Robot Competition 2022

Well-E Robot

Jiangfan Li, Jianan Mao, Ramy Charfeddine

École polytechnique fédérale de Lausanne

EPFL

Robot Competition 2022

Well-E Robot

by

Jiangfan Li, Jianan Mao, Ramy Charfeddine

Student Name	SCIPER Number
Jiangfan Li	337387
Jianan Mao	321124
Ramy Charfeddine	295758

Professor: Auke Ijspeert
Supervisor: Alessandro Crespi
Coach: Kai Junge
Institution: École polytechnique fédérale de Lausanne

EPFL

Contents

List of Figures

List of Tables

1 Introduction

The interdisciplinary robot competition from the EPFL STI faculty aims at designing, constructing and programming an autonomous cleaning robot able to move in a predefined arena that features obstacles and different terrains. The robot is intended to detect and gather empty PET bottles, which should then be brought to a specific recycling area. This time, the competition has involved four teams of three students with different educational backgrounds. The project being part of the Discovery Learning Program of EPFL, it is a unique opportunity to put into practice the theoretical knowledge acquired, while learning several new skills that are often asked for by employers nowadays: hands-on expertise in several domains (electronics, mechanics, programming...), project management, budget management, interacting with workshops and suppliers, working in a multidisciplinary team.

1.1. Working environment of the robot

The robot operates in an arena of 8 m x 8 m, consisting of four different types of terrain and a recycling area. An example of the arena's configuration is shown in Figure 1.1 but is actually slightly different from the actual competition's environment.

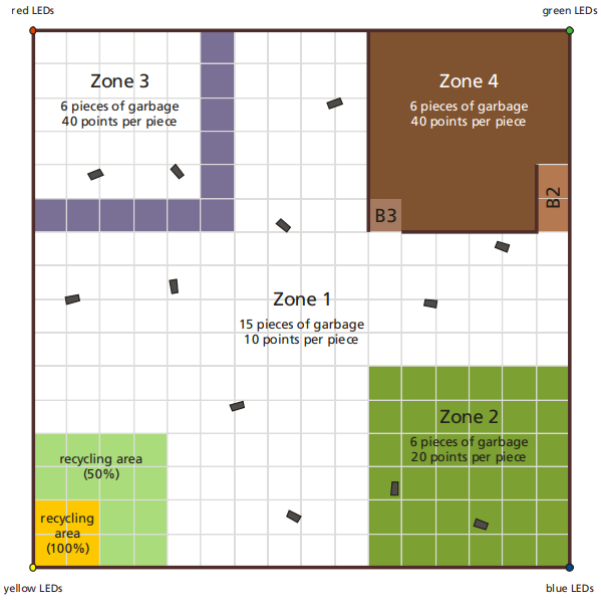


Figure 1.1: An example of configuration of the arena

As stated on Figure 1.1, each zone has specific features and makes the robot score differently depending on where the PET bottle has been collected.

- Zone 1: Made of carpet tiles, each bottle has value of 10 points.
- Zone 2: With a moderately rough terrain (artificial grass), each bottle has a value of 20 points.
- Zone 3: Made of carpet tiles as well, its access is harder as it is surrounded by rocks. Each bottle has a value of 40 points.
- Zone 4: A raised platform at approx. 30 cm off the ground, which can be accessed via a gentle slope (B2) and some steps (B3). Each bottle has a value of 40 points.
- Recycling area: The carpet tiles are colour coded. Each bottle deposited on the yellow corner earns the full points. The other surface, painted green, gives 50% of the points of the bottles it receives.

Obstacles are randomly distributed in all the environment besides the recycling area. Each corner of the arena is distinguished with a coloured bright vertical strip of LEDs.

1.2. Team members and motivation

Our team is made of Jiangfan Li, Ramy Charfeddine (both robotics master students) and Jianan Mao (mechanical master student). The three of us were thrilled by this project especially by its interdisciplinary aspect and the opportunity it gives to build something from scratch and learn lots of new skills at the same time. All along the design and implementation of our robot WELL-E, we had a value that we adopted from the beginning. The idea wasn't to make the most complex robot but rather the one that can adapt the easiest to different situations while being as simple as possible. Thus, the objective was to design the simplest robot that can win the competition and that's what we achieved. This report states all the different phase we have been going through to develop our robot starting with the reflection and strategy phase to the design and implementation phase of each of the robot's features. The challenges encountered as well as the project management process are also tackled.

2

Strategy phase

This chapter tackles the different strategies and discussions we had at an early stage of the project and the different possibilities we put on the table for each of the main robot's features. This will thus give an overview of the robot we imagined building after analyzing pros and cons of the different approaches. Even though after trials and testings, the final robot has changed a lot from it, the design in the early stage still provides an informative guideline our actual later on works.

2.1. General robot design strategy

The first question one has to ask himself when participating in such competition and when committing to this kind of ambitious project is what do we actually want to do? Which kind of robot do we want to build? Which kind of configuration do we want to go with? Although these questions can be trivial in a certain way, it was actually not that obvious for us to go with a unique differential drive robot and several suggestions have been evaluated as stated in Table 2.1.

Table 2.1: Comparison of different bottle collection ideas

Robot configuration	Details	Pros	Cons
A unique robot	One robot integrates all the functions, it has a collection system and storage space	Easiest solution Requires the less material	Slow bottle collection Less efficient Limited storage
Multiple identical robots	Similar to the first solution, but with multiple robots (2-3). Treat other robots as moving obstacles.	Faster bottle collection Less storage needed	More expensive More complex solution
Two robots A and B	Robot A mainly picks up bottles and has small storage space. Robot B has larger storage space and only transfers bottles from robot A to recycling area	Efficient and fast bottle collection	May encounter problems of robots' connection
A tower and a group of small identical robots.	A high tower robot moves to the centre of the arena at the beginning and detects the position of obstacles, bottles, different zones. Then it sends comments (moving paths) to small robots to pick up bottles and bring them to the recycling area.	Efficient bottles' collection. Motion of robots in a known map	The tower has to be stable and within required robot dimensions. Can't insure a very accurate map

All these suggestions were ideas for which it was worth it to evaluate the pros and cons and which made us realise that in the short time we have to design, construct and program our robot we should definitely go with the easiest solution. Therefore, we decided to develop a unique differential drive robot that will at the same time be able to collect the bottles, navigate autonomously without any external help and with a storage sufficiently large to collect a suitable number of bottles.

At that stage of the project, the real dimension weren't that precise yet but we had a rough idea that to be able to navigate in all the arena and access all the zones, some dimensional constraints need to be taken into account. Considering that the entrance of the raised platform is 50 cm and the height of a bottle is around 22 cm, the first approach we had was to design a robot with a body of about 30 cm large to store as many bottles as possible. Also the base should be high enough to be able to access the zone surrounded by the rocks.

The drawing of the initial design of the robot is shown in Figure 2.1.

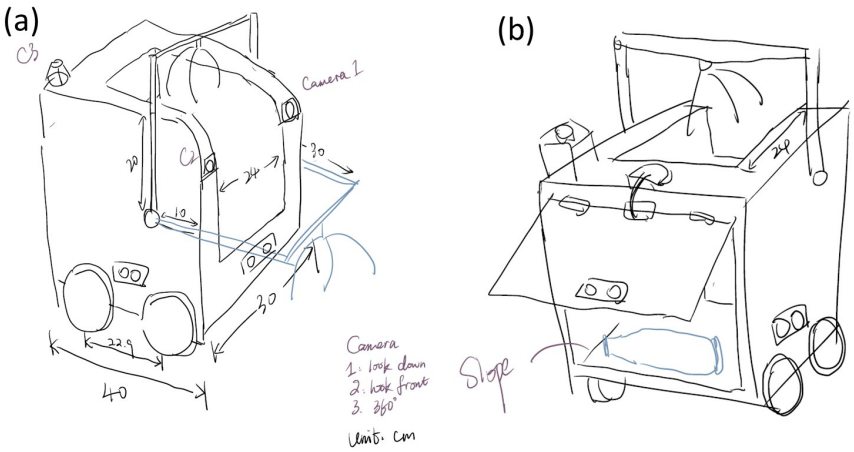


Figure 2.1: The initial design of robot: (a). front view; (b). back view.

2.2. General navigation strategy

In order to collect as many bottles as possible, the robot should be able to navigate in as many zones as possible, in a certain order, and taking into account the value of the bottles in each zone. Besides, it would be even better if the robot could follow the most optimal path possible. Different strategies and possible algorithms have been discussed and the evaluation of their pros and cons are stated in Table 2.2

Table 2.2: Comparison of different navigation strategies

Navigation strategy	Pros	Cons
Random navigation only	Easiest solution to implement	Not efficient Outcome highly depends on initial bottle placement
Go to a zone and random search in it	Reduces the random search to a smaller zone	Requires an accurate localization to be sure to reach the wanted zone
RRT* algorithm	Powerful algorithm to cover all the arena and build a map of the environment	Rather complex solution that requires some time to be implemented properly

After some discussion, the navigation strategy chosen is to define a path made of way points that takes the robot to different zones. The idea is then to let to robot do the bottle detection when following the path and let it update its way points' list when a bottle is detected. This solution is rather simple and even if it's absolutely not the most optimal path and doesn't necessarily efficiently cover the whole arena, it is a suitable solution for our navigation strategy due to the limited time for this project. The idea is to take the robot to all the different zones starting with the one surrounded by rocks in order to insure the highest number of points, then going to the grass area while going through the carpet zone. The platform is intended to be the last zone covered if the robot still has time to do so. The actual

implemented path is presented later in the implementation phase chapter. When the robot has reached a predefined maximum number of collected bottles, it should update its path and go to the recycling zone.

2.3. Locomotion

We agreed to have 4 wheels or maybe 6 later if our robot is too heavy. The wheels should be controlled in 2 channels at the same time so that it's able to turn in place, which facilities the later path planning parts.

2.4. Localization

In order to be able to navigate in its working environment and follow its path, the robot needs a localization system to determine its x-y position as well as its orientation. Different localization solutions exist. Whether if it is an absolute or relative positioning method, the different discussed solutions are stated in Table 2.3.

Localization strategy	Pros	Cons
Odometry	Easy to implement	Error accumulates quickly and positioning gets wrong with time
GPS	Relatively cheap, gives an accurate position and orientation. Navigation can easily be integrated	Doesn't work indoor
IMU	Real time localization Localization with only one sensor accurate despite slipping	Drifts greatly
LED beacons detection 360° vision	Absolute measurement of position and orientation. Gives an accurate localization	Sensitive to ambient light. Requires accurate beacon's detection thresholds Hard to use close to a beacon
LIDAR	Provides an accurate map of the arena with obstacles Very powerful	Need high computational power Complex implementation
RFID tag detection	Easy to implement	Low localization accuracy Only works if the robot's detector is exactly on it

Table 2.3: Comparison of different localization strategies

First of all, the localization method that is suitable to be chosen has to give an absolute positioning of the robot. Indeed, if the localization is relative, the new position will always rely on the accuracy of the previous estimated one and with the inevitable errors, the localization will loose accuracy with time and the robot's behaviour might be affected during its motion. Besides, the idea was to choose a method that gives an accurate position and orientation at all time without any complex implementation or high computational power needed. Therefore, after some reflection the wiser choice seemed to be the LED beacons' detection. Thus, even if some thresholds tuning was needed, we could easily localize the robot in all situations. Besides, in order to deal with the cases where the beacons' detection doesn't work properly, odometry is used until the initial method gives a correct result again. These cases being uncommon and not occurring for a long period of time, one can consider that odometry will give a reasonably correct result before being recalibrated. We also kept the digital compass as a backup solution in case the localization with the beacons doesn't work as expected, but we finally didn't need to implement it as the results were already of high accuracy.

2.5. Obstacle avoidance

At the beginning we planned to mount the range sensors around the robot so that it's able to see all of the obstacle in any direction. And ultrasonic sensors are chosen to serve as the range sensors monitoring the distance from the obstacles, because they are easy and more stable than the infrared sensors.

2.6. Bottle detection

We didn't discuss much about the approach to search for the bottles because we were all consent to use deep learning models to detect the bottles. So the question is which model?

First we tried Mask RCNN, which is able to get the pixel mask of the object rather than a simple box, so that it might be possible for the robot to deduce the status of the bottle – is it standing or lying on the ground? But later we realize that this is a huge model and it take 10 seconds to detect the bottles even on the computer, not to mention the Raspberry Pi and other edge devices. We then tried haar cascade, but we think it's of low accuracy though it's quick.

Finally we turned to small networks that someone used previously on Raspberry Pi, and the speeds are listed in Table 2.4

Model	Extra Device	Speed(FPS)	Note
Efficientdet_lite0	None	2.3	Source
Efficientdet_lite0	Coral's USB Accelerator	15	Source
MobileNetV3-SSD	None	8	Source
MobileNetV3-SSD	Coral's USB Accelerator	24	
Tiny Yolo	Movidius NCS	4.5	Source

Table 2.4: The speed of some object detection models on Raspberry Pi

It seems that we can start with tflite models. If the speed is not high enough, we can order a Coral USB Accelerator to facilitate the detection.

Since small models will have lower accuracy in bottle detection, we thought about to add some range sensors in front to confirm the detection result.

2.7. Bottle collection and storage

We discussed different methods of collection bottles and they are summarized in Table 2.5. We expected a robust but fancy method to collect the bottles. Therefore we decided to combine the sticky plane mechanism with a cable-driven soft gripper [4]. Even if the soft gripper didn't work as we assumed, the sticky plane could be the back up plan. The idea of the bottle collection system is that two arms control a plane that moves up and down. A soft gripper is attached to the plane and the servo control of the close of gripper.

Collection idea	Detail	Pros	Cons
Arm and big gripper	An arm of 4 DOF or 6 DOF with a gripper as the end effector.	Large workspace; able to deal with bottles in different positions and orientations; can be expanded to collect other trash	Complex; expensive and heavy if the accuracy is critical
Arm and vacuum suction	An arm with a vacuum to suck the bottle	Same as gripper	The air compressor is noisy; hard to deal with different orientations of bottles
Sticky plane	A sticky plane to stick the bottles and then hit a pole to detach it	simple; quick; only one rotary motion; robust	Viscosity of the sticky plane will age. Need to change it often
Two 'hands'	The shape of two hands like a scissor but are able to hold up the bottles	Robustness depends on the design;	Complex; more DOF; two hands need to move together
Rotating brushes + belt	Two rotating brushes help bottles enter the belt	Don't need to detect bottles and don't need to stop the motion to pick up the bottles.	Complex mechanism, problem with the rocks area and raised platform zone.
Rotating brushes + elevator	Two rotating brushes help bottles enter the elevator, and then delivery to storage	Robust	Elevator takes time; hard to deal with different orientations of the bottles

Table 2.5: Comparison of different bottle collection ideas

2.8. Processing unit

The choice of the hardware control is the Arduino mega 2560, which is powerful enough to deal with all the sensors and the actuators we are going to use. And another reason is that we are more familiar with Arduino rather than the STM32 or the other MCU.

We desired to use Jetson Nano as the main processing unit because we want to use deep learning model to detect the bottles. However, it was taken before we asked for it. And it's out of stock in almost all the online stores. So we had to choose small models which can run even on Raspberry Pi.

Figure 2.2 shows the initial idea about the control.

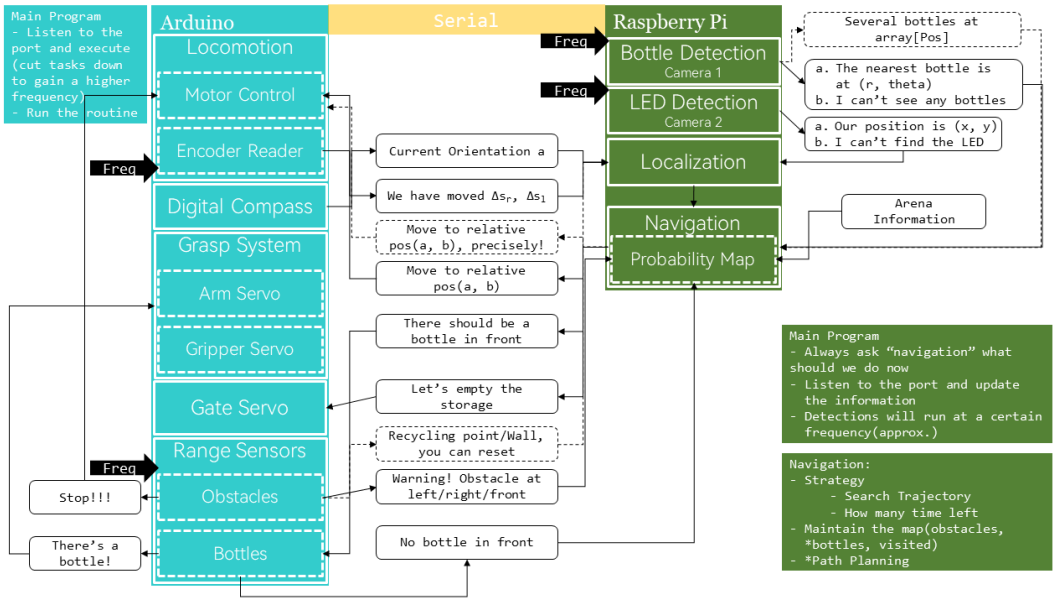


Figure 2.2: Functions analysis

3

Implementation phase

In this chapter, we will present our final design of the robot. It's not the same as what we designed in the last chapter. We implemented, tested, found some problems, re-designed and implemented again. After several such cycles, the robot becomes what it is now. This chapter thus gives the details of implementation of each part of the robot whether if it is of mechanical aspect, hardware or software, everything is gathered here.

The specific models of the components used are listed in Appendix A.

3.1. Overall Design

We have a three-layer structure of robot, the first layer for mounting wheels, motors and their controllers, the second layers for micro-controllers, and the third layer for bottle storage. Since we want to collect as much bottles as possible, we decided to design a large robot with dimensions of 400 mm x 400 mm.

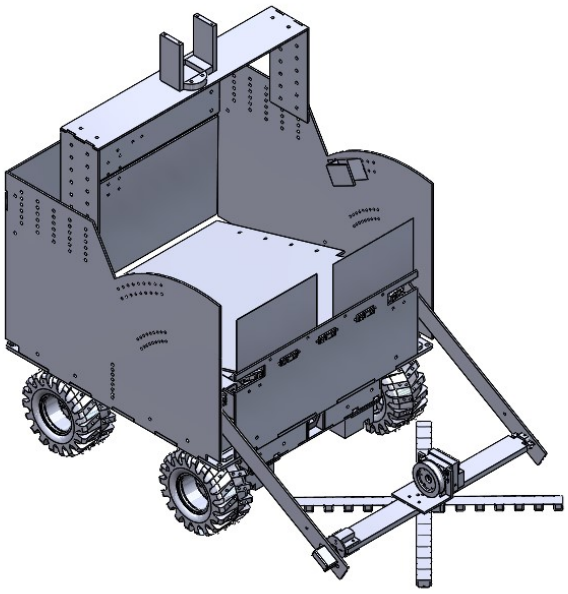


Figure 3.1: Overall design of robot

We used Arduino and Raspberry Pi to control the robot. All the actuators and the sensors except for the cameras are controlled by Arduino, and Raspberry Pi deals with the "high-level" things, as well as the image processing.

3.2. Locomotion

We made a square robot to have a large storage. And with four active wheels, we made it almost-omnidirectional to facilitate the later parts of the obstacle avoidance and possibly the path planning.

3.2.1. Wheels and configuration

Considering the rock area, we decided to have large wheels to increase the height of robot base. To avoid complex transmission system between motors and wheels, we decided to use direct-drive and put the motors at the most bottom of robot. Four WildTumper 120x60 mm wheels were used and connected to motors by wheel adaptors. The final distance between the base of robot and ground is around 8 cm. We only had two wheel adaptors, so another two wheel adaptors was machined by ourselves. The detailed drawing of wheel adaptors is in Appendix. To protect the robot and help it to turn when near the wall, four small pulleys were designed and assembled to the corners of robot. The pulleys can rotate freely when the corner of robot touch anything (obstacle or wall) and avoid any damage to the body of robot.

The wheel spacing is 26.5cm,

3.2.2. Motors

We used 4 Maxon motors to drive all of the wheels. Thanks to the ESCON servo controller board and the software ESCON Studio, we are able to set the controlling settings through graphic UI.

Input: Rotation Control To have a reliable control, we choose to use PWM wave to control the goal motor speed and two more digital pins to control the enabling and the direction. The on-board close-loop control will try to rotate the motor at the speed we give. Specifically, we have a setting that will map the pulse width percentage between 10% to 90% will to rotational speed of 0 to 7309 rpm(corresponds to 76.5cms which is 0.3cms $\times$ 255, making it more convenient for speed calculation). The direction control for motors on the left was set to be opposite to the motors on the right, saving the trouble for the subsequent hardware control to decide the rotating directions for motors on different sides.

Output: Speed Monitor One analog output was set to provide real time motor speed for the speed monitoring. Specifically, the current speed in rpm(revolution per minute) will be mapped to 0 to 4 volts. When it is rotating backwards at the highest speed(10000rpm), the analog pin will output 0V and 4V when it's rotating forwards at the highest speed. In theory it should output 2V when the wheel is still. But actually, the value read from Arduino is fluctuating between 405 to 417(roughly corresponds to 1.978 and 2.036V if the 5V level on Arduino is exactly 5V). So for a better displacement monitoring, we applied a filter to remove the noise when the motor should be still(i.e. disable the motor or set the pulse width to 0).

2-Channel control Since the motion of the front wheel should always be the same as the hind one on the same side, we connected the controlling cables on the same side together. As for the speed monitoring, we use the average value as the current speed on one side.

3.2.3. Motion Control

To have a trustworthy robot, we made a motor controlling architecture in Arduino code featured with a timeout. So if it go crazy in deed, it would be likely to stop after a while. Just as playing with the remote

control toy cars, when you release the button, the car will stop. Without further motion commands, the controller will stop the motion when timeout reached.

We also set a displacement monitor, which is useful for the localization by odometry and also to travel a precise distance. The monitor is realized by sampling the current speed at high frequency(100Hz in the final setting) and integration with respect to the time Eq. (3.1). In the real tests, the displacement monitor performed really well and have a error less than 2cm for a displacement of 50cm .

$$displacement = \sum(speed\ value \times \frac{1}{frequency}) \tag{3.1}$$

3.2.4. Protections

Since the motors are important and also vulnerable, a motor holder was designed and 3D printed to protect the motor and motor cable and assemble the motor to the base layer of robot. Also the Maxon ESCON controller boards and ESCON motherboards are vulnerable, plastic washers were added when assemble them to the base layer of robot to avoid any scratch or short circuit by metal fasteners. On the top of ESCON motherboards, a transparent plate were designed and assembled by plastic pillars to protect them and keep them clean.

3.3. Obstacle avoidance

3.3.1. Configuration

There are two type of the obstacle we want to avoid, the bricks and the wall. Both of them are higher than the bottles which we don't want to avoid. So we install the range sensors at 25cm high. They are all horizontally mounted to detect the obstacle in front. However, the slope will also be detected. But it doesn't matter too much since we decide not to go to the platform. In contrast, it will help to stop the robot to go up.

Since our robot can rotate in place, we only need to detect the obstacle in front. However, the existence of the two arms hinders us to put the range sensors close to the edge. Therefore, the two side sensors are tilted 15° outward.

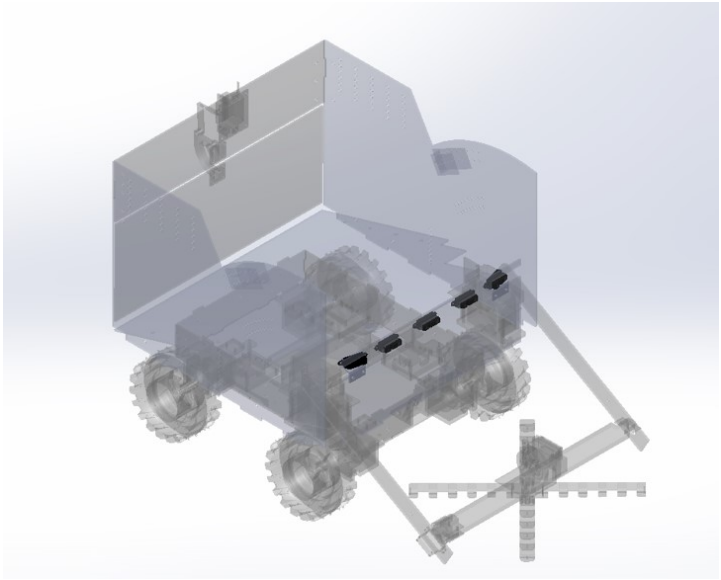


Figure 3.2: Configuration of the IR sensors

3.3.2. Infrared Range sensors

We use the infrared range sensors to detect the objects in front. They manage to detect the obstacle with different facing angles whereas the ultrasonic sensors fail. But the most fatal drawback of the infrared sensors is that the measure results are too noisy.

We applied a 25-unit long median filter to remove the noise, and then use another formula from SharpIR library(Eq. (3.2)), rather than the one in datasheet to estimate the distance from the object.

$$distance = 29.988 \times volt^{-1.173} \tag{3.2}$$

Finally, it's reliable to detect the obstacle in 50cm and the noise will not drop below 75cm when the obstacle is far away.

3.3.3. Avoidance Strategy

The avoidance is achieved on Arduino. It should automatically stop moving forward when the obstacle is too close in front, which is called stop threshold in our code. To avoid the obstacle, the robot will rotate in one direction which can be assigned by command, until no more obstacle can be seen. To have bigger clearance from the obstacle, the robot will rotate a bit more(specifically, 4° in final settings) even it can't see obstacles any more. To avoid get stuck, it will also go forward a little bit(which is 10 cm in our final settings) after avoidance. Therefore, even if the direction of the avoidance and the direction towards the goal waypoint conflict, it will get rid of the obstacle ultimately as shown in Figure 3.3.

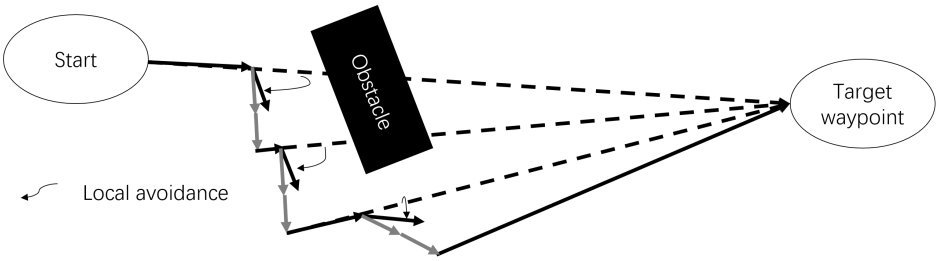


Figure 3.3: Avoidance strategy illustration

To make it safe for the gripper to reach out, there should be no obstacle in 30cm in front. So we set the avoid distance threshold to 30cm. And in case some dangerous motions are indeed needed, the stop threshold is set to 20cm.

3.4. Localisation

As stated in Section 2.4, to localize the robot, we chose to mainly rely on the beacons which can provide absolute position and orientation. If it doesn't work, the odometry can help for a while.

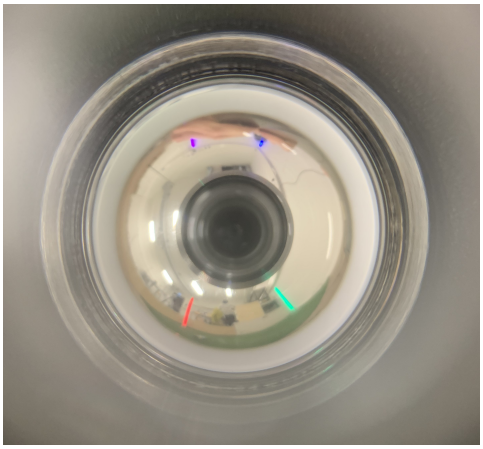
3.4.1. Localization using the beacons

Beacons' detection

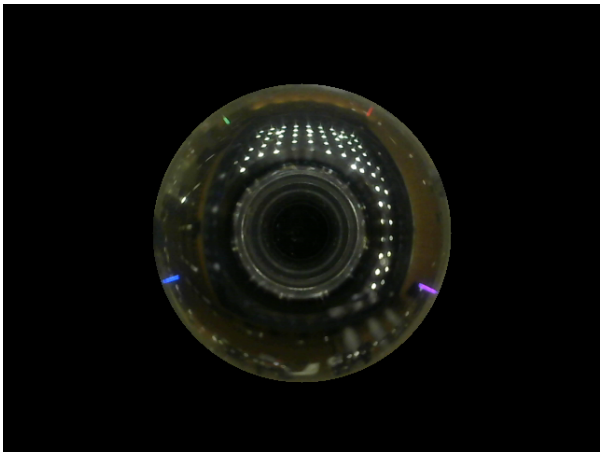
In order to properly localize the robot, we need to detect the four LED beacons, one at each corner of the arena and perform some calculations to deduce the robot's position and orientation. To detect all the beacons at the same time whatever the situation is, a camera looking up along with a 360° mirror are used (Figure 3.4). A 3D printed mechanical part is then designed to hold both the camera and the mirror together and avoid them to move during the robot's motion. The detection of the beacons is done on Raspberry Pi and the camera is connected to it by a simple USB cable.

Figure 3.4: System for beacon's detection and mechanical part holding the camera and 360° mirror

To have a better detection of the beacons, an exposure time is applied to make the beacons more distinguishable from the background. The value varies from the drivers of different devices. Besides, when moving to the actual competition arena, we noticed that the camera was also detecting the reflexion of the room's light on the wall and was disturbing the detection of the beacons. Therefore, in order to avoid any interference, a circular mask is applied to black out everything outside of it (Figure 3.5b).



(a) Beacons' detection without any exposure filter



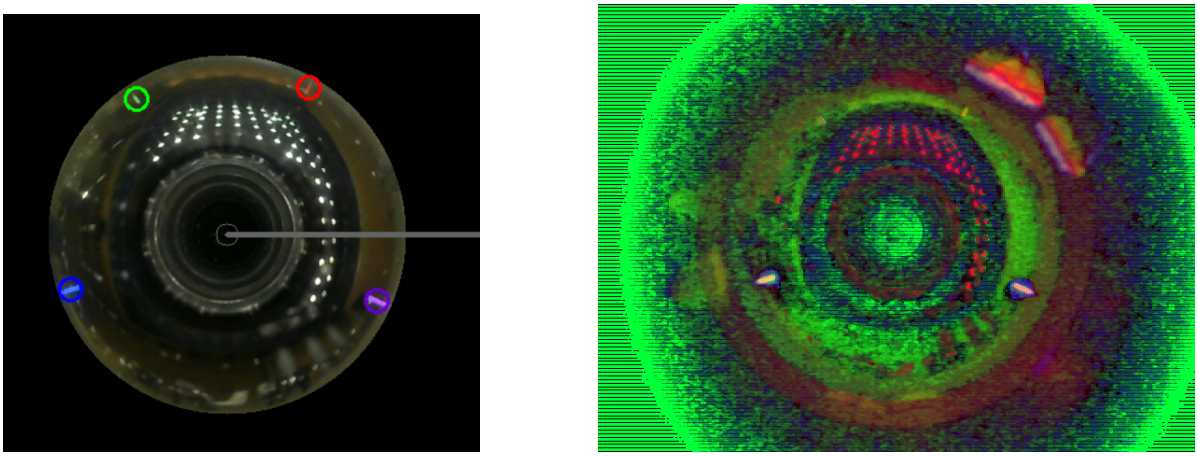
(b) Beacons' detection with an exposure filter of 70 on RPi and circular mask

Figure 3.5: Comparison of the beacon's detection with and without exposure filter

In order to localize the robot, each beacon should be extracted separately. To do so, the image taken by the camera is converted into an HSV space and two thresholds are applied on all the image. These thresholds are the maximum and minimum HSV values accepted. By doing so, we can isolate a specific LED color with a specific saturation and brightness. The HSV values are helpful to easily find these thresholds as they are independent parameters unlike the RGB values. In order to find these thresholds, a GUI program in python [1] is used and allows us to modify the thresholds and see their effects on the image in real time. Several experiments have been done at different positions of the arena to find the most suitable HSV thresholds(Table 3.1). An extraction example is shown in Figure 3.6, where the gray line stands for the front direction of the robot. And the result of the chosen thresholds is shown in Figure 3.7.

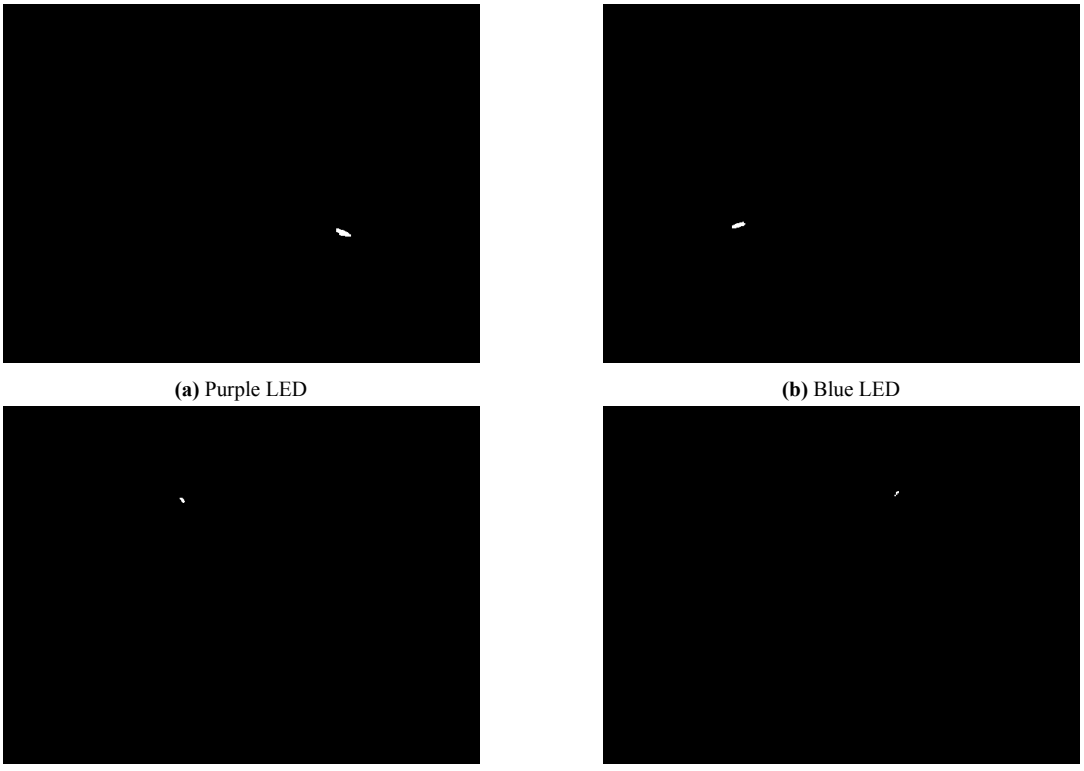
Color	Lower Bound	Higher Bound
BLUE	77, 148, 148	120, 255, 255
GREEN	49, 46, 116	88, 255, 255
RED	0, 160, 0	7, 255, 255
PURPLE	128, 114, 77	162, 255, 255

Table 3.1: HSV thresholds for the beacons



(a) **(b)**

Figure 3.6: Beacon's extraction example and its HSV representation



(a) Purple LED **(b)** Blue LED

(c) Green LED **(d)** Red LED

Figure 3.7: Extraction of each beacon separately

Once each of the beacons has been detected, their position on the image is obtained as the center of mass of the extracted pixel cluster. With each of those centers, the vector which goes from the center of the mirror to the point is constructed. Then, by some simple dot and cross products between the vectors, it gives the angles between the beacons and the angles between the beacons and the front of the robot. These information are then helpful to find the actual robot's position and orientation.

Position

With the information deduced from the method described above, the current position of the robot is determined by triangulation using the ToTal algorithm [2]. This algorithm has been chosen as it is known as the fastest to compute and was giving accurate results in most of our cases without too many computations. This algorithm actually uses the positions of 3 beacons to localize the robot. So if one beacon can't be seen, the robot can still identify its position. If all the beacons can be seen, we just use the first three. The details to find the robot's position are not stated here as they can easily be found on the ToTal algorithm research paper [2] but the basic idea is that the robot's position corresponds to the intersection of three circles each one passing by at least two beacons and which parameters depend on the beacons' positions and angle between them. It takes about 200ms on Raspberry Pi to grab a frame and calculate the current position with this algorithm. The result for the example in Figure 3.6 is (354.6, 156.6), where for a sake of convenience the x-y position is stated in cm.

Orientation

Based on the current position and the relative angle to the beacons, the orientation of the robot is calculated. As shown in Figure 3.8, for any beacon, the relative angle from front vector to the beacon, noted as α_i, can be acquired from the image, and the angle from the beacon to zero heading direction at the current position, noted as β_i, can be calculated by their positions. Therefore, the current orientation can be calculated as Eq. (3.3).

$$\theta = -(\alpha_i + \beta_i), i = 1, 2, 3, 4 \tag{3.3}$$

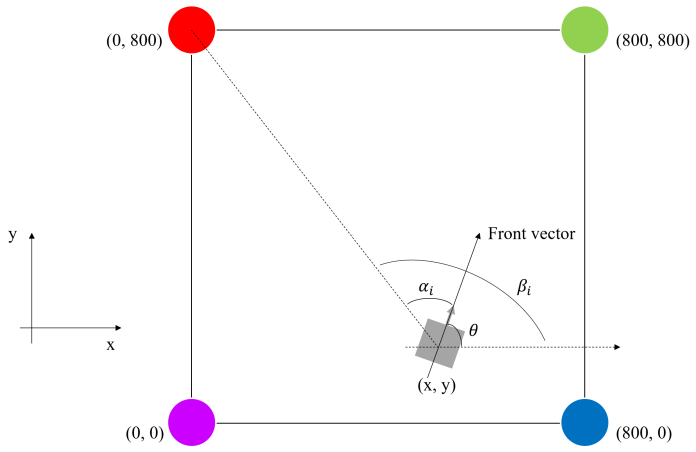


Figure 3.8: Orientation Calculation (Beacons' position in cm

All the beacons should give similar results. However, sometimes (e.g. when the robot is too close to one beacon) the color filtering can go wrong. So we use the median of the results as the estimation of the orientation.

Results

Overall, using the beacons' detection to estimate the position of the robot and its orientation gives accurate results in most of the cases and it is an elegant solution to solve the localization problem

without too much complexity. As a result, by tuning properly the HSV thresholds to detect each beacon separately, we managed to obtain a robot position with an error of maximum 20 cm on the x and y axis which can be considered as accurate enough for an arena of 8x8 m. Besides, most of the time, especially in the center of the arena, this error was even not exceeding 5 cm which is excellent for such a method. Regarding to the orientation, we also managed to achieve accurate results by determining the angle the robot makes with the x axis with an error always less than 10°. The only issue we faced with this localization method was when the robot was getting closer to one of the LED. Indeed, in this situation, the closer LED was often too bright to detect the other beacons or the thresholds weren't enough to delete it completely. That's why we decided to use odometry for a short period of time when this case was happening. Another issue that we've been facing is that the red LED wasn't bright enough and it was hard to detect it with the camera when the robot wasn't close to it. Hopefully, that was the only beacon having this problem, therefore, having an algorithm that works with only three beacons solved the problem.

3.4.2. Odometry

As explain the previous content, the odometry method serves as a backup. It's fast to provide a pose but will lose accuracy when travel too long from the starting point. So we only call it when the beacon localization doesn't work.

Eq. (3.4) shows the algorithm we used to calculate the new pose $(x_{i+1}, y_{i+1}, \theta_{i+1})$ from the last one. l_i and r_i are the displacement of the left and right wheels; b is the wheel spacing, which is 28cm estimated by experiments.

$$\begin{aligned} p_i &= (l_i + r_i)/2 \\ a_i &= 2 \cdot (r_i - l_i)/b \\ x_{i+1} &= x_i + p_i \cdot cos(\theta_i + a_i/2) \\ y_{i+1} &= y_i + p_i \cdot cos(\theta_i + a_i/2) \\ \theta_{i+1} &= \theta_i + a_i \end{aligned} \tag{3.4}$$

The calculation is done on Raspberry Pi. Arduino will keep reporting the displacement information at a frequency of about 10Hz, and it will be stored in cache until next valid pose comes. Therefore, if the beacon localization doesn't work, the robot can still use the odometry to infer where it is for a short period of time. When the beacon's detection algorithm works again, the odometry is updated to reduce the accumulated error when used again.

3.5. Bottle detection

3.5.1. Camera

The Raspberry Pi camera was fixed on the base layer of robot and used to detect the bottles. The robot base is too high for the camera to capture the lying bottles right in front, so we tilt the camera 20 degrees towards the ground (Figure 3.9) at the cost of the higher possibility of missing the standing bottles in distance, which actually doesn't matter a lot. A camera holder was designed and 3D printed for assembling the camera.

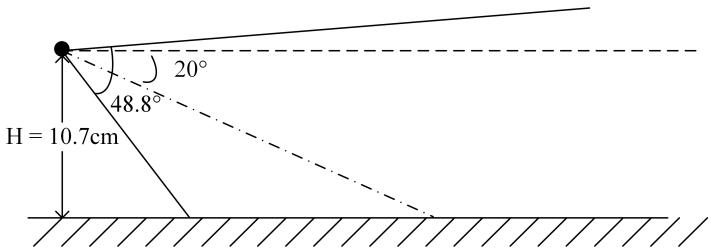


Figure 3.9: The tilting angle of the camera

3.5.2. Object detection

Model There are lots of different models for object detection. In the end, we decide to use EfficientDet[3], which is only of 4.4MB. The two main reasons to use it are that 1) it's fast. it's designed for the edge devices like Raspberry Pi; 2) the tutorials about how to use and train our own data are detailed, so that we don't need to pay much effort to have a model for our data and use it.

Data The most training data is 121 640-by-480 images filmed by Raspberry Pi camera when the robot moving in the arena(so some bottles with motion blur), and labeled with bottles and bricks[1]. Before the competition, we added 30 photos taken by cell phone of the bottles will be used. They are labeled by labelImg, and the annotations are stored in PascalVOC format.

Training We used the provided colab notebook[2] from tensorflow and our own dataset to train our model. In most cases, the model can recognize with high confidence(more than 0.5) the bottles that is big enough, which lie no more than around 2 meters away from the camera.

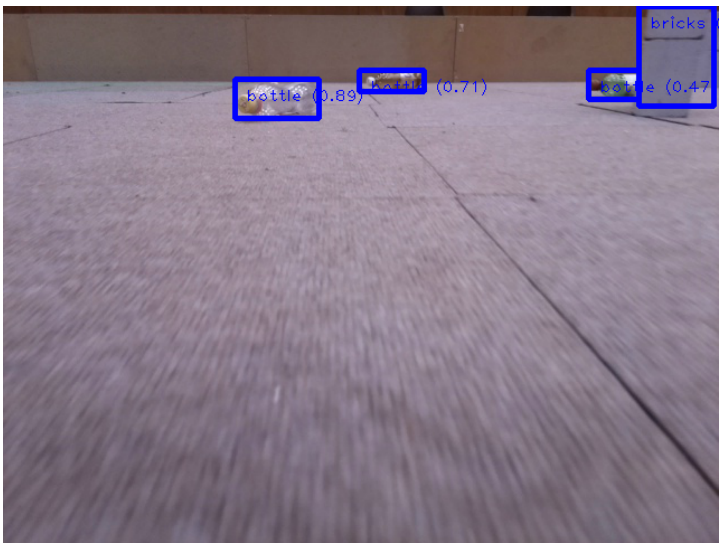


Figure 3.10: Bottle detection validation sample

Detection We followed the instruction from this repository to set up the environment. In test, one detection step on a 640x480 image cost about 200ms on Raspberry Pi. We failed to get a Coral USB accelerator but 200ms is already acceptable for our project.

[1]The label "bricks" wasn't been used in the following navigation part. We just want the label "bottle" not to be so lonely.

[2]Due to the very rapid update frequency of the libraries for tensorflow, this old version notebook actually doesn't work currently. The available one with correct libraries can be found in the attached source files

3.5.3. Relative Position

To calculate the relative position of the bottle from the camera, a perspective transformation matrix was determined by experiments on chess images, so that any pixels on the image can be mapped to the approximate relative position on the ground. We use the middle pixel of the detection box's bottom line to estimate the position of the bottle.

3.6. Bottle collection

The bottle collection system includes soft gripper, hand and two arms as shown in Figure 3.11.

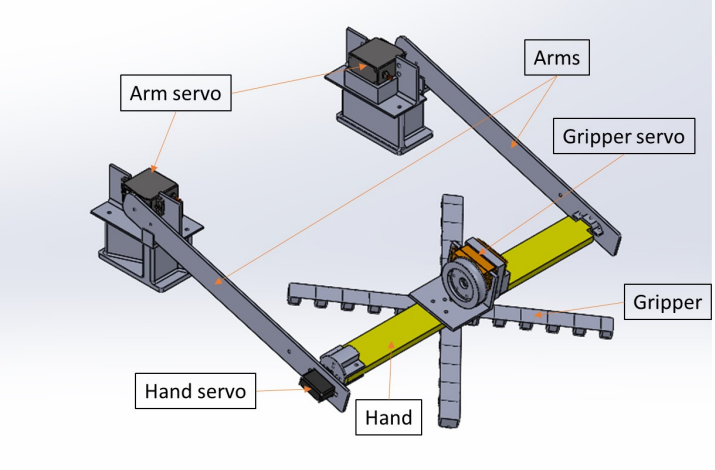


Figure 3.11: Bottle collection system

3.6.1. Gripper

The whole gripper was 3D printed and it has four fingers and a central part connecting all of them. Each finger has a 0.4 mm flexible layer and five segments on it. A cable (fish line) passes through all the segments of the finger and connects to a pulley. The thickness of the flexible layer was tested to find the most proper parameters of 3D printing. However, 3D printed finger has low friction and the bottle easily escape from gripper. To increase the friction, we found a foam material and stick on the surface of segments. This increase the success rate of grasping bottles.

The four fingers are attached to the central part by a mortise-and-tenon like connection and four cables run through the central part and out the top. Then four cables were fixed on a pulley driven by a strong servo. Four fingers will close together and release together as well. The servo is fixed on the hand.

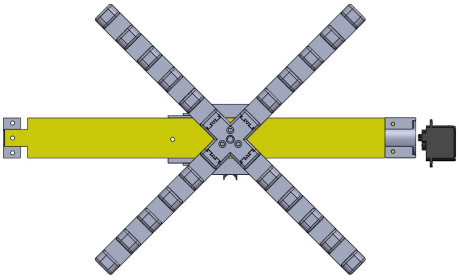


Figure 3.12: The connection between central part and fingers

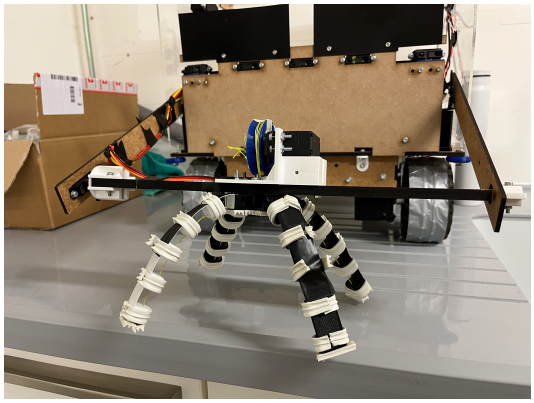


Figure 3.13: Final gripper with four fingers and foam material on each segments

3.6.2. Hand and the arms

To grasp the bottle and to release the bottle require different orientations of gripper, therefore, a small servo is added at the joint between hand and arms to change the orientation of the gripper. The other end of the hand pass the arm and clamped, so this end of hand can rotate freely. Two arms were powered by two strong servos to lift the gripper and put down the gripper together.

Arm servo holders and supporters were design to fix the servos and increase the height of arms. When the arms were put down, a part of the servo holder will support the arm to reduce the torque required from the servos. Similarly, two arm supporters were assembled on the bottle storage wall to support the arms on the lifting position.

3.6.3. Servos

We used simple PWM controlled servos for all the tasks. All the servos are separately calibrated for a precise positional control.

Previously we used the official servo library to generate PWM waves, but it conflicts with the motion control module[3]. So we turned to a servo board with IIC communication to avoid the possible conflicts.

It should be continuous of the position commands for a smooth servo movement. However, due to the unbalanced mass distribution of the Hand, the speeds of the left arm servo and the right have slight differences. To avoid doing any possible damage to the hand and also the servos, the two servos move in steps – they will rotate a little angle and wait for a while to ensure the synchronized actions and then another rotation cycle.

The whole grasping movement goes like this.

1. rotate the arm to the neutral position, which is perpendicular to the ground; rotate the hand preparing to grasp.
2. rotate the arm down and wait a while ensuring the hand is at the lowest position.
3. rotate the pulley tightening the cable and bending the fingers to grasp, and wait for a while ensuring the bottle is grasped tightly
4. move the arm back to initial position and then release the gripper, wait a while until the gripper fully released

[3]Not sure about it. The servos will have unwanted movements one by one periodically if motion control module enabled at the same time. We found this problem quite late so decided not to dig into it. The possible reason can be from the timers, since we also use the TimerThree library for monitoring the real-time speed.

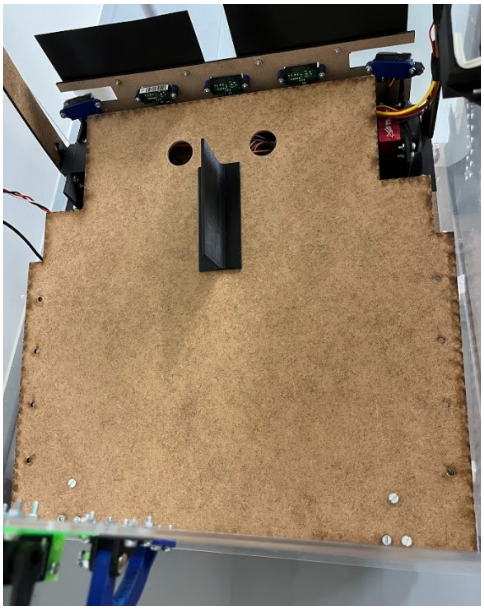


Figure 3.14: 15° plate with middle rib, and the thin flexible sheets on the front

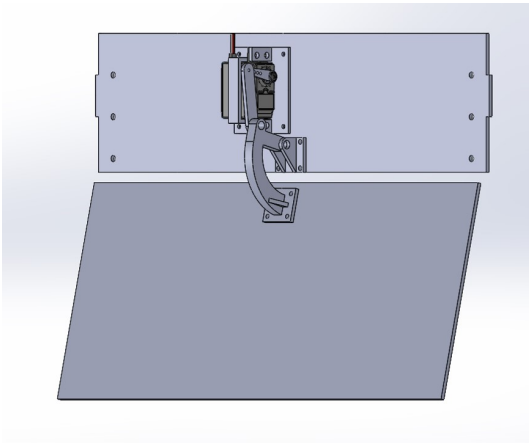


Figure 3.15: Bottle release system CAD design

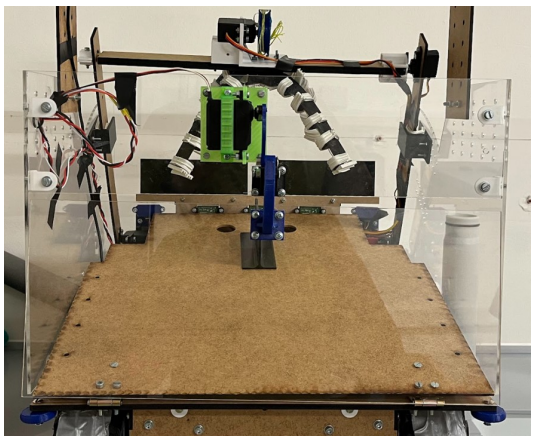


Figure 3.16: Bottle release system on the real robot

3.7. Storage

3.7.1. Storage space

The bottle storage space is above the third layer of robot. The collected bottle will place on a plate with 15°, the height of side wall is 28 cm to have a large storage space. The 15° plate is assembled to the second layer by two hinges and supported by two L-brackets on the side walls, so that the plate can be opened to operate the electrical hardware of the second layer. In front of the storage space, two flexible sheets were attached to prevent the bottles from falling out without interfering with the movement of the gripper. The release of the bottle is carried out by opening the back door shown in Figure 3.15 and Figure 3.16. A servo will rotate a connection bar and lift the back door by two hinges. To avoid the collected bottles getting stuck in the storage space together, a middle rib was attached to the plate forcing them to choose one side.

The servo is not very strong. It's better to have a smaller storage capacity rather than losing the collected bottles in the arena, so we set a relative small storage capacity, – 7, in the final competition.

3.7.2. Empty the storage

We set the recycling zone waypoint to (75cm, 75cm), and ask the robot to head to 45° before it open the back gate, so that the bottles released are likely to fail in the correct zone.

Since the open angle of the back gate is really small, merely one-bottle wide, some extra procedures designed to release the bottles. Specifically, the robot will move forward and backward for three times, providing some acceleration facilitating the bottles to drop.

3.8. Micro-controllers

We use Arduino Mega 2560 along with the Arduóne board to control all the sensors and actuators, namely, the range sensors, servos and the motors. Raspberry Pi drives two cameras for bottle detection and beacon localization respectively.

3.8.1. Arduino

We made Arduino not to remember anything. It does what Raspberry Pi tells it faithfully except for running towards the obstacle in front.

There are mainly four modules on Arduino: motors and motion control, servos, range sensors and local avoidance, and the communication with the Raspberry Pi. The first three modules have been introduced in the previous sections and the communication module will be presented in the later subsection.

In the main loop, it checks the distance of obstacles in front deciding whether to push the emergency stop button or report it, tracks the displacement and synchronizes the motion command, reads the commands from Raspberry Pi, executes them and reports when they are done. Without reading the commands(whose length varies leading to variable processing time), one loop cost around 20ms, most of which spent on reading values from the range sensors.

3.8.2. Raspberry Pi

Though it's more efficient to use Lite OS without graphical interface, we chose to use the Raspberry Pi OS with desktop as a easier start. After all, we have two cameras to check their outputs.

The programs run on Raspberry Pi will be explained in Section 3.10.

3.8.3. Communication and the Protocol

Arduino and Raspberry Pi communicate via serial. A serial cable connects their USB ports. The existence of the dtr wire enables Raspberry Pi to restart the Arduino without extra code on Arduino nor extra cable connected to Arduino's RTS pin.

The messages are encoded as byte arrays to shorten the time for communication. The byte arrays are composed of a header byte and several or none data bytes. Byte '0x00' is used as the separator and in the encoding system we are going to present later it is avoid to be used inside the message.

Header

As shown in Figure 3.17, first two bits indicate the length of the following data bytes. The length information provides the possibility for both of them to check the integrity of the message. The third to fifth bits stands for the command or report types. And the last 3 bits provide extra information about the command or the report. Table 3.2 and Table 3.3 show the specific value set for the protocol. For Motion command, the sixth bit indicates the motion command's mode, and the last two bits encode the direction. The stared commands and reports are not used in the final control strategy.

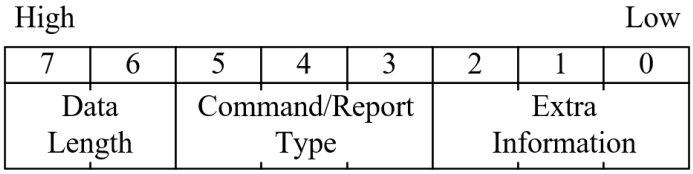


Figure 3.17: Protocol Header

Command Type		Extra Information		Extra Information 2	
Value	Command	Value	Info.	Value	Info.
0	Motion	0	Speed Mode	0 1 2 3	Backward Left Right Forward
		1	Distance Mode		
1	Servo Motion	0	Pick up		
		1	Empty Storage		
2	Advanced Functions	0	Local Avoidance towards Left		
		1	Local Avoidance towards Right		
3	*Request Information	0	Orientation		
		1	Declination		
4	*Direct Motor Control				

Table 3.2: Protocol header's command types with the values

Report Type		Extra Information	
Value	Command	Value	Info.
0	Warning of obstacles	x	the index of the sensor reporting the closest distance from the obstacle
1	Displacements		
2	Distances from the obstacle in front	x	the index of the sensor reporting the closest distance from the obstacle
3	*Distances from the obstacles in back	x	the index of the sensor reporting the closest distance from the obstacle
4	Tasks done	0	Pick up
		1	Empty Storage
		2	Local Avoidance

Table 3.3: Protocol header's report types with the values

Data bytes

For motion commands, the following first byte stands for the command value, which is either the speed value(0-255) or the distance in cm. The second byte stands for the timeout(unit: 100ms) of this command(The timeout can be omitted. And the Arduino will use a default value for this command).

The displacement value uses cm as the unit. The value from -127 to 127 will be mapped to 1 to 255

to avoid '0x00' being in the message.

3.9. Electrical Design

3.9.1. Connection

Most hardware components used in our robot and the connection between them are shown in Figure 3.18.

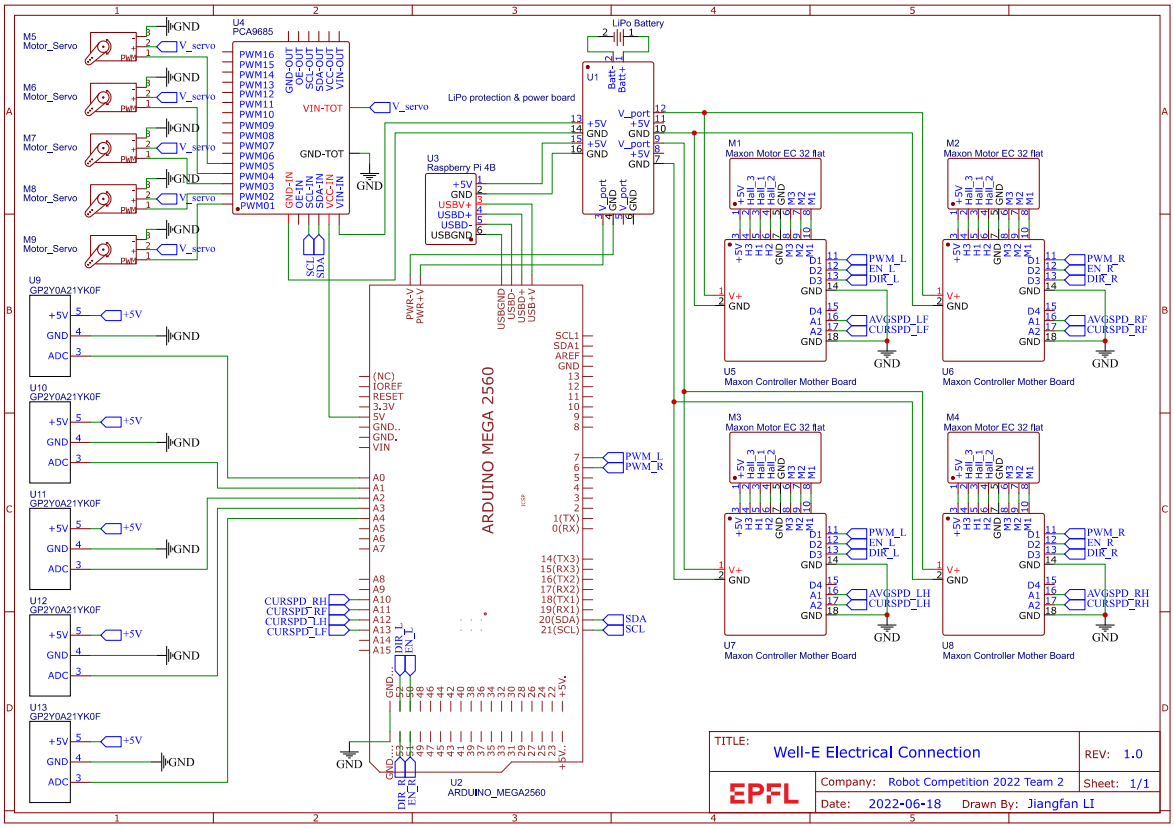


Figure 3.18: Wiring diagram

3.9.2. Power

Our battery capacity is 4000mAh, and the fuse's tolerance is 20A, which mean that, before the fuse begin to blow, the fully charged battery can support for at least 4000mAh / 20A = 0.5h = 30min, which is much longer than the competition time.

The power consumption of the hardware is listed in Table 3.4. As the robot will never use the servos and the motors together, and the battery is capable to provide power at 88.8W for 30 minutes, there's no doubt that our robot is able to run for at least 30 minutes before it runs out of the fully charged battery.

3.10. Navigation

3.10.1. Motion Control

Speed The nominal speed of the motor is 2060rpm at 9V. So the nominal speed at nominal voltage of the battery, i.e. 11.1V[4] should be 2617 rpm, which is 27.4cm/s. Higher speed means more bottles. And it's less likely that the robot will keep moving for a long time, since the arena is a 8-by-8-meter square

[4]But actually it's always higher before running out of power

Component	Operation Voltage(V)	Typical Current(mA)	Nominal Power (W)
Arduino	5	10	0.05
Raspberry Pi	5	1280[a]	6.4
Raspberry Pi Camera	3.3	250[b]	0.825
Camera Logitech C505e	5	500	2.5
Range sensor	5	30	0.15×5
Sum of the above			10.525
Maxon Motor	11.1	1060	11.766×4
Sum of the Motors			47.064
DFRobot DSS-M15S	5	1800[c]	9×3
Parallax Standard Servo	5	140	0.7
HS-82MG Micro Servo	5	1800[d]	9
Sum of the Servos			36.7

[a] power consumption under 400% CPU load (stress –cpu 4) from Raspberry Pi Dramble.
[b] Source
[c] Stall current
[d] Stall current

Table 3.4: Power Specification

with obstacles. Therefore, we set a little bit higher nominal speed of 35cm/s for the robot. For motions requiring precise travelling distance, e.g. approaching to one bottle, we use a slower speed of 15cm/s so that it will not glide too much after braking.

Turning parameter In theory, the factor between the angle turned and the difference of the displacements of left and right wheel should be the wheel spacing. However, there always be frictions and slippage. So we did several experiments to get the 'real' wheel spacing, which is 28cm.

3.10.2. Main Control Loop

The robot's finite state machine counts four main parts that the robot has to handle during its motion. It is actually, path following, obstacle avoidance, bottle detection/collection, go back to the recycling zone. The whole diagram are shown in Figure 3.19.

Let's first tackle the path planning part. As it has been explained in Section 2.2, we decided to go with a simple solution for the navigation strategy by defining the path planning as a list of way points that can be updated along the motion. The idea is thus to set a predefined path and to let the robot update it if it sees a bottle on its way or if the storage is full and it has to go back to the recycling zone. The initial path has been designed as in Eq. (3.5) and is shown on Figure 3.20, where the positions of the way points are in cm. It is designed to get the bottles in the zone 3 with high points first.

$$\begin{aligned}\text{INIT\_PATH} = [&[75, 300], [75, 725], [400, 725], [400, 500],\\ &[700, 400], [700, 100], [300, 100], [400, 400], [500, 300]]\end{aligned} \tag{3.5}$$

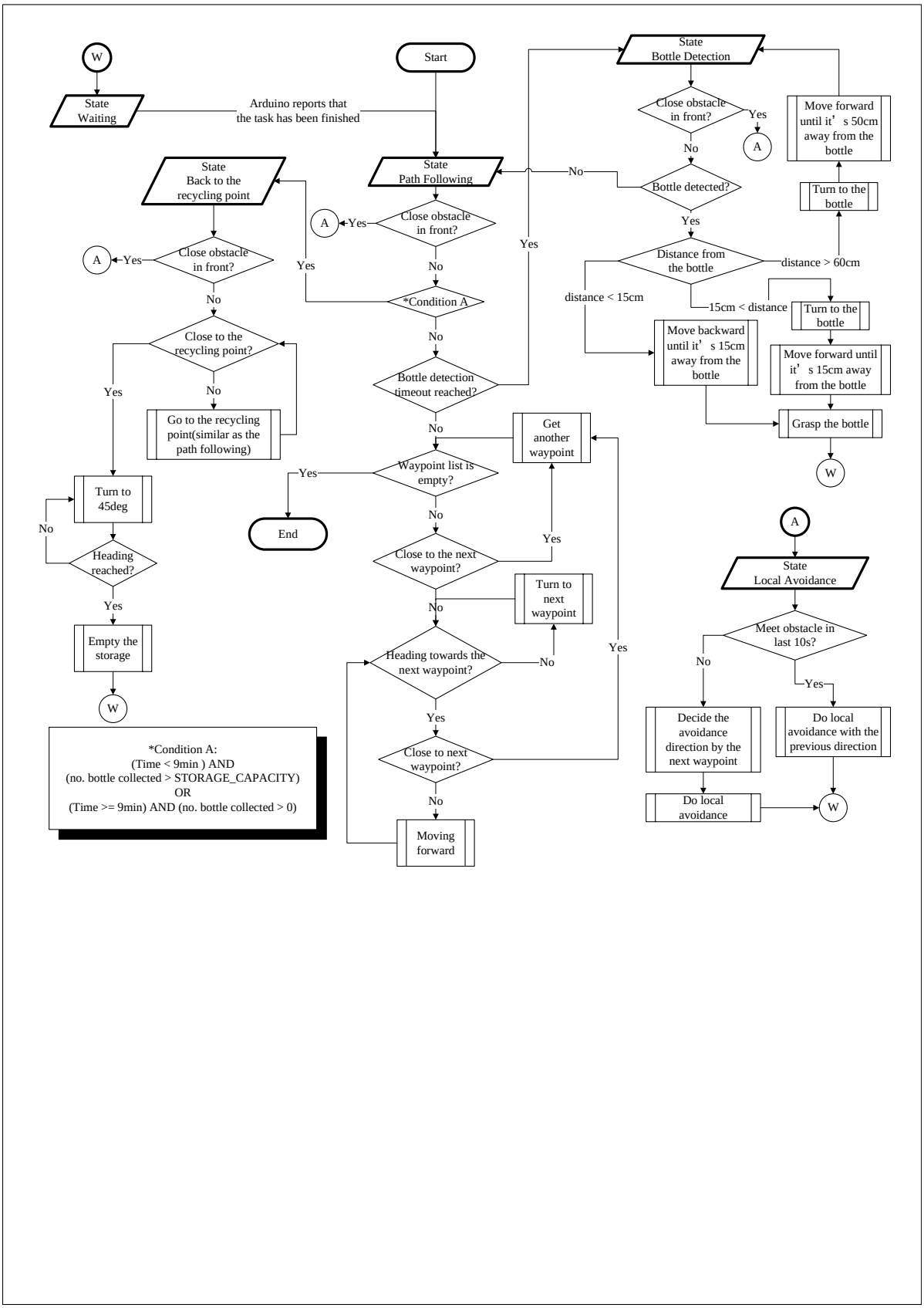


Figure 3.19: Main loop for the bottle collection strategy

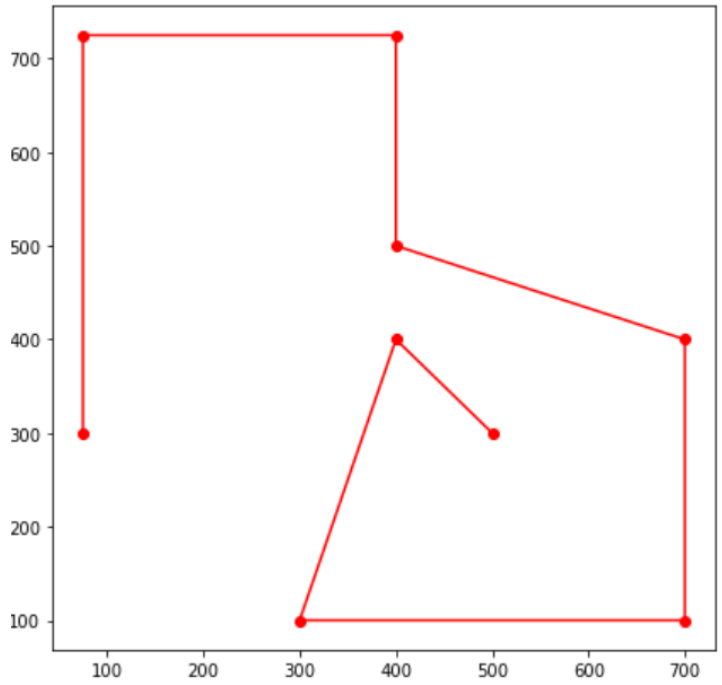


Figure 3.20: Initial path the robot was planned to go through

This path has been chosen a bit at the last minute first of all because we figured out that it was too risky to go on the platform and we never tested our robot there. Therefore, a wise choice is to avoid that zone. Besides, we decided to set the way points sufficiently far away from each other because the robot will lose some time each time it has to align itself with a target. Thus, in order to reduce the amount of time spent on following the path we decided to set targets far away from each other. Also, after some tests, knowing that the robot would probably never finish all this path(actually in the final competition, it just merely finished the first one in 10 minutes), we decided to not create any other path and let the robot follow the initial path for 10 minutes. That was a risky choice but as expected the robot didn't face any issue with that. Furthermore, as the position of the robot is not perfect and there is potentially an obstacle on the goal, we set the error on the goal at 50 cm. Thus, when the robot detects that it is closer than 50 cm from the goal, it switches to the next way point. Once the goal is achieved, it is removed from the way points' list and the next target is the next way point on list. In other words, the goal is always the first way point in the list.

The update of the path happens in two cases. The robot will detect if there's a bottle in front every 5 seconds. If the robot detects a bottle on its way, it set a new way point close to the detected bottle. Thus, the robot heads towards it then approaches it progressively. If the bottle is detected further than 50 cm away from the robot, then it approaches the bottle until reducing this distance to 15 cm. That's the distance from the bottle for which the robot can collect it perfectly. Otherwise, if the bottle is detected too close, the robot goes backwards to keep this minimum grasping distance of 15 cm, assuming there will not be any obstacles behind which is of high possibility. Once the bottle has been collected, the robot goes back to its initial path following and heads again towards its previous way point. Since we set a short detection interval less than the time of the collection motion, if the robot fails to collect the bottle, it will find it again and try to collect it again.

The other case where the path is updated is when the storage is full. Even if our robot's storage is big enough, we decided to set its "maximum capacity" to 7 collected bottles. Thus, after 7 trials to get a bottle, the robot is forced to go to the recycling zone and empty its storage. Note that the bottle's detection is disabled when the robot is on its way back to the recycling zone. This "maximum capacity" has been set to 7 so that we can at least insure some points quickly. Also, since the robot has no idea

whether if it has actually collected a bottle or missed it when grasping, it just goes back to the recycling zone after 7 trials to collect the bottle. When the timing is about to end, specifically, after 9 minutes past, the robot will go back after every collection trial.

Finally, in order to ensure that the robot achieves its desired way point, it keeps estimating the remaining distance to the goal and the angle to it, and do the necessary corrections if it deviates more than a given threshold that we set to 10° in our case.

4

Challenges Encountered and Solutions

4.1. Motion

4.1.1. Motors

We ordered four 34:1 DC motors(HP) at the beginning, but we only had one motor driver with two channels. And there was no suitable motor driver who can endure such high currents(6A) available online. So we changed the configuration to two 34:1 DC motors for two front wheels and used two castor wheel for two rear wheels. However, after testing the motors, this configuration was not powerful to move over the rock area (actually, the rock in MED testing room are much larger than the rock in the final arena) and the slope of raised platform. We had to change the motors to more powerful motors, in the end, we used four Maxon EC 32 flat (15 W) motors with 1:60 gearbox.

4.1.2. Displacement

During our tests, the displacement monitor may break after long time of use. We don't know what caused it. Since this situation is not that often and mostly happens after long time of use, we decided not to dig in. One conjecture is that when the battery level is low, the analog output of the Maxon controller board may drift.

4.2. Infrared Range Sensors

At first we used ultrasonic sensors for obstacles detection. The reason to choose it rather than the infrared one is that the formula is simpler and the reading value is more stable an d more precise. However, in the last week we found the ultrasonic sensor doesn't respond to the obstacle that is not exactly facing to it. In our previous tests, we always made the obstacle facing to it so we didn't find the problem. So we modified the mechanical and electrical design quickly to switch to the infrared range sensors.

One problem we meet is that the laptop power seems to fail to provide enough current for the sensors so that the reading values fluctuate drastically. Theoretically, one I/O pin on Arduino can only provide 20mA whereas the infrared range sensors' typical current is 30mA. This was solved by using the battery to provide the power.

Another problem is that the reading values are really noise which leads to "false positive obstacle", making the robot to stop time to time. So we applied a median filter of 25 units long, to have a stable value at the cost of time.

In addition, the example of distance measuring characteristic figure in the datasheet doesn't match the real respond from the bricks. So we adopt the formula from SharpIR library (Eq. (3.2)).

4.3. Gripper and arm

Gripper The initial idea was to use silicone for gripper and after some discussion we realised that silicone soft grippers would take a long time to fabricate and iterate and that silicone was too heavy to maintain the opening of gripper. Our coach suggested directly 3D printing gripper as the thin 3D printed layers can also be flexible and 3D printing is easy and quick. Thanks to coach's suggestion, we tested the thickness of flexible layers, different number of finger segments, different length of finger, different number of finger to find a better combination of gripper configuration. We found that: longer finger doesn't means larger grasping range, because the finger will bend due to its gravity; increase the number of finger will increase the ability of grasping bottle in different orientation, but increase the required torque; higher the finger segments, easier to close the finger, but need to consider the diameter of bottle. But we didn't have time to do deeper research to optimize the gripper design. In the end, we decided to using the current gripper configuration shown before. One problem with soft gripper is that it is less durable, the bending angles of the fingers increase after many tests and the gripper can easily break if it hits something hardly.

Hand At first we used 3 mm MDF as the material for the hands, then we realised that the whole gripping system (gripper and servo) is too heavy and the hands would bend significantly. We also tried 3 mm PMMA, but no improvement of the bending problem. In the end, we used 8 mm MDF (but maybe 5 mm is enough) for hand. The drawback of using thick material is increasing the weight and requiring more torque from arm servos.

Arms The two arms were controlled by two strong servos. We used parallax standard servos at first, but they are not powerful enough to lift the gripper. Then we changed to DSS-M15S Servo from DFRobot. During testing, we burnt a servo, which we think was because the torque required was too high. A better design to reduce the risk of burning the servo was to have a guide way for both arms, which will also increase the stability of arm movement. We considered to use thicker material for arms, but to avoid any damage on servo again, we kept using 3 mm MDF (breaking the arm first instead of breaking the servo). However, we did not have time to change the mechanical design. A suggestion for the future is to be careful when using servos and not to use them for high torque applications.

4.4. Back door

Intuitively, the back door mechanism appears to work, but in practice, the opening range of the back door is very small and the servos can only rotate at small angles. Due to the limitations of the back door mechanism, where some servo positions are impossible to reach without breaking the connection parts, we burned out several servos while testing the back door. Therefore, this design is not recommended. It would be simpler and easier to control by using hinges and mounting the servo directly on the door.

4.5. Bottle detection

We trained the model in the MED testing room, which works well. After the final arena built, we filmed another video of final arena by Raspberry Pi camera to train the model again. However, the version of tensorflow updated and the model we used is not supported anymore. We tried install different version of tensorflow and finally found that the model can work with following versions: tensorflow 2.8.2, tflite-model-maker 0.3.4 and scann 1.2.4.

5

Project Management

5.1. Budget management

One of the requirements of this project is that each team respects its allocated budget. It is actually made of two different parts, a virtual budget of CHF 2000 which can be used to purchase components that are available in stock and a real budget of CHF 750 to buy any additional required materials and components from suppliers.

The actual use of the real and virtual budget is stated in Appendix A.4. We finally used 96.12% of the virtual budget which is equivalent to CHF 1922. It is indeed quite a lot but these costs include lots of components that we bought in the beginning to test them and do some comparisons and that weren't used afterwards. On the other hand, we spent CHF 248.06 from the real budget which corresponds to 33.1% of the real budget. Most of these expense were for the mechanical structure of the robot. Overall, we can say that we correctly respected our budget without any over expense.

5.2. Time management

Time management has been handled using a grant-chart that we developed in the beginning of the project before doing anything else. The idea was to schedule the different steps as much as possible to have certain deadlines imposed to ourselves and avoid having to do everything at the end in the rush. This schedule is presented in the next page. Note that since this schedule has been implemented even before knowing what our robot was going to look like, all the different steps are a bit general and don't clearly specify which actual part has to be done. Besides, at some point of the project, we realised that this timeline was a bit to optimistic especially keeping one full month of testing was idealistic. Indeed, we figured out that the implementation of each function separately was taking way more time than expected. Besides, the production of the actual robot has been delayed a lot due to some delays from suppliers on the one hand but also because the three of us were busy with some projects from other classes. Overall, we present here our initial schedule but in reality it has been a bit put on the edge when we realise that it wasn't possible to follow it. Anyway, even if the timeline hasn't really been respected, we managed to control our time and have a robot ready for the real competition.

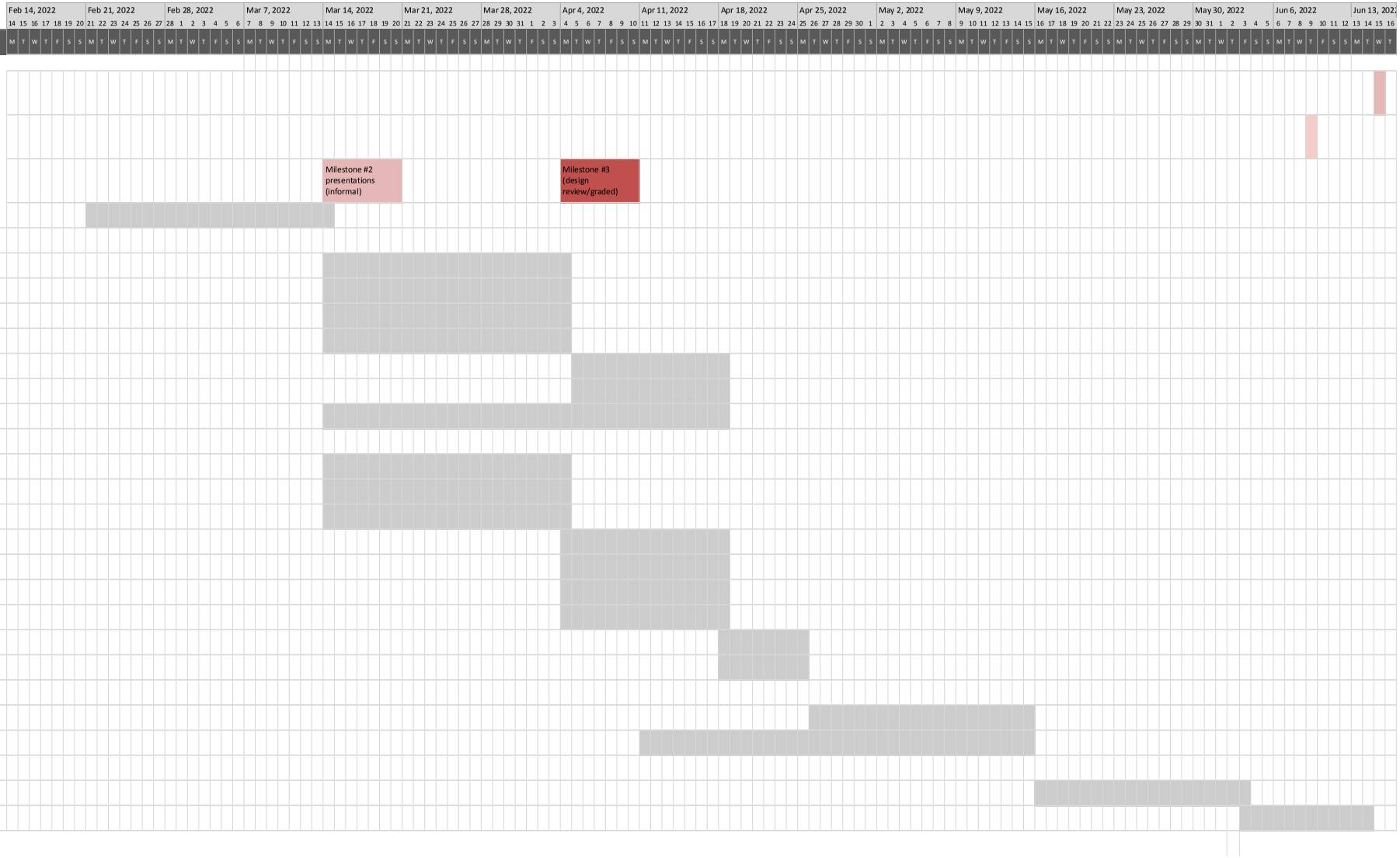
Robot competition

EPFL

Jianan Mao, Ramy Charfeddine, Jiangfan Li

Project Start: 22/02/2022

TASK	START	END
Competition	15/06/2022	15/06/2022
Rehearsal competition	09/06/2022	09/06/2022
Milestone	22/02/2022	15/06/2022
Reflection/Brainstorming	21/02/2022	14/03/2022
Function analysis implementation		
Locomotion method	14/03/2022	04/04/2022
Collection method	14/03/2022	04/04/2022
Bottle detection	14/03/2022	04/04/2022
Localization method	14/03/2022	04/04/2022
Navigation method	05/04/2022	18/04/2022
Obstacle avoidance	05/04/2022	18/04/2022
Selection of the components and testing	14/03/2022	18/04/2022
Design		
Initial design	14/03/2022	04/04/2022
Build CAD model	14/03/2022	04/04/2022
Material selection	14/03/2022	04/04/2022
PCB design(if any)	04/04/2022	18/04/2022
Verification of locomotion	04/04/2022	18/04/2022
Verification of collection	04/04/2022	18/04/2022
Verification of localization	04/04/2022	18/04/2022
Verification of navigation	18/04/2022	25/04/2022
Verification of obstacle avoidance	18/04/2022	25/04/2022
Final robot's production		
Integration of all functions	26/04/2022	15/05/2022
Building and assemble the robot	11/04/2022	15/05/2022
Testing		
Testing in testing arena	16/05/2022	03/06/2022
Testing in final arena	03/06/2022	14/06/2022



5.3. Team management and task division

Team management is a very delicate issue, which could determine the success or failure of a project. To insure a good project management we thus decided to define the responsibilities of each team member at the beginning of the project, so that each one of us could work independently from the others. Besides all along the semester we met regularly to check on our progress and discuss possible problems/solutions, and after each meeting we tried to assign ourselves a specific task with an indicative deadline. Of course things never go as expected, so we kept things informal and we avoided writing things down explicitly, which granted us more flexibility. In addition to those meetings we created a slack channel to be able to discuss easily between us and with our coach as well. In this project, the tasks have been divided as stated in Section 5.3. Note however that at the end of the project and when the competition started to approach we all worked together and helped each other to get things done as fast as possible.

	Task	Student Name(s)
Mechanical design	CAD and robot structure design	Jianan
	Griper design	Jianan
	Storage and retrieval	Jianan
	Assembling the robot	Jianan, Jiangfan, Ramy
Hardware design	Locomotion	Jiangfan
	Obstacle avoidance	
	Bottle detection	
Software design	Localization	Ramy
	Navigation	Ramy
	Integration of the different functions	Jiangfan, Ramy

Table 5.1: Distribution of the workload

6

Discussion and Conclusion

6.1. Competition

During the competition, our robot grasped four bottles from Zone 1 and three bottles from Zone 3 but one of these bottles got stuck on the robot. Therefore, our final points were $10 \times 4 + 40 \times 2 + 40 \times 1 \times 25\% = 130$. We are happy with this results, but indeed, there are some aspects that can be improved:

- The bottle detection of our robot works quite well, so when the robot is moving towards the way point, the robot was interrupted by grasping bottle. Our robot only achieved one way point in 10 minutes. But the advantage is that the robot always tried to go to Zone 3 where the bottles have high points.
- One bottle got stuck on the robot because the bottle cap was stuck on the thin sheets in front of the robot. Two possible solutions to avoid this: change the shape of the thin sheets to a semi-circle rather than a rectangle to help the bottle fall into the robot or change the direction of the hand when releasing the bottle to face more towards the inside of the robot, we did not manage to increase the angle of facing due to the limitations of the servo rotation range. Therefore, the easiest solution would be to change the shape of the thin front sheets.
- There is no feedback to check whether the robot has successfully collected a bottle. We count each grasping motion as one bottle, but in reality a bottle may require two grasping motions (e.g. a standing bottle may require one grasping motion to push it down and one grasping motion to collect it).
- For some unknown reasons(possibly the beacon localization doesn't work when it's close to the beacon, and the displacement monitoring system has some latency), our robot started to turn around when it arrived to the recycling zone and didn't behave as expected. Luckily, it managed to stop this weird motion finally and did the normal bottle release. This situation made us lose a lot of time but hopefully that wasn't critical.
- Our robot had some troubles with the obstacle avoidance. The avoidance wasn't that smooth and it behaved the same way when facing the wall and the bricks. Besides the distance kept when avoiding the obstacles was really not enough. With some more time we would have liked too improve this issue.
- Another element that we would have liked to improve is that the griper can't grasp standing bottles. One solution would be to make longer fingers or use a more powerful servo but all this has to be tested.

Overall, our robot behaved most of the time as expected, as stated before some improvements would have made it a bit more robust but we are already satisfied with the result achieved in such a short period of time.

The most important advise we can give for next teams is to start as soon as possible because lots of problem will appear at the end and the sooner they get solved the better will be the outcome. Indeed, it's hard to deep into something when we know that we still have time but that's maybe the secret to save lots of time afterwards. Also, this project is definitely not the easiest and the challenge is huge so don't try to make something very complex for which you will spend most of the time understanding rather than implementing. Instead, go with the simplest solutions, the one you know they work and afterwards focus on the things you can improve to achieved even better results.

6.2. Conclusion

In conclusion, thanks to this projects we've learnt lots of new things and the quantity of skills we have developed is just flabbergasting. This includes transversal skills in project management but also numerous practical skills. Thus, while participating at the STI Robot Competition, we did not only successfully build robots that could detect, grab and recycle PET-bottles, but we've learnt to develop a whole robotic system from scratch and we are thankful for the chance we had to be part of this experience.

7

Acknowledgments

Such competition and the robot we implemented wouldn't exist without the help and the effort of people we are really thankful to. First of all, we would like to thank Alessandro Crespi, thanks to whom this competition can take place at EPFL. We would like to thank him for his availability at any time and for his help to order the different components. We are also really thankful to our coach Kai Junge who has put lots of efforts to help us to achieve a great result in this competition and we did it! We also thank him for his precious advises and his availability at any time especially when we had troubles with some components. Finally, We would also like to thank the SKIL and the DLL assistants who helped us build our robot.

References

[1] *Choosing the correct upper and lower HSV boundaries for color detection with'cv::inRange' (OpenCV)*. URL: https://stackoverflow.com/questions/10948589/choosing-the-correct-upper-and-lower-hsv-boundaries-for-color-detection-withcv.

[2] Vincent Pierlot and Marc Van Droogenbroeck. *A New Three Object Triangulation Algorithm for Mobile Robot Positioning*. URL: http://www.telecom.ulg.ac.be/publi/publications/pierlot/Pierlot2014ANewThree/. (accessed: 19.06.2022).

[3] Mingxing Tan, Ruoming Pang, and Quoc V. Le. "EfficientDet: Scalable and Efficient Object Detection". In: (2019). DOI: 10.48550/ARXIV.1911.09070. URL: https://arxiv.org/abs/1911.09070.

[4] Shadab Zaidi et al. "Actuation Technologies for Soft Robot Grippers and Manipulators: A Review". In: *Current Robotics Reports 2021 2:3* 2 (3 May 2021), pp. 355–369. ISSN: 2662-4087. DOI: 10.1007/S43154-021-00054-5. URL: https://link.springer.com/article/10.1007/s43154-021-00054-5.

A

Components

A.1. Mechanical Connectors

Most of the component holders were 3D printed and then fixed to the robot with screws and nuts. For laser cut plates, notches were added to the attachment edges to help align the plates, which are then secured with plastic L-shaped brackets, screws and nuts. For sensitive electrical components, 3D printed washers and spacers, plastic nuts and plastic pillars were used for assembly to avoid any damage to them. In total, we used around 12 $\times$ M2 screws and nuts; 32 $\times$ M2.5 screws and nuts; 74 $\times$ M3 screws and nuts; 86 $\times$ M4 screws and nuts. This is one of the reasons why our robots are quite heavy.

A.2. Hardware

Catalog	Component	Note	QTE
Microcontroller	Arduino Mega 2560		1
	Arduóne		1
	Raspberry Pi 4B		1
Wheel	WildTumper 120x60 mm wheel, 4 mm shaft		4
Motor	Maxon EC 32 flat (15 W)	Reduction, adaptor board	4
Controller Board	Maxon ESCON 24/2 controller		4
Mother Board	ESCON 24/2 motherboard		4
Motor plug adaptor	Maxon EC-flat to MM8 adapter		4
Range Sensor	SHARP GP2Y0A21YK0F		5
Camera	Logitech C505e	for beacons localization	1
Mirror	Kogeto 360° dot Camera lens	for beacons localization	1
Camera	Raspberry Pi Camera v2.1 with IR filter	for bottle detection	1
Camera cable	Flex cable for Raspberry Pi camera - 46cm		1
Servo board	Adafruit PCA9685 16-Channel Servo Driver		1
Servo	micro servo HS-82MG	for the hand	1
Servo	Parallax standard servo	for the back door	1
Servo	DFRobot DSS-M15S	for arms and the gripper	3

Table A.1: Hardware Components

A.3. Miscellaneous

Some other material used on the robot: driven cable of the gripper is strong fish line; foam material on the surface of gripper is *tesamoll® E-Profile*; electrical tape, normal tape and self locking straps were used to fix the cables of electrical components; no glue was used.

A.4. Budget

Virtual Budget	
Description of the component	**Amount**
34:1 DC motor with encoder [#3]	-36.95
Raspberry Pi 4 (4 GB) [#10]	-58.25
34:1 DC motor with encoder [#4]	-36.95
Digital compass module (boxtec 47024) [#1]	-23.55
LiPo protection & power board [#15]	-25.00
34:1 DC motor with encoder [#5]	-36.95
Arduino Mega2560 R3 [#14]	-44.50
LiPo Battery 3S (11.1 V) 4000 mAh [#7]	-40.00
Raspberry Pi Camera v2.1 with IR filter [#2]	-25.00
Arduone connector/power board [#5]	0.00
WildTumper 120x60 mm wheel, 4 mm shaft [#10]	-7.50
WildTumper 120x60 mm wheel, 4 mm shaft [#11]	-7.50
MicroSD card 32 GB UHS-I [#12]	-12.25
80 cm IR proximity sensor [#8]	-10.95

Virtual Budget	
Description of the component	**Amount**
Ultrasonic range module (HC-SR04) - 10mm spacing [#7]	-4.00
Micro-HDMI?HDMI adapter [#2]	-3.00
micro servo HS-82MG [#1]	-20.00
Parallax standard servo [#4]	-7.00
Parallax standard servo [#1]	-7.00
USB-C Raspberry Pi power cable (#4)	0.00
APSX RW-210 RFID Reader Module [#2]	-48.80
Arduone v.1.21 connector/power board [#5]	-25.00
Return: 34:1 DC motor with encoder [#3]	36.95
Return: 34:1 DC motor with encoder [#4]	36.95
Return: 34:1 DC motor with encoder [#5]	36.95
Return: Arduone connector/power board [#5]	0.00
Return: 34:1 DC motor with encoder [#8]	36.95
75:1 DC motor with encoder (HP) [#3]	-36.95

Virtual Budget	
Description of the component	**Amount**
Ultrasonic range module (HC-SR04) - 10mm spacing, straight connector [#2]	-4.00
150 cm IR proximity sensor [#1]	-13.95
Ultrasonic range module (HC-SR04) - 10mm spacing, straight connector [#1]	-4.00
Ultrasonic range module (HC-SR04) - 10mm spacing, straight connector [#4]	-4.00
ACI (PCB workshop), order #6190	-58.00
Return: DFRobot FIT0403 90:1 DC motor with encoder [#3]	32.20
Return: 75:1 DC motor with encoder [#1]	34.95
Return: 75:1 DC motor with encoder [#6]	34.95
Maxon EC 32 flat (15 W), 1:60 gearbox [#7]	-108.90
Maxon EC-flat to MM8 adapter [#34]	-5.00
Ultrasonic range module (HC-SR04) - 10mm spacing [#3]	-4.00
Maxon EC 32 flat (15 W), 1:60 gearbox [#17]	-108.90
Maxon ESCON 24/2 controller [#26]	-68.20
WildTumper 120x60 mm wheel, 4 mm shaft [#8]	-7.50

Virtual Budget	
Description of the component	**Amount**
Kogeto 360° dot Camera lens [#2]	-15.00
HDMI-VGA converter [#2]	0.00
WildTumper 120x60 mm wheel, 4 mm shaft [#3]	-7.50
DFRobot DRI0018 2x15A motor driver [#10]	-45.00
WildTumper 120x60 mm wheel, 4 mm shaft [#7]	-7.50
34:1 DC motor with encoder [#8]	-36.95
150 cm IR proximity sensor [#7]	-13.95
30 cm IR proximity sensor [#2]	-11.95
DFRobot FIT0403 90:1 DC motor with encoder [#3]	-32.20
75:1 DC motor with encoder (HP) [#8]	-36.95
75:1 DC motor with encoder [#1]	-34.95
Dynamixel AX-12A motor [ID 1]	-15.00
75:1 DC motor with encoder [#6]	-34.95
Tower Pro MG995R digi hi torque [#1]	-10.00
Parallax standard servo [#2]	-7.00

Virtual Budget	
Description of the component	**Amount**
Parallax standard servo [#3]	-7.00
ESCON 24/2 motherboard [#16]	-12.00
ESCON 24/2 motherboard [#18]	-12.00
ESCON 24/2 motherboard [#13]	-12.00
Infrared sensor 80 cm (4x unmarked)	-43.80
Ultrasonic range module (HC-SR04) - 10mm spacing [#9]	-4.00
Maxon ESCON 24/2 controller [#29]	-68.20
Maxon EC 32 flat (15 W), 1:60 gearbox [#9]	-108.90
Maxon EC-flat to MM8 adapter [#10]	-5.00
Ultrasonic range module (HC-SR04) - 11mm spacing [#6]	-4.00
ESCON 24/2 motherboard [#15]	-12.00
ESCON 24/2 motherboard [#14]	-12.00
ESCON 24/2 motherboard [#17]	-12.00
Ultrasonic range module (HC-SR04) - 11mm spacing [#3]	-4.00
Maxon EC-flat to MM8 adapter [#3]	-5.00

Virtual Budget	
Description of the component	**Amount**
Maxon ESCON 24/2 controller [#22]	-68.20
Maxon ESCON 24/2 controller [#4]	-68.20
Maxon ESCON 24/2 controller [#15]	-68.20
Maxon EC-flat to MM8 adapter [#21]	-5.00
WildTumper 120x60 mm wheel, 4 mm shaft [#5]	-7.50
Maxon EC 32 flat (15 W), 1:60 gearbox [#6]	-108.90
Maxon EC-flat to MM8 adapter [#30]	-5.00
Maxon EC 32 flat (15 W), 1:60 gearbox [#8]	-108.90
Maxon EC-flat to MM8 adapter [#11]	-5.00
Ultrasonic range module (HC-SR04) - 10mm spacing, straight connector [#3]	-4.00
micro servo 2.8kg HK15148B [#2]	-5.00
Ultrasonic range module (HC-SR04) - 10mm spacing [#17]	-4.00
Maxon ESCON 24/2 controller [#8]	-68.20
Maxon EC 32 flat (15 W), 1:60 gearbox [#10]	-108.90
Flex cable for Raspberry Pi camera - 46cm [#1]	-3.00

Virtual budget balance	**77.55 CHF**

Real budget	
Description of the component	**Amount**
Commande Farnell du 11.4.2022	-31.90
Bois + vis	-2.50
Bois découpe	-7.20
Bois + vis + electornic cables	-6.06
Commande Distrelec du 29.3	-111.35
Commande Digitec du 14.4	-29.00
Commande Distrelec du 10.6	-27.05
MDF and PMMA @SKIL	-14.40
Bois + visserie @SKIL	-5.70
bois + PMMA @SKIL	-11.30
cables @SKIL	-1.60

Real budget balance	501.94 CHF

B

Source Files

The source files are attached in the zip files. Here in the appendix are short description of the code files and also some drawings.

B.1. Code

B.1.1. Arduino

motion Interface with the motors and also the control of the motion. Library TimerThree was used to sample the speed at a assigned frequency.

obstacle avoidance Interface with the infrared range sensors and also the avoidance loop.

servos Interface with the servos via PCA9685 and also the grasping and the storage emptying motions.

protocol Protocol to communicate with Raspberry Pi via serial.

B.1.2. Raspberry Pi

Environment: Python 3.9

protocol Interface with the Arduino. Store the message from the Arduino, and decode them when called. Encode the commands and send them to Arduino.

localization Initialize the camera, set camera parameters(e.g. exposure), calculate the position and the orientation from the image.

bottle detection Run object detection model and return the relative position to the nearest bottle detected.

motion control Provide some helpful functions for motion control.

main file Initialize all the modules and run the strategy.

B.1.3. Model Training

B.2. CAD files

CAD files includes all the parts of our robot, and 3D printing parts, laser cutting parts and machined part.

Some CAD files of standard components are downloaded from *GrabCAD Library*: Arduino Mega; WildTumper 120x60 mm wheel; HS-82MG servo; Parallax servo; Raspberry Pi; Raspberry Pi camera V2.1; IR sensor; Tower Pro MG996R Servo.

B.2.1. 3D printing

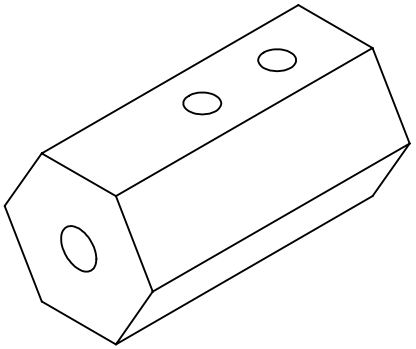
All the 3D printing parts were printed in DLL building with 3D printer *Original Prusa i3 MK3S*. Printing settings are: 0.2 mm SPEED, 20% infill; material used is PETG. The support structure was added to some parts with overhanging structures and then removed after printing. The detailed drawings are not created for 3D printing parts cause only G-code files were needed for 3D printing.

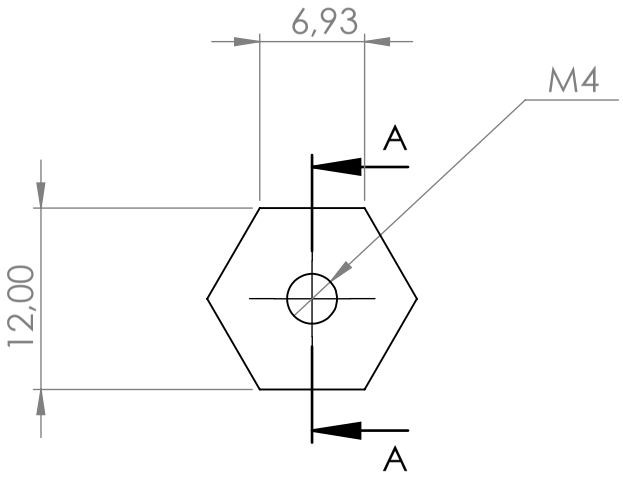
B.2.2. Laser cutting

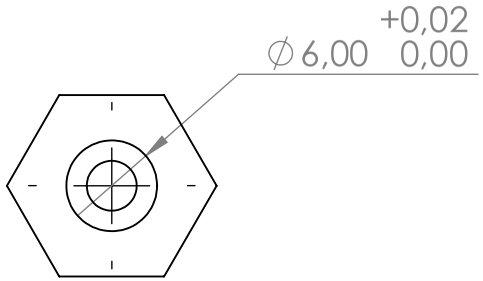
All the laser cutting parts were cut in SKIL with laser cutter *Bodor BCL-0605MU*. The settings are: moving speed 13 mm/s and power 100% for 3 mm thickness material; moving speed 10 mm/s and power 100% for 5 mm thickness MDF; moving speed 10 mm/s, power 100% and cut twice for 8 mm thickness MDF. Again, the laser cutter only need .dxf files, so the detail drawings of laser cut parts were not created.

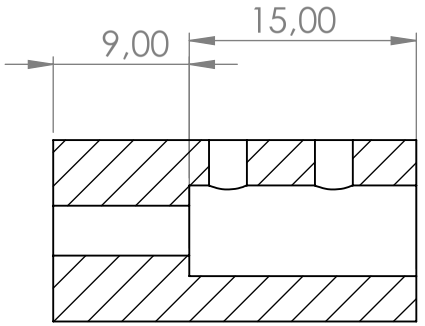
B.2.3. Machining

Two wheel adapters were machined in DLL supervised by DLL coach. The detailed drawing of wheel adapter is presented below. Lathing machine and milling machine were used to drill the holes, then tapped manually.

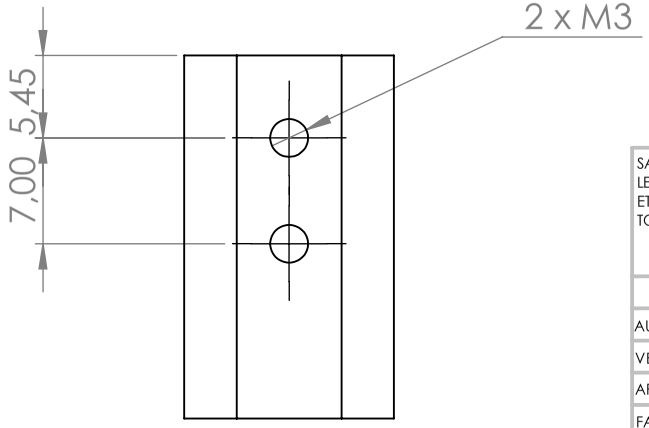






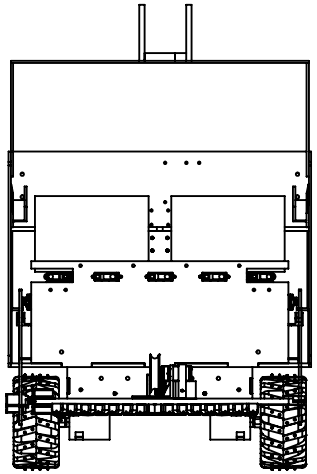


COUPE A-A

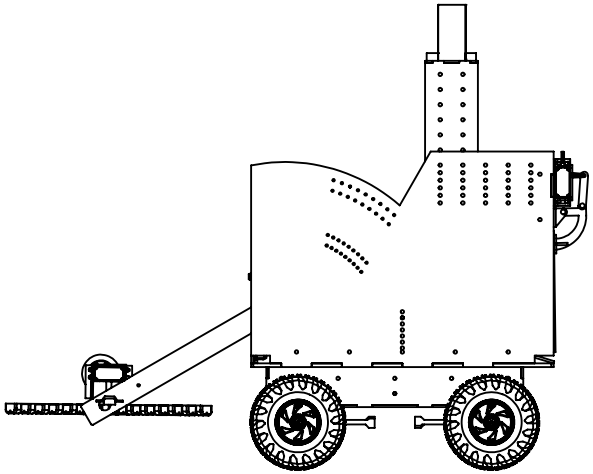


SAUF INDICATION CONTRAIRE: LES COTES SONT EN MILLIMETRES ETAT DE SURFACE: TOLERANCES: LINEAIRES: ANGULAIRES:	FINITION:		CASSER LES ANGLES VIFS	NE PAS CHANGER L'ECHELLE	REVISION		
	NOM	SIGNATURE	DATE				TITRE:
AUTEUR							
VERIF.							
APPR.							
FAB.							
QUAL.				MATERIAU:			No. DE PLAN wheeladapter
							A4
				MASSE:	ECHELLE:2:1	FEUILLE 1 SUR 1	

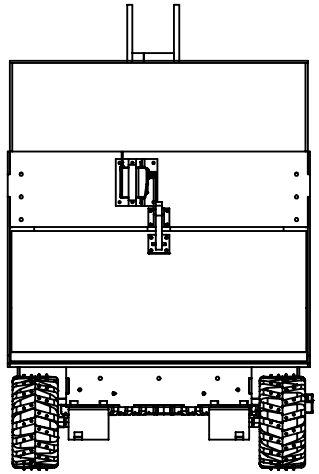
B.2.4. Assembly drawings



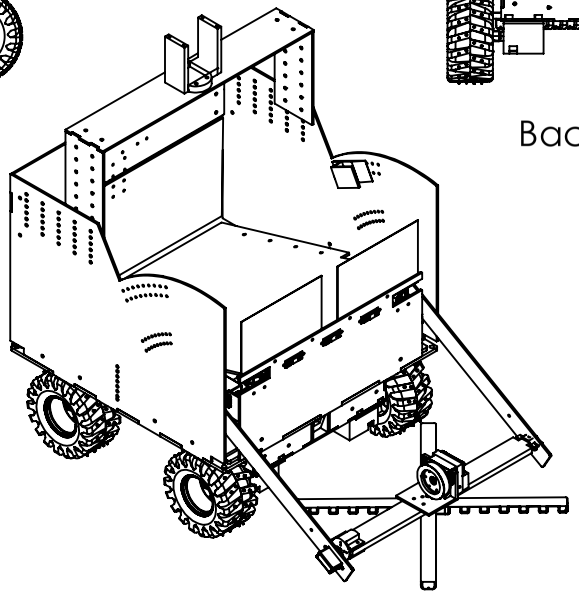
Front view

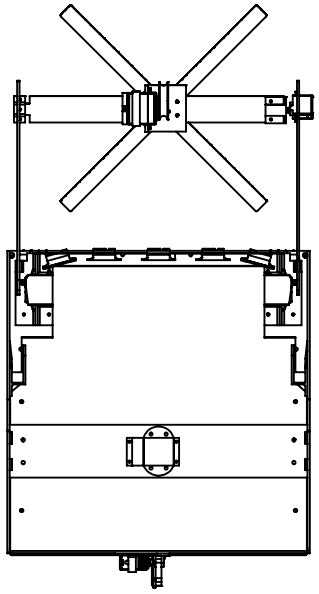


Side view



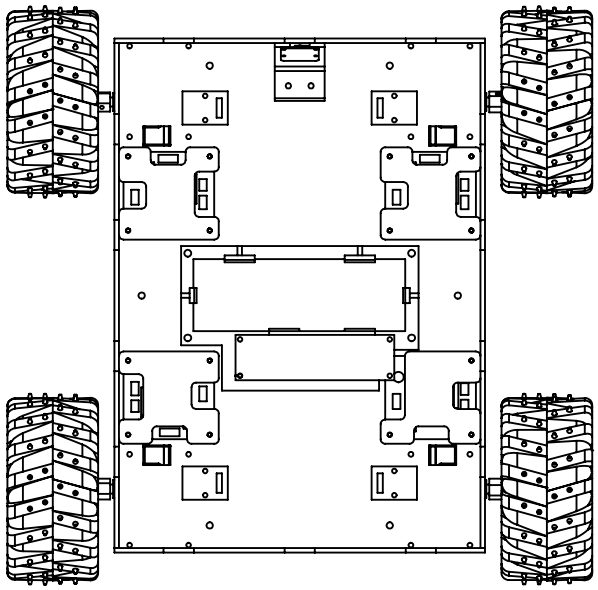
Back view



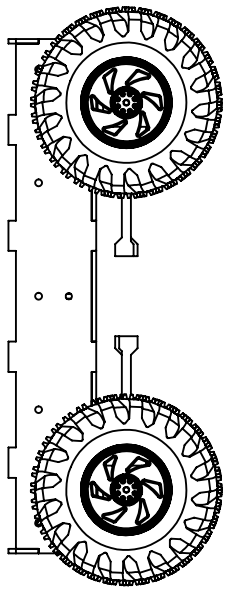


Top view

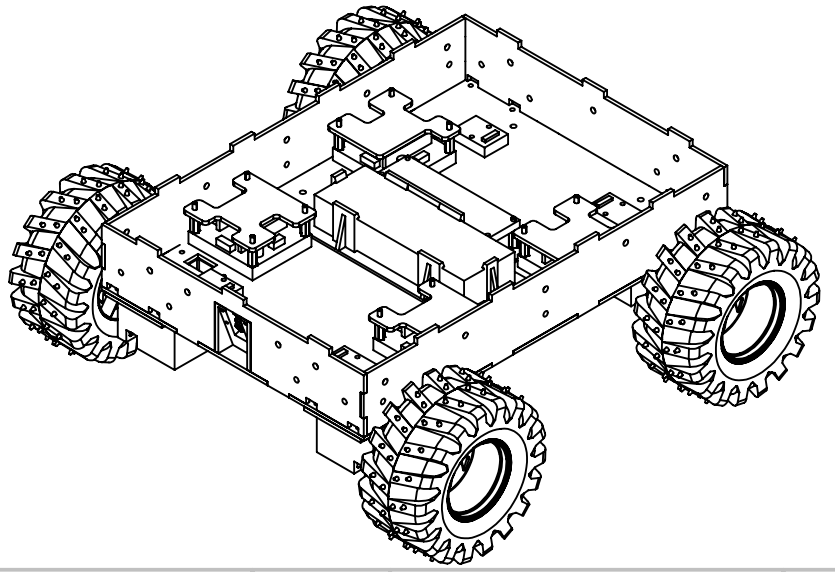
SAUF INDICATION CONTRAIRE: LES COTES SONT EN MILLIMETRES ETAT DE SURFACE: TOLERANCES: LINEAIRES: ANGULAIRES:		FINITION:		CASSER LES ANGLES VIFS		NE PAS CHANGER L'ECHELLE	REVISION
	NOM	SIGNATURE	DATE			TITRE:	
AUTEUR							
VERIF.							
APPR.							
FAB.							
QUAL.				MATERIAU:		No. DE PLAN	A4
						WELL-E	
				MASSE:		ECHELLE:1:10	FEUILLE 1 SUR 1

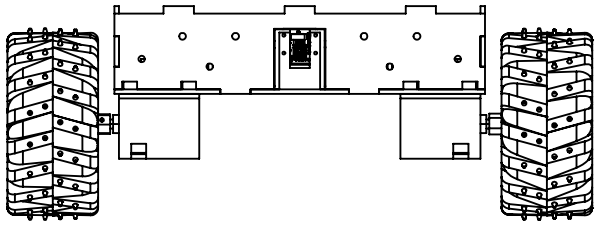


Top view



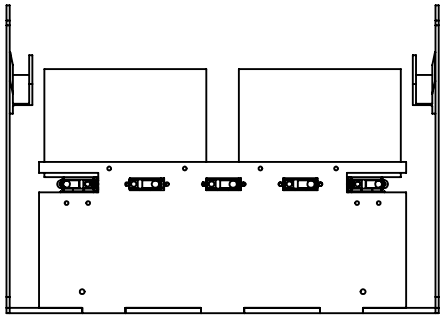
Side view



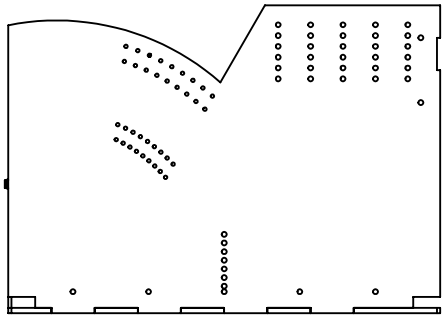


Front view

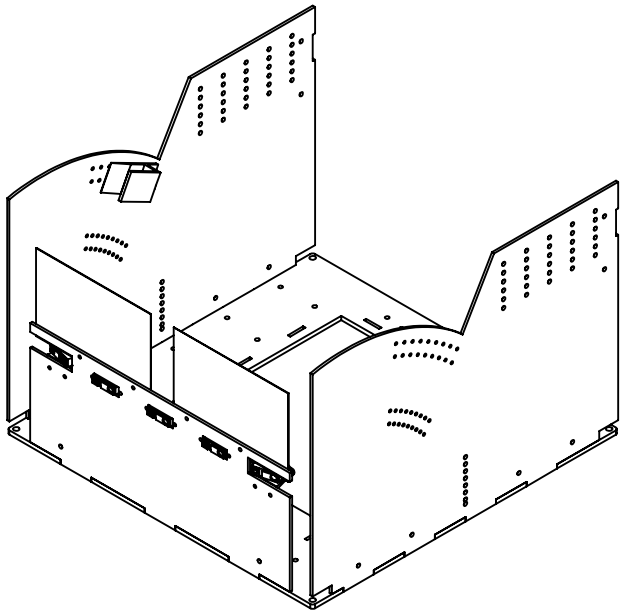
SAUF INDICATION CONTRAIRE: LES COTES SONT EN MILLIMETRES ETAT DE SURFACE: TOLERANCES: LINEAIRES: ANGULAIRES:		FINITION:		CASSER LES ANGLES VIFS		NE PAS CHANGER L'ECHELLE	REVISION
	NOM	SIGNATURE	DATE			TITRE:	
AUTEUR							
VERIF.							
APPR.							
FAB.							
QUAL.				MATERIAU:		No. DE PLAN Basement	A4
				MASSE:		ECHELLE:1:5	FEUILLE 1 SUR 1

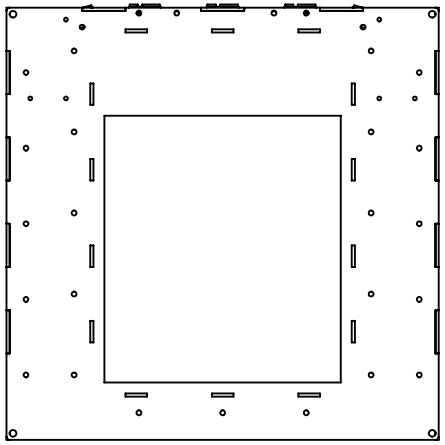


Front view



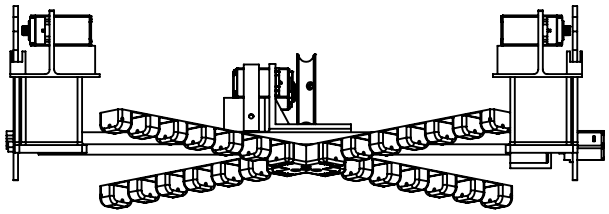
Side view



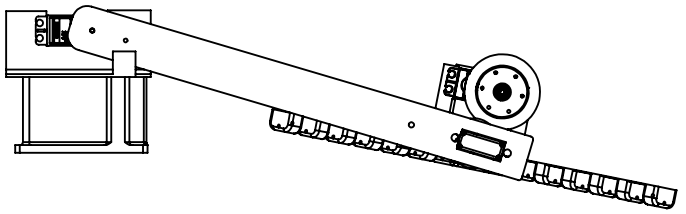


Top view

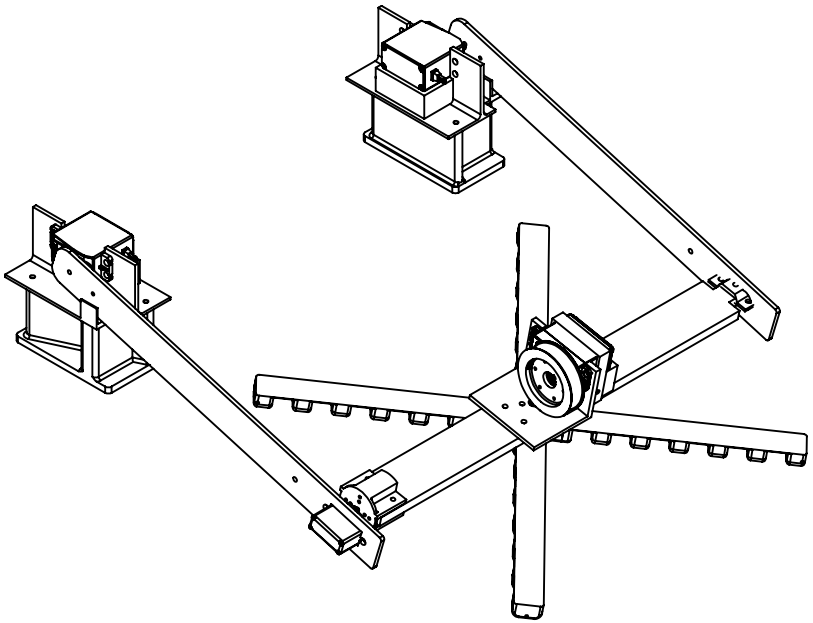
SAUF INDICATION CONTRAIRE: LES COTES SONT EN MILLIMETRES ETAT DE SURFACE: TOLERANCES: LINEAIRES: ANGULAIRES:		FINITION:			CASSER LES ANGLES VIFS	NE PAS CHANGER L'ECHELLE	REVISION
	NOM	SIGNATURE	DATE			TITRE:	
AUTEUR							
VERIF.							
APPR.							
FAB.							
QUAL.				MATERIAU:		No. DE PLAN Second layer	A4
				MASSE:		ECHELLE:1:7	FEUILLE 1 SUR 1

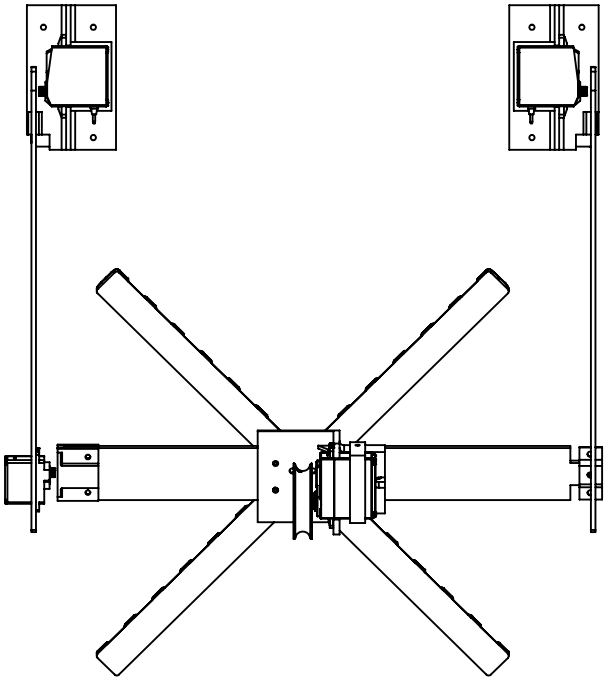


Front view



Side view





Top view

SAUF INDICATION CONTRAIRE: LES COTES SONT EN MILLIMETRES ETAT DE SURFACE: TOLERANCES: LINEAIRES: ANGULAIRES:		FINITION:		CASSER LES ANGLES VIFS	NE PAS CHANGER L'ECHELLE	REVISION	
	NOM	SIGNATURE	DATE				TITRE:
AUTEUR							
VERIF.							
APPR.							
FAB.							
QUAL.				MATERIAU:		No. DE PLAN	A4
						Gripper_system	
				MASSE:	ECHELLE:1:5	FEUILLE 1 SUR 1	

B.2.5. BOM

Table B.1: Bill of Materials

NO. ARTICLE	DESCRIPTION	MATERIAL	QTE
1	Base_plate	MDF 3mm	1
2	Maxon EC 32 flat (15 W)	/	4
3	Maxon 1:60 gearbox	/	4
4	Maxonmotor_mounting	PETG	2
5	Mirror_Maxonmotor_mounting	PETG	2
6	WildTumper 120x60 mm wheel	/	4
7	Maxon ESCON 24/2 controller	/	4
8	ESCON 24/2 motherboard	/	4
9	Pillar_plastic	/	18
10	motor_board_top_1	PMMA 3mm	1
11	motor_board_top	PMMA 3mm	3
12	battery_holder	PETG	1
13	LiPo Battery 3S (11.1 V) 4000 mAh	/	1
14	LiPo protection & power board	/	1
15	Raspberry Pi Camera v2.1 with IR filter	/	1
16	Camera_holder	PETG	1
17	Base_front	MDF 3mm	1
18	Base_side	MDF 3mm	2
19	Base_back	MDF 3mm	1
20	secondlayer_base	MDF 5mm	1
21	wheel_adapter	BRASS	4
22	secondlayer_side	PMMA 3mm	2
23	secondlayer_front	MDF 3mm	1
24	80 cm IR proximity sensor	/	5
25	IR_spacer	PETG	6
26	IR_holder_15degrees	PETG	2
27	thin_plate	PETG	2
28	Arm_stoper15mm	PETG	2
29	backdoor_up	PMMA 3mm	1
30	backdoor_down	PMMA 3mm	1
31	hinge2	PETG	1
32	hinge1	PETG	1

Continued on next page

Table B.1 – continued from previous page

NO. ARTICLE	DESCRIPTION	MATERIAL	QTE
33	hingebar	PETG	1
34	backdoor_Servo_horn	/	1
35	backdoor_servo_holder	PETG	1
36	backdoor_servo	/	1
37	backdoor_servo_cover	PETG	1
38	armservo_holder	PETG	1
39	Mirror_armservo_holder	PETG	1
40	Arm_Servo	/	2
41	Arm_bar_servo	MDF 3mm	1
42	Arm_bar	MDF 3mm	1
43	aremervo_support	PETG	1
44	Mirror_aremervo_support	PETG	1
45	Gripper_center	PETG	1
46	Gripper_finger	PETG	4
47	Hand_bar	MDF 8mm	1
48	Gripper_MG995Servo	/	1
49	Gripper_servo_holder	PETG	1
50	Gripper_servo_Star_horn	/	1
51	Gripper_servo_pulley	PETG	1
52	gripper_bar_fix	PETG	1
53	Hand_servo	/	1
54	secondlayer_plate	MDF 3mm	1
55	secondlayer_plate_hinge	/	2
56	top_camera	/	1
57	top_camera_plate	MDF 3mm	1
58	top_camera_side_plate	MDF 3mm	2
59	top_camera_holder	PETG	1
60	top_mirror_holder	PETG	1