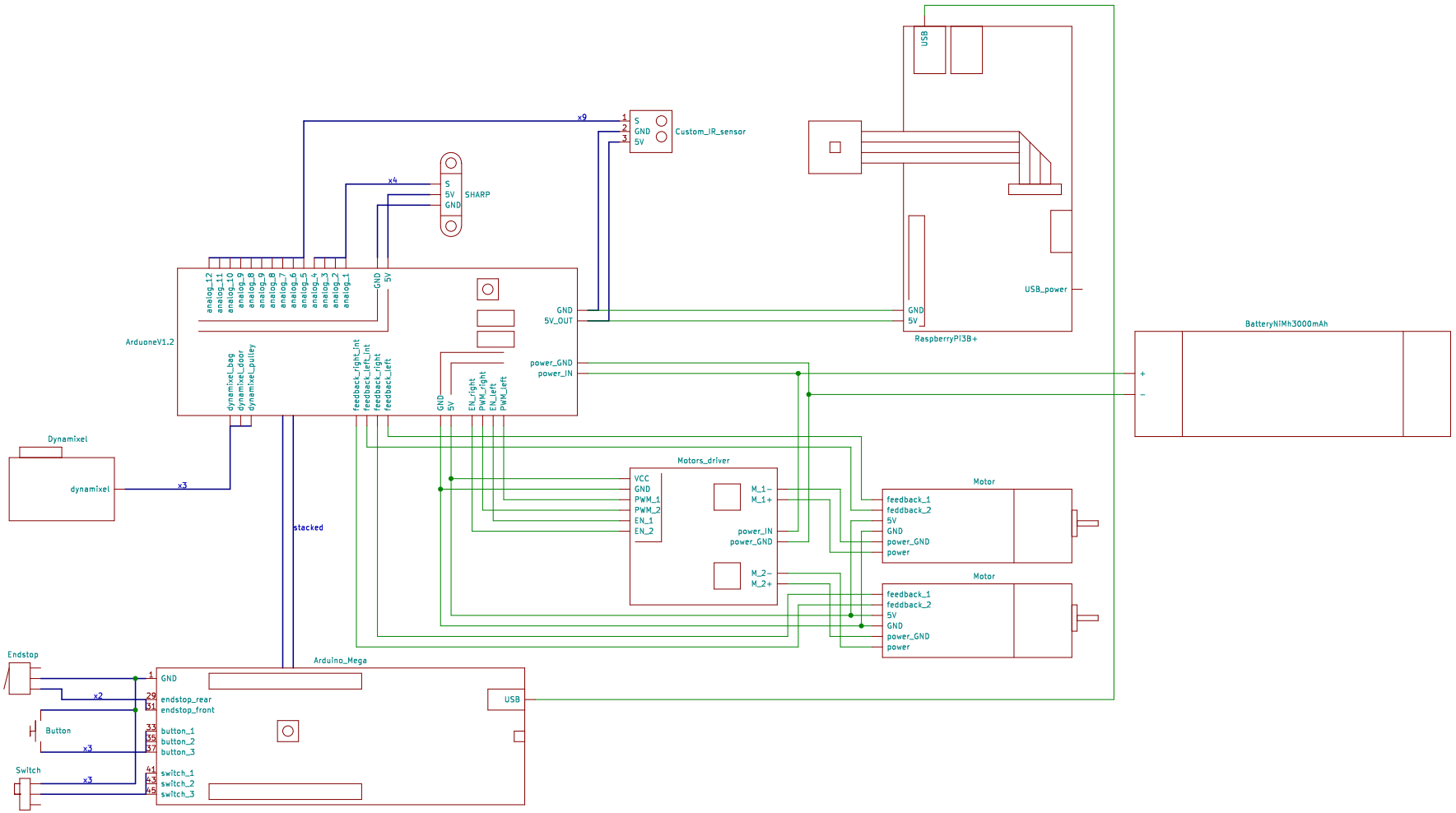


Figure 14: Solder point breadboard with buttons, switches and good male connectors

The lack of the two last analog inputs from the Mega on the Arduone was a problem because we planned to use more sensors a first. This issue was resolved by placing the sensors differently and replacing redundant sensors. We also had

a backup plan to use a multiplexer to increase the inputs number.
(Because of an overlap between the servos communication interrupt and the motors feedback interrupt we had to rewire the motors interrupt on standard interrupt instead of Pin Change Interrupt.)


x9
S
GND
5V
Custom_IR_sensor
x4
S
5V
GND
SHARP
USB
USB_power
GND
5V
RaspberryPi3B+
BatteryNiMh3000mAh
+
−
analog_12
analog_11
analog_10
analog_9
analog_8
analog_7
analog_6
analog_5
analog_4
analog_3
analog_2
analog_1
GND
5V
ArduoneV1.2
GND
5V_OUT
power_GND
power_IN
dynamixel_bag
dynamixel_door
dynamixel_pulley
feedback_right_int
feedback_left_int
feedback_right
feedback_left
GND
5V
EN_right
PWM_right
EN_left
PWM_left
Dynamixel
dynamixel
x3
stacked
Motors_driver
VCC
GND
PWM_1
PWM_2
EN_1
EN_2
M_1−
M_1+
power_IN
power_GND
M_2−
M_2+
Motor
feedback_1
feddback_2
5V
GND
power_GND
power
Motor
feedback_1
feddback_2
5V
GND
power_GND
power
Endstop
x2
Button
x3
Switch
x3
Arduino_Mega
1
GND
29
31
endstop_rear
endstop_front
33
35
37
button_1
button_2
button_3
41
43
45
switch_1
switch_2
switch_3
USB

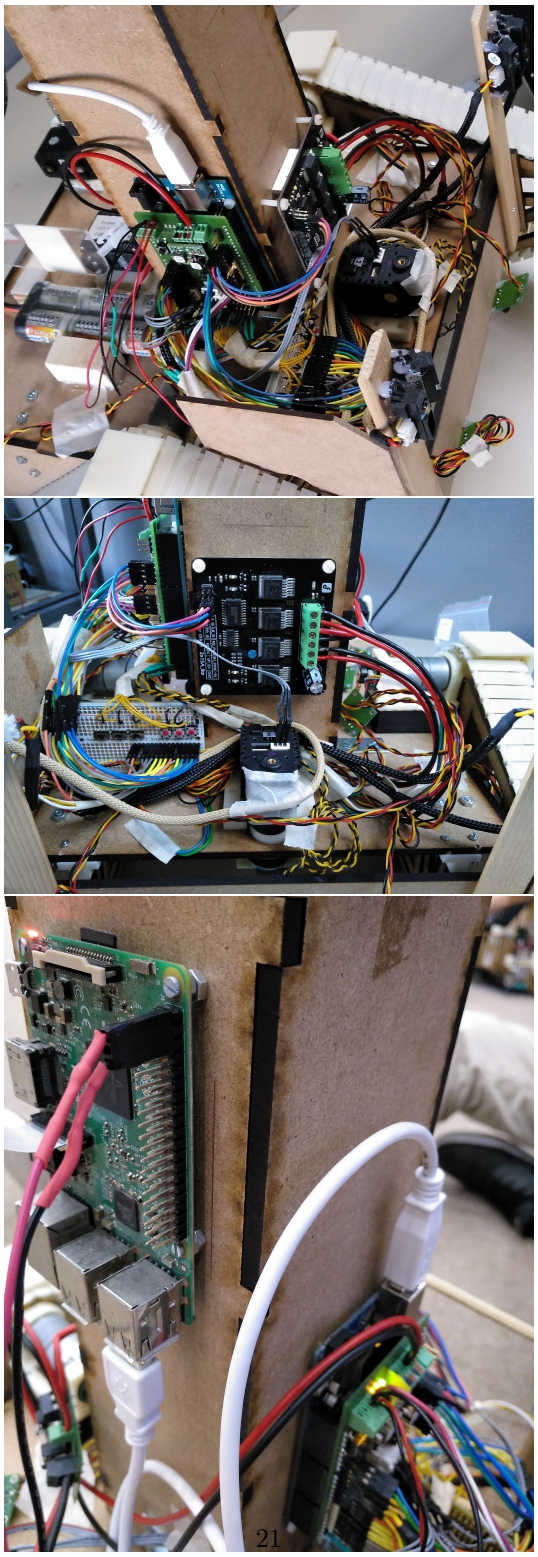


Figure 15: Pictures of the final montage from different points of view

7.3 Power supply

The estimated power consumption of the principal modules is presented in the table below (figure 16).

Component	min [mA]	max [mA]	number	t_{max}/t_{tot}	total [mA]
Raspberry Pi	400	600	1	100%	600
Arduino Mega	80	100	1	100%	100
Custom IR sensors	50	50	12	100%	600
SHARP IR sensors	30	30	4	100%	120
Motors	400	2500	2	30%	2000
Servos	50	1500	3	10%	600

Figure 16: Power consumption with estimated time of use

The total current drawn in idle mode with the motors disabled was around 1,2A and the peak current could go as far as about 5A. The battery capacity being 3000mAh (2850mAh), we can expect the robot to last at least half an hour in the worst conditions. This is sufficient for the 10 minutes of the competition and we have still a decent margin.
However, an issue that we noticed only too late after assembling all the modules together was the battery internal resistance. Because the robot was drawing more current, the voltage of the battery tended to drop and this caused sometimes the Raspberry to reboot although the battery was not yet empty. We tested in real condition during the 10 minutes with a fully charge battery and it never happened.
We could have reduced the idle consumption a lot by multiplexing the sensors power and gained about 0.7A.

8 Software

8.1 Arduino

All low level controls and drivers are implemented on the Arduino side because the feedback loop needs to be fast for better performances. As the processing power is relatively low, the localization, path planning, navigation, etc. are implemented in the Raspberry Pi side. The different parts implemented are presented below in more details.

8.1.1 State machine

For better clarity of the program, a state machine architecture is used. Each state is connected to another with actions that are based on different inputs.

- **Idle**
  This state is the first one after the initialization and is used to wait until

the competition starts. If the push button is triggered, this state is left and the robot goes into the *Move* state.

- **Move**
  In this one the robot reads the proximity sensors and computes the motors speeds and send them to the Raspberry Pi. When it sends back the command speeds, the motors control is updated to match the command. When ether the global timeout is passed or the number of bottles is greater than a fixed value, the state changes to *Return* and a message is sent to the Raspberry Pi. It can also be triggered by the Pi itself instead.

- **Return**
  This state is very similar to the *Move* state and was almost obsolete towards the end. When the Raspberry Pi send a *Home reached* message, the robots goes to *Empty*.

- **Empty**
  Here a list of servos and motors command is applied sequentially in order to empty the bottles from the bag. This is implemented in the same manner as a state machine with timings to pass from one sub-state to another. When all is done, it return to the *Move* state and keeps going forever.

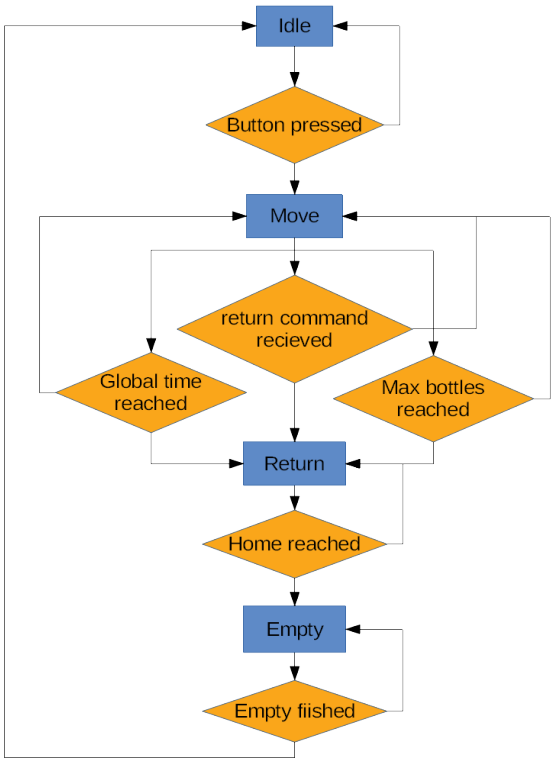


Figure 17: State machine implemented with state in blue rectangles and actions in orange diamonds

8.1.2 Motors control

- **PI controller**
  The proportional controller was not sufficient for our design as the motor internal friction and the tracks friction was relatively high, so we added an integrator term. Although there is some overshoot after the friction diminish, the derivative term is not needed because we don't need to control the speeds precisely for the slow robot speed. The controller was custom made because we wanted to add functionality not implemented in the Arduino PID library. Our controller implements an anti-windup as the integrator term is very important and it tended to saturate the command.
- **Feedback**
  The motor feature a 48 edges per turn in a quadrature fashion, but we only used one of the rising edge to compute the speed. This simplify the code and we did not need that much precision for moving. When the

rising edge triggers the interrupt, we test the speed direction by reading the signal in quadrature and then store the time in microsecond. Once in the main program loop, the speeds are calculated by checking the time elapsed between the last two interruptions. This proved to give a better precision than counting the steps in between two main loop calls.

8.1.3 Sensors

- **Geometric IR sensors (SHARP)**
  This kind of sensors does not have a linear response but one that increase the shorter the distance is. This behaviour is in fact practical because our potential field force norm was the inverse of the squared distance. This permits us to save some calculation as the curve is close on the most part already.

- **Custom Reflection IR sensor**
  As for the previous ones, the response is not proportional and can be directly used in our potential field. As the surface covered is relatively big, there can be cross-talk between the sensors that are close together. To face this problem we hoped to power each sensors in turns with an Arduino IO pin but we finally did not implement this because the current into the sensors was to big and we did not have time to add a power multiplexer. The cross-talk issue was finally addressed by placing the sensors differently.

8.1.4 Mechanism

The mechanism explained below can be activated only in the *Move* or *Return* state.

- **Bottle capture**
  In front of the robot at the entrance of the retrieve mechanism, one of the custom IR sensor is placed to detect bottles coming. When a falling edge is detected, the mechanism activates after some time in order to avoid the rod to fall on the bottle and get stuck. As the robot moves slowly, we determined that 10 seconds would be a good start and it also permits the bottles to arrange themselves sideways which improves the bag emptying. In the initialization of the robot, a mechanism full turn is done in order to ensure its good placement. As the servo is in wheel mode and thus does not provides position, two mechanical trigger (end-stop) are used to determine when the mechanism must go forward or backward. A timer is also used for stopping the rod in the middle of the course when not in use. For ease of testing and face some mechanical issues of the mechanism, we use a switch to enable or disable it.

- **Bag emptying**
  When the robot returns in the *Home* area, the bag needs to be empty.

For this the "tail" of the robot is lifted at the same time as the "door" (bag entrance) in order not to break the bag or put to much load on the servos. Then as the bottles do not always completely fall from the bag, the "tail" does a back and forth 10 times to shake the bag. When all the bottles have fallen, the robot needs to move forward so as the "door" does not get stuck on the bottles.

- **Rock avoidance**
  If the position is entering or leaving the rocks area, the Raspberry Pi sends a message so that the robot knows that it has to lift or lower the door. The door folds inward in order to prevent a breaking by touching rocks and also to retain the bottles inside the bag.
  If the bottle capture mechanism is already running the robot stops until it is completed. The two are mutually exclusive to prevent one from interfering with the other.

8.1.5 Mechanical Inputs

We used push buttons, slider switches and end-stop to get mechanical feedback. All of those were wired with one side connected to the ground and the other to an Arduino input pin. When the button, switch or end-stop is triggered, it connects the pin to the ground. Because the signal is undefined if the wire is disconnected, the pins were configured as *input pull-up* to ensure good readings.

8.2 Raspberry Pi

Motivation

In order to localize and detect bottles, we decided to use image processing. We decided to use Raspberry Pi, since, image processing would be too intensive for Arduino, when combined with the potential field navigation which requires working with trigonometry would increase computational costs far too much for the main loop of Arduino. Thus, all processing that require a lot of computation are done on the Pi, especially the one requiring floating point operations. Although the entire processing takes around 250ms, it is not a problem as the fast feedback loop of the motors is already taken care of by Arduino. This slower rate is still fast enough for the navigation.

Functions and overall architecture

Raspberry Pi is used for the computationally intensive operations such as localization, bottle detection and navigation. Further, the commands for the wheel motors are communicated to the Arduino using Serial communication in real time. This architecture is fast and efficient as the fast computing power of raspberry pi is used in conjunction with fast motor control of Arduino which gave us a cycle time of less than 250ms for Raspberry Pi. Out of this 250ms cycle time, about 100ms is taken for bottle detection using OpenCV, additional 100ms for

localization using beacons and about 23ms for serial communication between Arduino and Pi. In addition, computations related to navigation took about 5ms and rest of the time is taken for the overall programming logic, intermediate checks etc.
The three major functionality implemented in Raspberry are :

- Localization - Using beacons
- Bottle detection - Using PiCamera and OpenCV
- Navigation - Using potential field

8.2.1 Localization

In order to accurately localize the robot in the given arena, we use Kalman filtering with odometry for prediction of the states and beacons placed at the corners for the measurement. A PiCamera with an omnidirectional mirror takes an image containing all four beacons which is further processed using computer vision to accurately measure the position of beacons. Subsequently, we estimate the angles between beacons in the image to localize the robot at any given time. Algorithm is validated by verifying the results in the actual arena. The accuracy of the measurement is found to be between ±10cm to ±25cm depending on the location, with center of the arena giving the best accuracy. Methodology and formulations for the localization are given in this section.

Algorithm

Overall flowchart of the localization algorithm using beacons is given in Figure 18. There are three major steps in localization algorithm namely, prediction using odometry, measurement using beacons and position update using Kalman filter. These three steps are described below.

Step 1: Odometry

In order to navigate in the given arena, it is important to know the position and orientation of the robot at any given time. The state of the robot can be tracked using position (x, y) and orientation(α) data which are depicted in Figure 19. We denote the initial position of the robot as X, given by,

$$X = \begin{pmatrix} x \\ y \\ \alpha \end{pmatrix}$$

Then, we can update the position using wheel rotation speeds($\omega_{L,R}$) measured by encoders on the wheel motor using the following odometry equations.

$$X^{t+1} = X^t + \frac{r\Delta t}{2} \begin{pmatrix} \cos\alpha & \cos\alpha \\ \sin\alpha & \sin\alpha \\ \frac{1}{l} & -\frac{1}{l} \end{pmatrix} \begin{pmatrix} \omega_R \\ \omega_L \end{pmatrix} \tag{1}$$

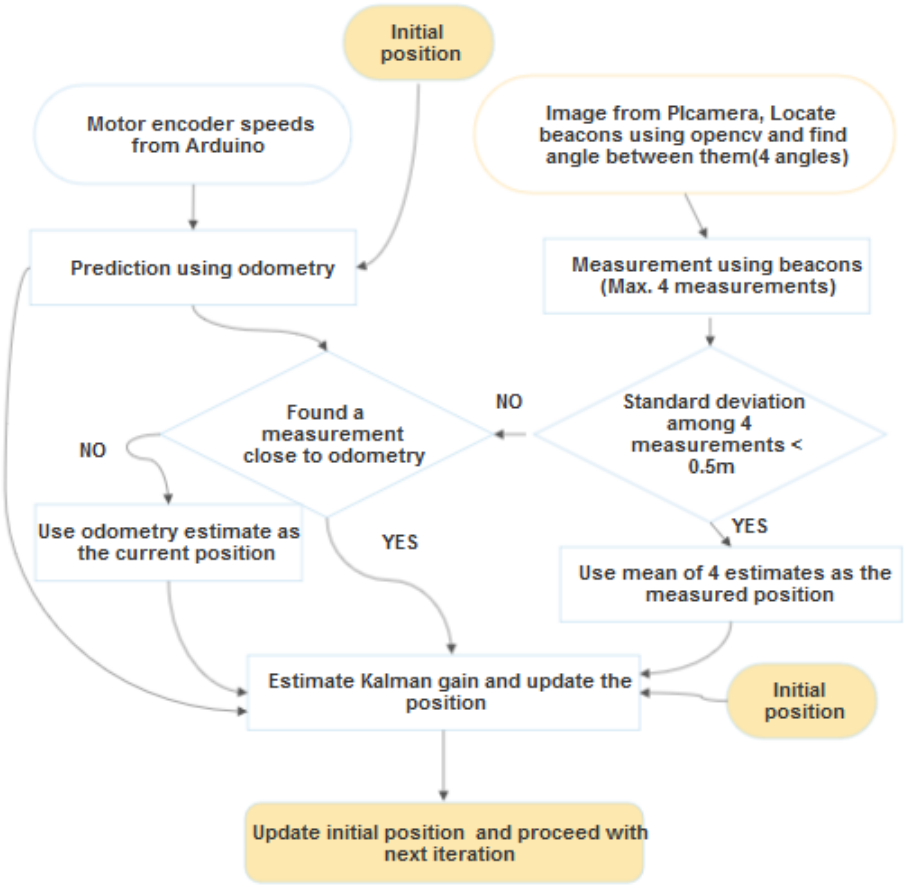


Figure 18: Flow chart for the Robot localization algorithm using beacons

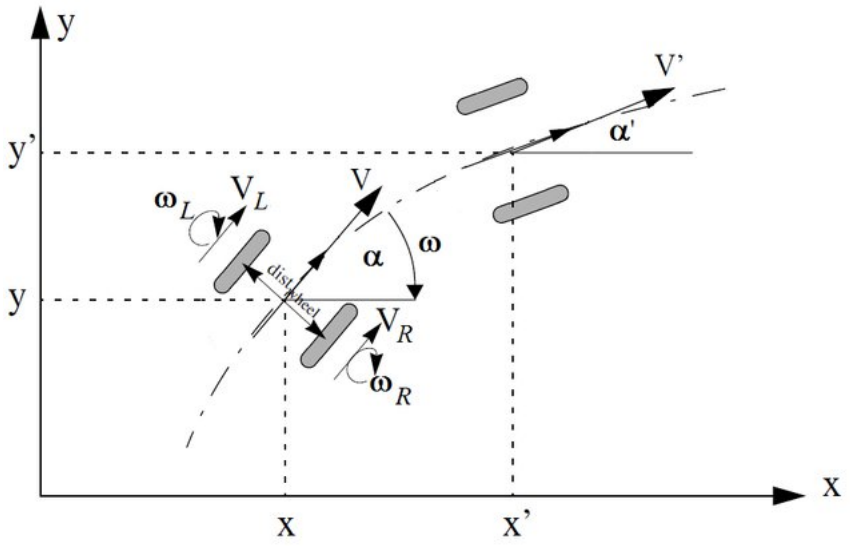


Figure 19: Odometry for the position prediction - Schematic representation

where r is the radius of the drive wheel and l is the distance between the center of the two wheels.

Step 2: Localization using beacons

State vector(X) of the robot can be measured using beacons located at each corner of the arena. This is achieved in three steps described below.

1. **Capturing image** Omnidirectional reflection mirror is placed just above the PiCamera, an image containing beacons is captured at every cycle. A sample image captured in the arena is shown in Figure 20. We notice that the beacons are visible to the naked eye, however there is lot of lighting from the arena and colors are not very distinguishable. The vision algorithm developed in this case is very efficient in distinguishing each color and identifying then accurately.

2. **Locating beacons and angle measurement** Due to the particular shape of the mirror used in our design, we always get the beacons concentrated in a particular segment between two ellipses in the image. Steps involved in beacon detection are as follows :

    - Mask the image to only show the region of interest, i.e the region containing beacons,
    - Convert the image to HSV and apply inRange filter for each color of the beacon (Values for the inRange given in code attached),
    - Convert the image to gray scale for finding contours,

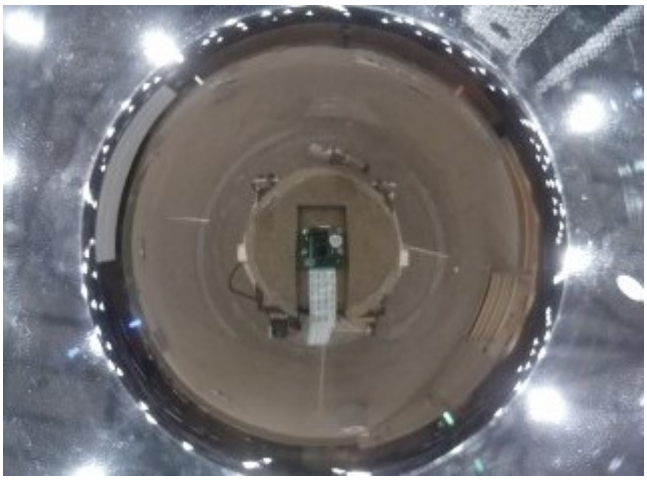


Figure 20: **Image captured by the localization camera**

- Dilate the image to make the filtered elements to appear clearer,
- Apply a binary filter along with OTSU(Choose filtering criteria automatically) to make all the elements in the image to have same color intensity,
- Use OpenCV function to grabContours and sort the contours by area,
- Check if the contour properties match the beacon contour, if so identify the contour as beacon, otherwise continue with next contour.

An image with the actual locations of the beacons as identified by the procedure describe above is shown in figure 21. We can notice that all the beacons seen in right of the image stack (Actual image from the Picamera) are identified correctly and contours of the beacons are shown accurately in the left part of the image(Processed image showing the locations of beacons with respective color). Subsequently, we estimate the centroid of contour to calculate the angle between them, which is required for the localization algorithm. Location of the beacons as identified by the vision algorithm at various places including home area, rocks area, grass area and near the platform area are shown in Figures from 22 to 25

3. **Estimate position of robot given angles between beacons** After identifying the angles between beacons, we can estimate the position of the robot inside the given arena using two adjacent angle measurements. Since, we have four angle measurements, we can have four independent estimations of the position(x, y). Procedure followed for estimating the position given two angles is given below. Refer to figure 26 for the following description. This procedure is directly adapted from the previous years report (Year 2013, Group-1 report, pages 33-34). Lets assume that the

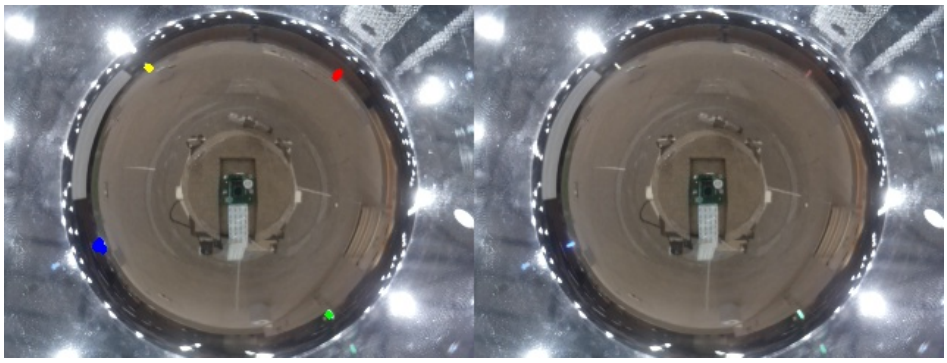


Figure 21: **Beacon detection - close to Center of the Arena**

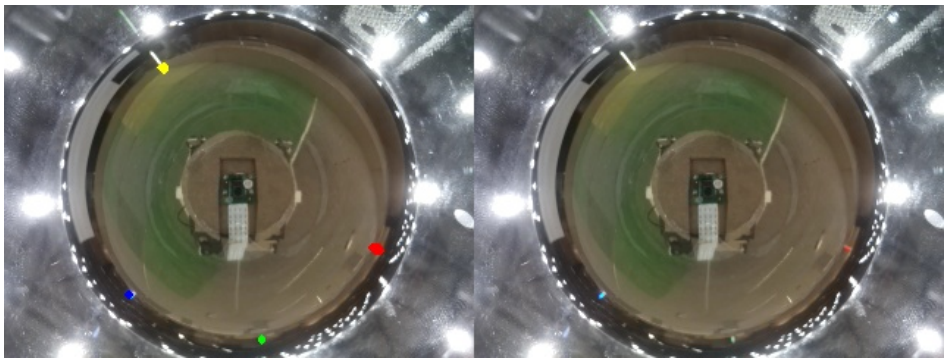


Figure 22: **Beacon detection - Home area**

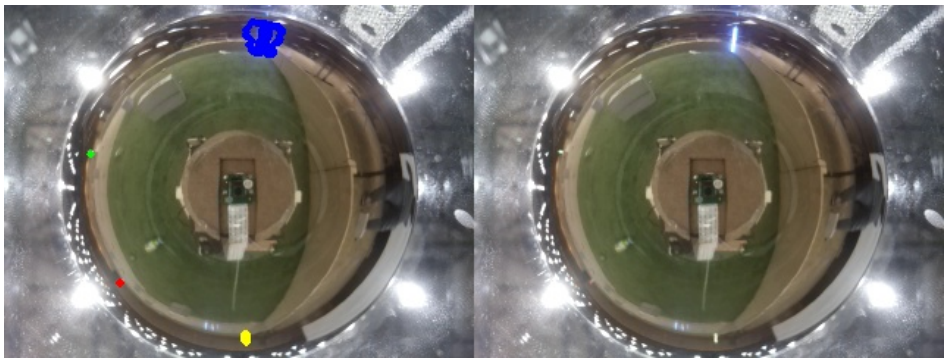


Figure 23: **Beacon detection - Grass area**

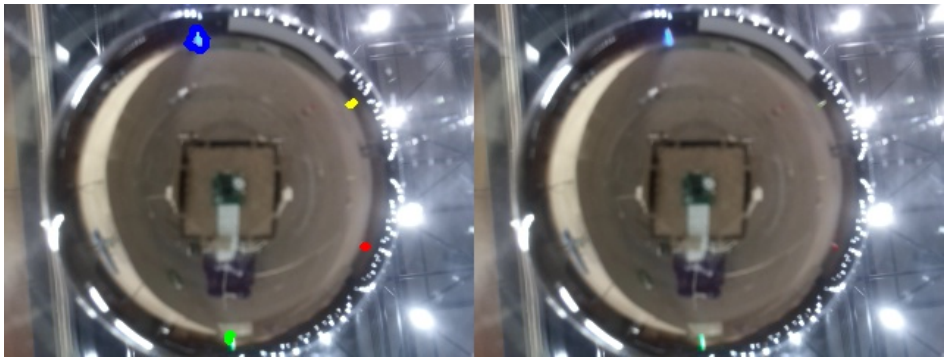


Figure 24: **Beacon detection - Platform area**

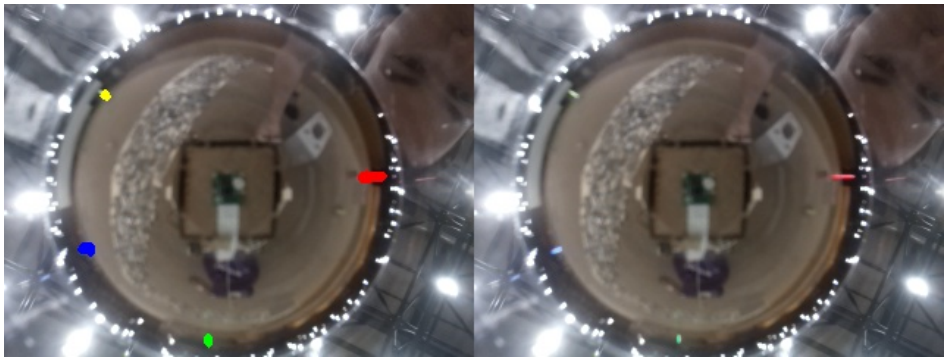


Figure 25: **Beacon detection - Rocks area**

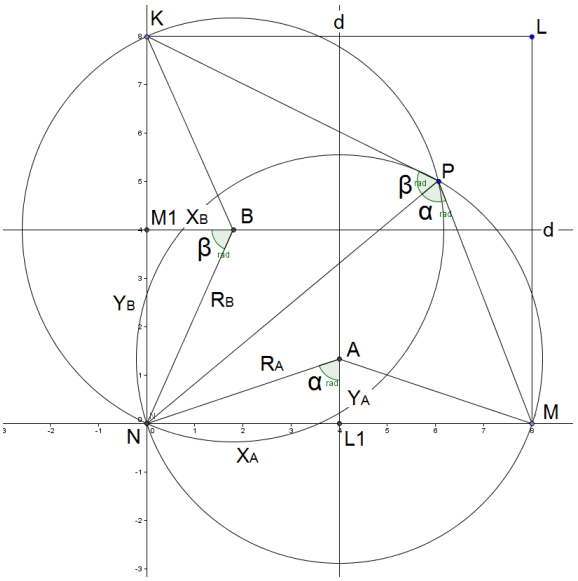


Figure 26: **Localization using beacons**

robot is at position **P**, then the angles of the beacons as seen by the robot are shown in the figure as α and β. Given two angles(α, β), there are two possible solutions for the position (x1, y1) and (x2, y2) given by following set of equations. We eliminate the solution that is not within the arena (0 to 8m).

$$x_1 = \frac{-B + \sqrt{(B^2 - 4AC)}}{2A} \qquad y_1 = N - x_1 \times M$$

$$x_2 = \frac{-B - \sqrt{(B^2 - 4AC)}}{2A} \qquad y_2 = N - x_2 \times M$$

Where A, B, C, M, N are given by following equations (d = 8m for the given arena)

$$x_A = y_B = \frac{d}{2} \qquad y_A = \frac{d}{2\tan\alpha} \qquad x_B = \frac{d}{2\tan\beta}$$

$$R_A = \frac{d}{2sin\alpha} \qquad R_B = \frac{d}{2\sin\beta}$$

$$M = \frac{x_A - x_B}{y_A - y_B}$$

$$N = \frac{R_B^2 - R_A^2 - x_B^2 + x_A^2 - y_B^2 + y_A^2}{2\left(y_A - y_B\right)}$$

$$A = M^2 + 1$$

$$B = 2y_A M - 2MN - 2x_A$$

$$C = x_A^2 + y_A^2 + N^2 - R_A^2 - 2Y_A N$$

Step 3: Kalman filtering for position update

- **Prediction step** Expectation of the predicted state ($\bar{\mu}^t$) is obtained by using the previous available position ($\mu^{(t-1)}$) as follows :

$$\bar{\mu}^t = \mu^{(t-1)} + \frac{r\Delta t}{2} \begin{pmatrix} \cos\alpha & \cos\alpha \\ \sin\alpha & \sin\alpha \\ \frac{1}{l} & -\frac{1}{l} \end{pmatrix} \begin{pmatrix} \omega_R \\ \omega_L \end{pmatrix}$$

  Covariance matrix of the predicted state is computed using the following equation where Q is the noise covariance matrix for the estimated states.

$$\bar{\Sigma}^t = I\Sigma^{(t-1)}I + Q$$

- **Meausurement step and Kalman gain** Let the measured position be X^t, then the Kalman gain is computed as follows : Covariance of the innovation (Error between measurement and prediction) is given by S^t where R is the covariance matrix of the measurement noise. Further Kalman gain($K^(t)$) is computed.

$$S^t = I\bar{\Sigma}^t I + R \qquad K^t = \bar{\Sigma}^t I \left(S^t\right)^{-1}$$

  In the final step, we update the present position estimate and its covariance using the following equations, where X^t is the measured position.

$$\mu^t = \bar{\mu}^t + K^t \times (X^t - \bar{\mu}^t)$$
$$\Sigma^t = (I - K)\bar{\Sigma}^t$$

  We continue the Kalman loop, with the updated prediction and measurement steps for each iteration.

8.2.2 Bottle detection

Motivation

One of the key objective of the competition is to detect and collect the bottles in the arena. Our initial attempts to detect bottles placed in front of the bot based solely on ultrasonic and infrared sensors were not successful. This is attributed to various reasons such as low range of sensors and too much noise from the sensors which makes this method of bottle detection unreliable. Hence, we decided to use computer vision to detect bottles.

Methodology

In order to navigate the robot towards the bottles, we are only interested in finding the bottles which are close to the robot as it moves. A typical image taken using the picamera for localization is shown in Figure 20. As we can see, the image captures very well the area surrounding the robot with decent clarity, thanks to our design of the camera mount and omnidirectinal mirror. This is a very efficient option by design, as it gets away with the need of another Raspberry Pi and Picamera for bottle detection. In addition to increasing cost, using another Pi would mean increased complexity in communication between two Raspberry Pis. In summary, we use the same image taken for the purpose of localization, to detect bottles using computer vision. Bottle detection methods are described in this section.

Bottle detection using computer vision

For the bottle detection, we explored three different options during the design phase. We tested each of them, before finalizing the method for its use in the actual competition. Three methods experimented for bottle detection are as follows :

1. **Filtering using OpenCV** Images taken from the test arena are processed using basic OpenCV which included pre-processing of the image, edge detection using canny edge filter, thresholding of the image followed by finding contours and checking if its a bottle or not based on the contour features. Finally we sort the image based on the confidence criteria. Results of this filtering approach on images from test arena are shown in Figure 27

2. **Deep learning based on latest available framework YOLOv3 (R-CNN)** We also experimented with the deep neural network module of OpenCV for bottle detection. For this purpose, we used the latest YOLOv3 architecture for training the bottle data set. A trained model on almost 180 objects is already available which includes bottles dataset. We tested the bottle detection capability of this opencv feature with the images taken from test arena. We observe that this framework accurately detects bottles in the image, shown in Figure 28. However, when tested in personal

Figure 27: Bottle detection using image processing with OpenCV

Figure 28: Bottle detection using deep learning in OpenCV

computer, this module took about 3s for bottle detection. Subsequently, when tested with Raspberry Pi, this resulted in almost 1 min for detecting bottles in an image. In order to reduce the computational efforts, we trained exclusively on the bottle dataset with a smaller architecture using yolo-tiny configuration files. However, this turned out to be not so good classifier as the original one and also it was still computationally intensive. Hence, we decided not to use this framework for the bottle detection in the actual competition.

3. **Haar Cascade Classifier** We explored less computationally intensive cascade classifier. Haar classifier was trained using Haartoolkit.exe available online by creating positive and negative datasets each containing 2000 images from the internet. However, multiple attempts in training the same went in vain due to poor convergence results.

After verifying various options for bottle detection, we decided to use OpenCV

with basic processing operations like filtering and contouring for the final bottle detection due to simplicity of the algorithm and to enable faster computations. Steps followed in bottle detection are as follows.

- Mask the image to only show the region of interest, i.e the region of approximately 1m around the robot. Further, the robot itself will be visible in the center of image by construction. Hence, the region covering the robot and back of the robot are also masked. Typical mask created for bottle detection is shown in Figure 29 in the right side of the image.
- Convert the image to HSV and apply inRange filter (Values for the inRange given in code attached),
- Convert the image to gray scale for finding contours,
- Dilate the image to make the filtered elements to appear clearer,
- Apply a binary filter along with OTSU(Choose filtering criteria automatically) to make all the elements in the image to have same color intensity,
- Use opencv function to grabcontours and sort the contours by area,
- Check if the contour properties matches the bottle contour, if so identify the contour as bottle, otherwise continue with next contour until a maximum of 4 bottles are identified. Further, sort the bottle by Area and compute the required deviation in orientation required for reaching the bottle.

A typical image taken by the camera is shown in the Figure 29 along with the results of bottle detection. We observe that all the bottles within the vicinity of the robot are detected. Since, we used only 320x240 pixels image, we could achieve approximately a frame rate of 10 using this method, which is very fast for the given application. Similarly, few other cases with bottles towards left of the robot, front of the robot and right to the robot are shown in Figures 30 to 32.

8.2.3 Navigation

We decided to use potential field navigation for the competition Since we have accurate localization and the bottle detection using computer vision. In addition, we placed various infrared sensors for obstacle detection and avoidance. Information about present location, target location, bottle direction and the obstacles enables us to define the forces to reach the target and to avoid obstacles resulting in a smooth trajectory. Forces related to target and bottle are considered as attractive forces and infrared sensor values are treated as repulsive forces representing obstacles. The weights for attractive and repulsive forces are decided based on the simulations and final tests in the arena. This section give details of the navigation algorithm.

Figure 29: Bottle detection using OpenCV. Right side shows the mask along with bottles in the arena. Left side shows the bottle detected by the algorithm. Contours indicated in red color, Centroid of the contour indicated in white color

Figure 30: Bottle detection using OpenCV. All bottles to the left.

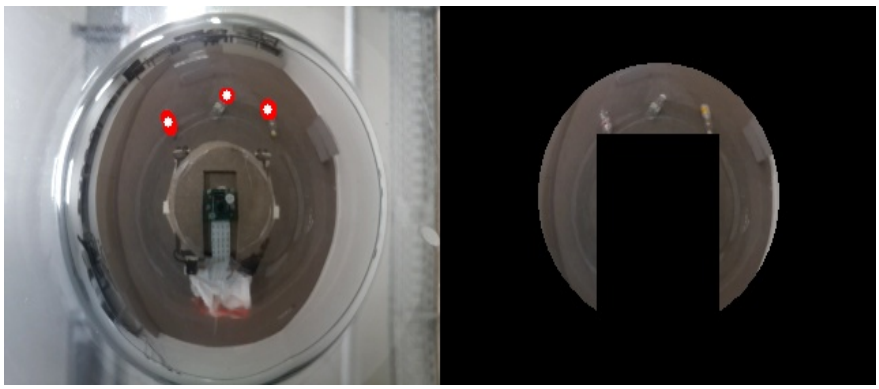


Figure 31: Bottle detection using OpenCV. All bottles in front.

Figure 32: Bottle detection using OpenCV. All bottles to the right

Potential field navigation

Net force acting on the robot is computed as follows

$$F = F_{\text{target}} + k_{\text{obstacles}} F_{\text{obstacles}}$$

Where $k_{\text{obstacles}}$ is the gain for the obstacle force to avoid the obstacles in a smooth manner.

- **Target force** Target force vector is computed based on the distance to the target and the deviation in angle required. Since, the overall objective of the competition is to collect bottles, we decided to switch targets with bottle location whenever bottle is in the vicinity of the robot. This is achieved by defining a weighting factor (α) for the target deviation angle (θ_{target} and bottle direction (θ_{bottle}. Error in orientation is always measured w.r.t present robot orientation. Hence, net error in orientation of the robot at any given instant due to target force is given by following equation.

  $\theta_{\text{required}} = \alpha \times \theta_{\text{bottle}} + (1 - \alpha) \times \theta\text{target}$ Where $0 <= \alpha <= 1$

  Where α is further defined as follows :

  ```
  if bottle is present :
      alpha = 0.995
  else :
      alpha = 0.005
  ```

  Since the max speed of the robot is only $\approx$ 15 cm/s, we decided to drive the motors at constant speed while the direction alone is being dictated by navigation algorithm. i.e to say that, we don't want to throttle the

speed when we near the target, hence we will be able to make best use of the available time. Final equation for the target force is given by the following equation

$$F_{\text{target}} = |F| \times (\cos\theta_{\text{required}} + \mathbf{1j}\sin\theta_{\text{required}})$$

where |F| is constant force.

- **Obstacle avoidance** As described in the design section, we placed 12 nos. of infrared sensors at various locations to optimize the obstacle avoidance performance. The locations of the sensors are finalized based on numerous tests in the arena. Further, we also introduced weights for each of the sensor readings to better sense and turn in case of sharp corners. The net repulsive force from the infrared sensors is computed as follows.

$$F_{\text{obstacles}} = \sum_{i=0}^{12} w_i \times IR_i \times (\beta_{1i}\cos\theta_{\text{i}} + \mathbf{1j}\beta_{2i}\sin\theta_{\text{i}})$$

Where IR_i is the sensor reading, w_i is the weight for each sensor reading, β_{1i} and β_{2i} is equal to +1 or -1 depending on the orientation of the sensor and θ_i is the orientation of the sensor w.r.t the robot frame of reference.

From the above computations, we get net force vector (F) at each instant. Subsequently, we compute the error in present orientation according the effective force vector (θ_{err}).

Navigation commands

Based on the net error in the present orientation, we compute the commands for both the motors which drive wheels.
Let u_r Command for correcting the error in 'r' direction and let $\mathbf{u}_\theta$ be the command for correcting error in orientation. Since, we don't want the robot to go back because we don't have sensors for obstacle avoidance in the back of the bot, we always provide positive speed error(u_r). For computing control input for error in orientation(u_θ), we use proportional gain (k_θ). Then the commands for right and left wheels are computed by the following equations.

$$\begin{aligned} u_r &= max(0, |F| \times \cos\theta_{\text{err}}) \\ u_\theta &= k_\theta \times \theta_{\text{err}} \\ \text{speed}_{right} &= u_r + 0.5L \times u_\theta \\ \text{speed}_{left} &= u_r - 0.5Lu_\theta \end{aligned}$$

The values for the speed are further thresholded and converted using the calibration constants to suitable integers before communicating them to Arduino via Serial.

8.3 Communication

The communication between the Raspberry Pi and the Arduino is done by Serial and pass through a USB cable.
When the Raspberry starts the Serial, the Arduio resets and in the setup, the Arduino waits for the Serial to establish before starting the program. Because there is always a possibility of failure during the competition, a fail-safe is set so that if the Arduino is reseted the program can keep going.
Each command sent to the Arduino is composed of three letters followed if needed by a number. This simplify the reading and avoid to interpret bad signals. All commands are read in a non blocking way.

Pi communication and commands

Incoming commands : start, data, return

- **Start command** : Whenever, Arduino sends a start command, the timer data stored in Pi is communicate to Arduino and time synchronization is established.
- **Data** During the course of the competition, the motor encoder data as well as IR sensors data is communicated to Pi using this command. Since, data is only sent when the Arduino is active, the timer is only updated when the data is received from Arduino. This way, we ensure that the pause is implemented when there is reset from the Arduino.
- **Return command** Depending on the global timeout of 500s, Arduino sends a 'return' command. Once the return command is received from Arduino, the targets are reset to home area and bottle detection is disabled.

Outgoing commands : time, spl, spr, hom, ret, zone

- **tim** Upon receiving start command from Arduino, timer data is sent to Arduino for enabling time synchronisation.
- **spl, spr** : Navigation algorithm computes the required speeds for left and right wheels. These speeds are communicated to Arduino in every cycle using the command 'spl' for the left wheel motor and 'spr' for the right wheel motor.
- **ret** In order to set the Arduino state to return after completing each of the navigation parts names PartA and PartB, a command named 'ret' is sent to arduino.
- **hom** : In order to activate the bottle release mechanism after reaching the home area, a command named 'hom' is sent to Arduino.
- **zone** : When the robot is near the rocks, in order to enable lifting of the back to safeguard bag from capturing rocks, the zone data is sent. This carries two values either 'roc' or 'std' where 'roc' stands for rock area.

8.4 Strategy

Since we chose the overall design of our robot with the continuous tracks, we decided that we wanted to go to the *rocks* zone. The continuous tracks seemed perfect to pass the rocks barrier at the time and only after we encountered some issues. For safety during the final competition we chose to go first to the grass area and then in the middle (zone 2 and 1 respectively). This would ensure us to get some good points to bring back before we try to go over the rocks.
We decided to spend 6 minutes in zone 1 and 2 and then return to drop the bottles. After we would go straight to the rocks area to bring back more points until the final timeout occurs. Those positions were defined only as checkpoints in order to let the freedom to follow the detected bottles.

Navigation strategy

To achieve the overall strategy for the competition, we divided the navigation into two parts each separated by a predefined timeout for each part.

- **Part A:** We created various targets(12 nos) in Zone 1 and Zone 2. Further, whenever robot passes around a target(within 0.5m), it's deleted from the list of targets. After going through each of the target, the robot is programmed to reach home area. In addition, we keep a time limit of 300s for PartA, beyond which robot resets its target to home area and stops scouting for bottles. Subsequently, when the robot reaches home area, robot is aligned before executing bottle release mechanism. Once partA is completed, the navigation switches to a different state (PartB).

- **Part B:** Similar to PartA, we created various targets in Zone 3, which is rocks area. After completing PartA, we set the targets in Zone 3 and disable detecting bottles until the robot is in Zone 3. Once robot is in Zone3, we activate bottle detection and bottle capture. In addition, whenever, the robot is close to the rock area, we enable lifting of the bag to keep it away from rocks. Further, the orientation of the robot is always maintained to be in rock area thanks to the availability precise orientation of the robot from localization algorithm.

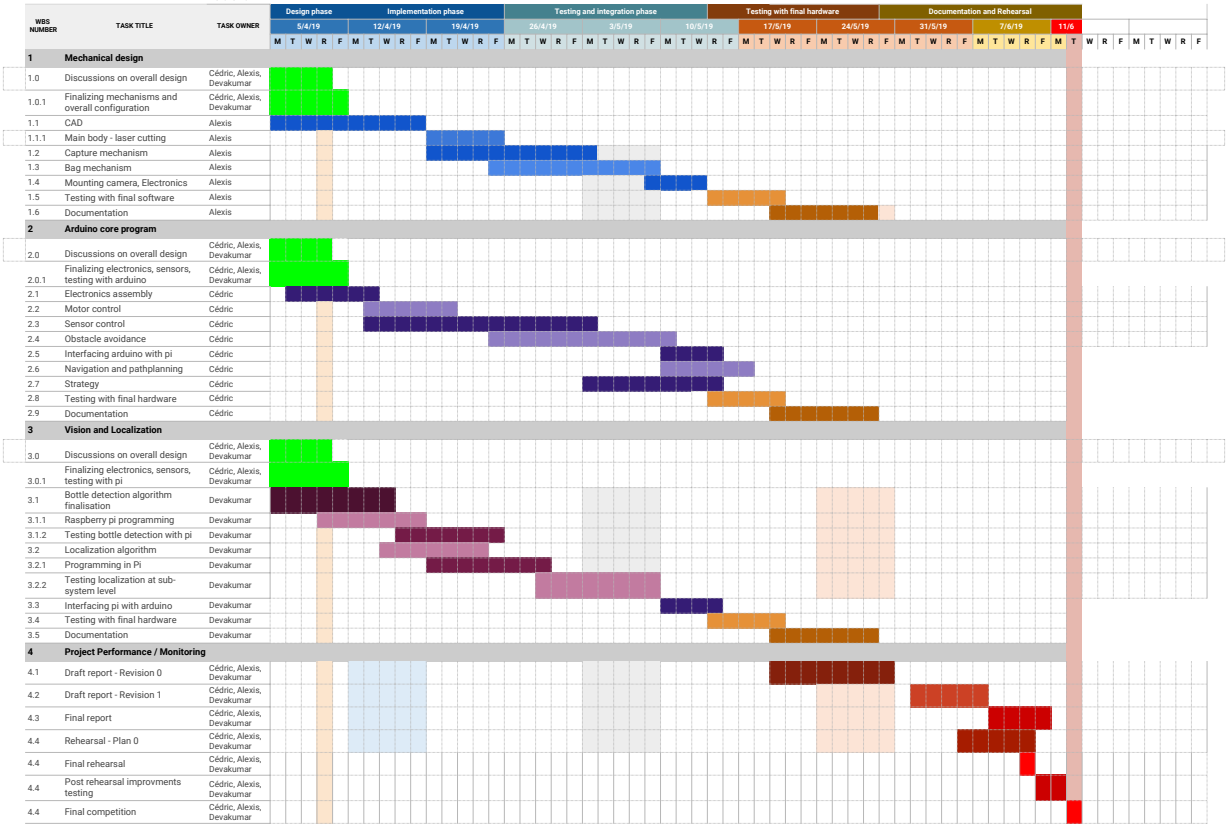
9 Time planning

STI Robotics Competition-Team 4

PeTco

PROJECT TITLE	PeTco	COMPANY NAME	EPFL
PROJECT MANAGER	Cédric, Alexis, Devakumar	DATE	03/04/19

WBS NUMBER	TASK TITLE	TASK OWNER
1	**Mechanical design**	
1.0	Discussions on overall design	Cédric, Alexis, Devakumar
1.0.1	Finalizing mechanisms and overall configuration	Cédric, Alexis, Devakumar
1.1	CAD	Alexis
1.1.1	Main body - laser cutting	Alexis
1.2	Capture mechanism	Alexis
1.3	Bag mechanism	Alexis
1.4	Mounting camera, Electronics	Alexis
1.5	Testing with final software	Alexis
1.6	Documentation	Alexis
2	**Arduino core program**	
2.0	Discussions on overall design	Cédric, Alexis, Devakumar
2.0.1	Finalizing electronics, sensors, testing with arduino	Cédric, Alexis, Devakumar
2.1	Electronics assembly	Cédric
2.2	Motor control	Cédric
2.3	Sensor control	Cédric
2.4	Obstacle avoidance	Cédric
2.5	Interfacing arduino with pi	Cédric
2.6	Navigation and pathplanning	Cédric
2.7	Strategy	Cédric
2.8	Testing with final hardware	Cédric
2.9	Documentation	Cédric
3	**Vision and Localization**	
3.0	Discussions on overall design	Cédric, Alexis, Devakumar
3.0.1	Finalizing electronics, sensors, testing with pi	Cédric, Alexis, Devakumar
3.1	Bottle detection algorithm finalisation	Devakumar
3.1.1	Raspberry pi programming	Devakumar
3.1.2	Testing bottle detection with pi	Devakumar
3.2	Localization algorithm	Devakumar
3.2.1	Programming in Pi	Devakumar
3.2.2	Testing localization at sub-system level	Devakumar
3.3	Interfacing pi with arduino	Devakumar
3.4	Testing with final hardware	Devakumar
3.5	Documentation	Devakumar
4	**Project Performance / Monitoring**	
4.1	Draft report - Revision 0	Cédric, Alexis, Devakumar
4.2	Draft report - Revision 1	Cédric, Alexis, Devakumar
4.3	Final report	Cédric, Alexis, Devakumar
4.4	Rehearsal - Plan 0	Cédric, Alexis, Devakumar
4.4	Final rehearsal	Cédric, Alexis, Devakumar
4.4	Post rehearsal improvments testing	Cédric, Alexis, Devakumar
4.4	Final competition	Cédric, Alexis, Devakumar



Our worked was much more parallelized in comparison to the planning given during the presentation. Some mechanical parts were finished a couple weeks late and some programming parts were finished earlier than expected.
Something that we can observe in this Gantt chart is the fact that at the end mostly the programming remains. As it is difficult for the three team members to work on the same program, the work tended to be unbalanced during the last couple of weeks.

10 Budget

Our plan was if possible to make an "economic" robot and we managed for the most part to make it very cheap. The biggest expanse as we can see in figure 33, is the 3D printed parts for which the tracks elements were the more expensive. Each link would cost around 0.70chf and we needed about 100 per side. The alternative was to buy already made tracks but we had a hard time finding one that satisfied our criterion.

ID	Group	Creator	Budget	Timestamp	Amount	Description
1603	4	acrespi	real	26/02/2019 11:47:07	750.00	Initial budget
1753	4	acrespi	real	01/05/2019 11:11:10	-18.80	Commande Conrad du 25.4.2019
1754	4	cotture	real	01/05/2019 15:27:22	-2.70	Achat 1x MDF 5mm
1755	4	acrespi	real	02/05/2019 10:44:19	-89.30	Commande Play-Zone du 25.4.2019
1756	4	acrespi	real	02/05/2019 10:45:03	-1.45	Commande Distrelec du 25.4.2019 (article pris chez DigiKey)
1764	4	acrespi	real	06/05/2019 10:41:05	-16.60	Commande Distrelec du 25.4.2019
1774	4	cotture	real	10/05/2019 15:25:27	-2.70	1x MDF 5mm
1775	4	acrespi	real	13/05/2019 10:15:26	-5.40	Commande Distrelec du 25.4.2019 (2ème facture)
1780	4	cotture	real	16/05/2019 07:54:57	-5.40	Achat 2x MDF 5mm
1797	4	cotture	real	22/05/2019 15:19:42	-2.70	Achat MDF 5mm
1803	4	cotture	real	29/05/2019 15:07:56	-5.40	Achat MDF 5mm (2 plaques)

Current budget balance: 599.55 CHF.

ID	Group	Creator	Budget	Timestamp	Amount	Description
1598	4	acrespi	virtual	26/02/2019 11:46:32	2000.00	Initial budget
1680	4	acrespi	virtual	02/04/2019 11:22:46	-36.20	Raspberry Pi 3 (model B) [#4]
1681	4	acrespi	virtual	02/04/2019 11:22:46	-44.50	Arduino Mega2560 R3 [#5]
1682	4	acrespi	virtual	02/04/2019 11:22:46	-45.00	DFRobot DRI0018 2x15A motor driver [#5]
1683	4	acrespi	virtual	02/04/2019 11:22:46	-31.55	Raspberry Pi Camera [#11]
1684	4	acrespi	virtual	02/04/2019 11:22:46	-15.00	Dynamixel AX-12A motor [ID 6]
1685	4	acrespi	virtual	02/04/2019 11:22:46	-36.95	75:1 DC motor with encoder (HP) [#5]
1686	4	acrespi	virtual	02/04/2019 11:23:12	0.00	Arduone connector/power board [#3]
1720	4	acrespi	virtual	17/04/2019 16:55:24	-15.00	Dynamixel AX-12A motor [ID 7]
1721	4	acrespi	virtual	17/04/2019 16:55:24	-25.00	Arduone v.1.21 connector/power board [#1]
1722	4	acrespi	virtual	17/04/2019 16:55:24	-18.60	80 cm IR proximity sensor [#22]
1723	4	acrespi	virtual	17/04/2019 16:55:24	-34.95	75:1 DC motor with encoder [#2]
1724	4	acrespi	virtual	17/04/2019 16:55:24	-18.60	80 cm IR proximity sensor [#43]
1725	4	acrespi	virtual	17/04/2019 16:55:24	-15.00	Dynamixel AX-12A motor [ID 2]
1726	4	acrespi	virtual	17/04/2019 16:55:24	-18.60	80 cm IR proximity sensor [#33]
1727	4	acrespi	virtual	17/04/2019 16:55:24	-15.00	Dynamixel AX-12A motor [ID 5]
1728	4	acrespi	virtual	17/04/2019 16:55:24	-34.95	75:1 DC motor with encoder [#5]
1817	4	acrespi	virtual	05/06/2019 13:38:40	-19.95	NiMH rechargeable battery pack (7.2 V, 3000 mAh) [#2B]
	4	(automatic)	virtual	14/06/2019 11:17:37	-310.12	3D printing: 609.18 cm³ model, 302.92 cm³ support

Current budget balance: 1265.03 CHF.

Job owner	Group	Completed	Name	Model (cm³)	Support (cm³)	Price
Alexis Georges Schizas	4	16/04/2019 10:00	Pack_TRACK_ELEM	0.04	0.26	0.10 CHF
Alexis Georges Schizas	4	16/04/2019 13:52	TRACK_ELEM	2.13	1.20	1.13 CHF
Cédric Nils Athanasiadès	4	24/04/2019 13:32	down_servo_holder	15.25	2.55	6.05 CHF
Cédric Nils Athanasiadès	4	24/04/2019 18:59	Pack_side_servo_holder	40.95	7.57	16.50 CHF
Cédric Nils Athanasiadès	4	25/04/2019 15:45	Pack_wheel	36.89	7.58	15.12 CHF
Cédric Nils Athanasiadès	4	26/04/2019 10:01	Pack_chain_flat	1.93	1.85	1.28 CHF
Cédric Nils Athanasiadès	4	26/04/2019 21:10	Pack_chain_flat	47.74	23.68	24.28 CHF
Cédric Nils Athanasiadès	4	30/04/2019 21:58	Pack_wheel	56.28	36.50	31.55 CHF
Cédric Nils Athanasiadès	4	03/05/2019 12:09	Pack_pulley_holder	10.70	5.62	5.55 CHF
Cédric Nils Athanasiadès	4	07/05/2019 19:41	Pack_motor_holder	80.75	38.24	40.46 CHF
Cédric Nils Athanasiadès	4	08/05/2019 14:28	Pack_chain_flat	39.07	37.32	25.97 CHF
Cédric Nils Athanasiadès	4	08/05/2019 23:19	Pack_belt_holder	43.78	35.50	26.96 CHF
Cédric Nils Athanasiadès	4	10/05/2019 00:00	Pack_chain_flat	39.15	37.12	25.93 CHF
Cédric Nils Athanasiadès	4	13/05/2019 18:39	Pack_slider	20.16	10.34	10.37 CHF
Alexis Georges Schizas	4	21/05/2019 15:06	Pack_CLAPET	8.17	3.10	3.83 CHF
Alexis Georges Schizas	4	22/05/2019 13:39	Pack_CLAPET_2	12.68	5.83	6.29 CHF
Alexis Georges Schizas	4	01/06/2019 15:13	Pack_wheel_motor	30.96	5.99	12.56 CHF
Alexis Georges Schizas	4	04/06/2019 13:47	Pack_SUPPORT_CHENILLE	29.16	13.86	14.63 CHF
Cédric Nils Athanasiadès	4	05/06/2019 02:23	Pack_TENDEUR_CHENILLES	57.81	24.01	27.82 CHF
Alexis Georges Schizas	4	05/06/2019 10:04	Pack_SUPPORT_CHENILLE_2	7.76	1.68	3.21 CHF
Alexis Georges Schizas	4	08/06/2019 20:55	Pack_wheel	27.42	2.89	10.31 CHF
Alexis Georges Schizas	4	11/06/2019 15:45	calicat	0.40	0.23	0.21 CHF

Figure 33: Real, virtual and 3D printer budget respectively

11 Challenges encountered

As soon as we were able to test our robot on the real arena, we realized that getting over the rocks to get to Zone 3 wasn't as easy as we thought. Unfortunately, we had some issues due to the tension in our tracks leading to the tracks sometimes coming off. We observed that when the friction was too high between the ground and the bottom of the track (when going over rocks for example), the track would slide outwards. In an attempt to avoid this issue, we 3D printed a few elements to help keep the tracks in their axis, as well as a piece that help the tracks against the motorized wheel to avoid skipping a track element. We also tried adding a fifth wheel on the ground-touching part of the track to reduce the distance of track between two wheels. In the end, our robot was able to cross the rocks without trouble as long as it's trajectory was close to being perpendicular to the rocks. Due to the random disposition of the rocks, sometimes it could struggle a bit if a larger rock was just underneath the track, but all in all we think that the objective of being able to cross the rocks was achieved.

Another issue we encountered was that our first versions of the guiding mechanism of the rod for the bottle capture had too much friction, which led to the rod not being able to get up to its "high" position. There was also too much give, which sometimes resulted in the rod falling out of the mechanism. We greatly diminished the give, added a slight slope to help avoid losing the rod when transitioning from the "high" to "low" position and diminished the upwards slope to reduce friction. After this, the mechanism was running smoothly without issue. We had to clean it from time to time, since MDF produces a lot of particles, it would start to feel stiff again after repeated usage, but by simply cleaning the rod from time to time we were able to avoid any issues.

Unfortunately, we realised that in some rare cases, the cap of a bottle could get stuck between the door of the bag and the side of the robot (where the mechanism was fixed. This caused the door to slightly bend backwards since it was flexible, and the bottle could not get in. We tried to change the orientation of the door so that it would bend less, added slopes on the door to help the cap slide, but while it did seem to help a bit, it did not completely solve the issue. The solution to this would have been to redesign the mechanism and the door so that the door was larger than the mechanism, to make it so that there isn't anywhere for a bottle cap to get stuck. Unfortunately, by the time we noticed this issue, it was already too late to completely redesign the whole system.

12 Results and discussion

12.1 Results

- **Practice competition**
  We had some issues with the navigation. The obstacle avoidance software

worked, but we realized the sensor positioning wasn't optimal since the robot would sometimes see the obstacles too late or not at all if it came at a bad angle. We decided to place them differently in order to see the obstacles sooner. Due to a lack of time we weren't able to implement bottle capture/release inside the navigation. A few bottles were caught under the robot, but because of the navigation issues it got stuck and wasn't able to bring them back.

- **Final competition**
  Our robot spent most of the time roaming the arena, captured a large amount of bottles that were either completely in the bag, or dragged along at the entrance of the bag. It came back to the home area after spending more than half of the time roaming. There was an issue with the motor that actuated the door on which the bag was attached, so even though the release script was executed the door didn't open and the bottles stayed inside the robot, which ended up in us scoring 35 points, 10 by the full points of a bottle being brought earlier, and 25 by the points of all the bottles inside the robot.

12.2 Discussion

We aren't satisfied with the results of the practice competition. We were desperately trying to fix most of our problems and didn't have a working result. We would've prefered to have a working alternative on which to build on and improve rather than trying to make our complex solution work right away. Looking back, we think we should have build a temporary robot just to test navigation, localization, bottle detection and slowly add new elements on top.

Regarding the Final competition, we have mixed feelings on the outcome. While it was not perfect and still had a lot of issues, our robot chose it's path, captured many bottles and was able to come back to the home area. We had an issue with a motor that we had never encountered before. Due to a too high torque, it went in a state where it had to be reset. Out of all our tests we had performed, this had never happened to us, so we were very surprised. We realise we should have tested everything more and to be prepared for the worst, but since our door was working 100% of the time, we focused on improving the other elements with higher failure risks first. Had the door worked, our robot would have gotten 110 points, which we would have been satisfied with.

13 Improvements

To summarize all that went wrong and what could have been done to avoid it, here is a list of all the improvements we would have made for each section.

13.1 Tracks

We would have designed our robot with the possibility to completely adjust the tension in the tracks at any time. We would also have made it so that there was less give between each track.

13.2 Motors

We would have used more powerful motors. Since our robot was pretty heavy, the speed wasn't very high and it didn't have much power. It would have been easier to get over the rocks with more power in the tracks.

13.3 Sensor placement

To avoid any dead angles, we would have placed more sensors on the front and sides of the robot. Since we still had a lot of budget in the end, we could also have invested in higher end sensors.

13.4 Battery

We would definitely have bought a bigger battery, and more reliable as well (we had some bad connector issues).

13.5 Mechanism

We would have made the mechanism more reliable by making it impossible for bottles to get stuck anywhere. We would have made the whole robot larger just to increase the capacity of the bag as well. It might also have helped with the issue of our bottles getting stuck sometimes. We would also have investigated ways to make the mechanism faster.

14 Conclusion

The most important thing we take from this project is that we learned a lot. We realised it was a lot more difficult than expected, we encountered many more problems than we thought we would, not everything worked the way we thought it would, and we wish we had had more time, or that we had managed it better.

Even with the few problems that our robot had, we are glad we chose this specific design. Implementing both localization and bottle detection with a spherical camera was challenging, but all in all we are proud of the result. We definitely could have chosen a simpler mechanical design, but we think we have at least demonstrated that our design can work, and with a little more time and effort put in it, it could deliver very satisfying results. We decided to push our idea until the end to see what it was capable of and are glad we did so, because it taught us a lot, both in the technical aspects of the design/conception and

the management and organization of such a large project.

We would like to greatly thank Pr. Auke Ijspeert and Alessandro Crespi for making all of this possible and giving us the amazing opportunity to participate in this contest. We also want to thank our coach Özdemir Can Kara, for giving us advice and most of all believing in us until the end and always greeting us with a smile. We want to thank everyone that gave us a hand over at the SKIL. Finally, we also want to thank and congratulate all the other teams, for being such good sport during all the semester, sharing ideas and overall creating a really good atmosphere for this project.

15 References

1. "Rule book – Robot competition." [Online]. Available: https://robot-competition.epfl.ch/info/rulebook-2/. [Accessed: 14-Jun-2019]

2. A. Rosebrock, "PyImageSearch - Be awesome at OpenCV, Python, deep learning, and computer vision," PyImageSearch. [Online]. Available: https://www.pyimagesearch.com/. [Accessed: 14-Jun-2019]

3. "Reports and code from previous years – Robot competition." [Online]. Available: https://robot-competition.epfl.ch/wp-content/uploads/2018/11/group1-1.pdf. [Accessed: 14-Jun-2019]

4. "Reports and code from previous years – Robot competition." [Online]. Available: https://robot-competition.epfl.ch/info/old/. [Accessed: 14-Jun-2019]