Final Report

MICRO-499 : Microengineering Project II

STI Interdisciplinary Robot Competition:
Bottle-Collector Robot

Audrey Boulard - MT
Émilie Collot - MT
Aurélie Selfslagh - MT

Supervisor       PETER ECKERT
Professor        AUKE JAN IJSPEERT

Date: June 6, 2014
The purpose of this semester project is to build from scratch a robot to collect PET bottles and bring them in a recycling area. The robot should be realised in a group of three people during one semester. The students should develop technical skills through a multidisciplinary and project-based learning. They should ideally come from three different sections of the STI Institute. Soft skills such as interdisciplinary team work, project management and budget management are also expected to be developed throughout this project.

This report is one of the results of this work, the other being the robot. It has been realised by three people from the Microengineering section with different backgrounds. Audrey has a Bachelor degree from Supélec (France) and has therefore a larger background in Electricity. Emilie studied Microengineering at EPFL (Switzerland) while Aurélie studied Electromechanical Engineering at Université Catholique de Louvain (Belgium).

For the methodology, it has been decided to divide the work into three main parts: electricity, mechanics and control. Each member of the group being responsible for one part. The global strategy and the integration of the features have been realised together but the main part of the work has been done individually.

The robot should autonomously collect PET bottles, as mentioned earlier, in an arena composed of different parts, such as synthetic grass or small rocks. The rules are explained in the rulebook on the competition’s official website.

This report starts with the identification of solutions to this project. It then explains the mechanical, electrical and control functions for the final solution that we chose. Our take-home messages are described in the conclusion.

The code, the mechanical plans, the PCB plans and the video of our robot are all provided with this report.
The aim of this section is to present the idea as it was at the beginning of the development phase. The matrices showing the considered solutions are first shown, followed by the final choices that were made for our problem and the specifications that were decided for its implementation.

1.1 Choice Matrices

In order to decide which approach is to be implemented to realise this project, some choice matrices were made (cf. Figures 1.1 and 1.2). For each task that needs to be developed, the different solutions considered were “graded” according to relevant parameters and to each other. The best grade being 5 and the worst 0. Each parameter was then given a certain weight, depending on its importance in the task. Finally, the grading of each solution was summed and normalised to have an indication of the quality of our assessments.
Figure 1.1: Evaluation of the solutions for the big robot

### 1.2 Chosen solutions

**Global Strategy** Multi-robot strategy: a quick one for the easy zone, a slower one for the difficult zones that takes the bottles out of the hard zones

#### 1.2.1 Big robot

**Localisation** Odometry (easy to implement), bumpers (to eliminate odometry errors by bumping into known walls)
Obstacle and bottles detection  Bumpers, whiskers (for proximity detecting), camera (for long-range sensing)

Navigation  "Coverage" algorithm (cover the whole assigned zone), navigate towards the bottles (of course, avoiding obstacles)

Locomotion  2 directed wheels (differential drive) + extra contact point for stability

Bottle management  conveyor belt (combines a solution to bringing the bottles to the storage place, being a storage place that can contain a certain number of bottles and dropping the bottles from the storage place)

After spending more time on these choices, the solution that we came up with for the big robot is shown in Figure 1.3.

1.2.2 Small robot

Global Strategy, Localisation, Obstacle, Bottle detection and Navigation  Same as big robot. Even if a multi-robot strategy was decided, the idea is to have both robots the most similar so that the work is made easier.

Locomotion  The odometry will probably not be completely correct as the whegs might need to do extra rotations before getting somewhere. More bumper position updates
Figure 1.3: Initial complete solution for the big robot

will be needed, but the locomotion can also be determined with the information coming from the camera.

**Bottle management** Advance on bottle to bring it to storage place, no storage place, send the bottles out of the zone

**Bottles sending** Piston

After spending more time on these choices, the solution that we came up with for the small robot is shown in Figure 1.4.

Figure 1.4: Initial complete solution for the small robot

In Figure 1.5 is an idea of how the canon for the small robot could be like.

Figure 1.5: Canon to shoot bottles
1.3 Specifications

Now that the greater image of the project has been decided, it is interesting to determine several specifications that need to be met (cf. table 1.1).

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Value</th>
<th>Explanation / Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field of view</td>
<td>1m</td>
<td>Seams to be a reasonable quantity for a basic camera and image processing. With this field of view the big robot should travel a bit more than 40m to cover visually zone 1 (vs. 150m for a FOV of 50 cm)</td>
</tr>
<tr>
<td>Maximum speed</td>
<td>0.5 m/s</td>
<td>As fast as possible but both robots should be able to stop before touching the objects and obstacles. The electronics should enable a speed of 1 m/s.</td>
</tr>
<tr>
<td>Maximum acceleration</td>
<td>0.5 m/s²</td>
<td>Or rather the maximum deceleration: this value depends of the field of view and the speed for data processing. We assumed that it took less than one second to get the picture, extract the right data and set the command to the actuators.</td>
</tr>
<tr>
<td>Maximum weight</td>
<td>4 kg</td>
<td>Very rough estimation</td>
</tr>
<tr>
<td>Maximum height</td>
<td>400-450 mm</td>
<td>/</td>
</tr>
<tr>
<td>Bottles stored</td>
<td>9-10</td>
<td>(for the big robot only)</td>
</tr>
</tbody>
</table>

Table 1.1: Specifications for both robots. The strategy is to have them quite similar

1.4 Risk evaluation

As we knew that in a project like this one, there are always unexpected things that happen, a risk evaluation was done, and replacement solutions suggested. They are presented in the table 1.2). The solutions are evaluated from 0 to 10, with 0 meaning no risk of failure, and 10 meaning high probability of failure.

<table>
<thead>
<tr>
<th>Risked solution</th>
<th>Evaluation of the risk</th>
<th>Replacement solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catapult</td>
<td>9</td>
<td>Just push the bottles in the easy area</td>
</tr>
<tr>
<td>Conveyor belt</td>
<td>6</td>
<td>Simple box storage</td>
</tr>
<tr>
<td>Obstacle / Bottle idetification</td>
<td>5</td>
<td>Predetermined path</td>
</tr>
<tr>
<td>Overloaded motors</td>
<td>8</td>
<td>Spare motors</td>
</tr>
<tr>
<td>Odometry (precision)</td>
<td>3</td>
<td>Bumpers</td>
</tr>
</tbody>
</table>

Table 1.2: Specifications for both robots. The strategy is to have them quite similar
Development of the mechanical functions

This section aims to present the mechanical features used for the robot. As explained in Chapter 1, the robot is a two differential wheeled robot containing a conveyor belt. This chapter is divided in three main parts, the first two are about the main sub-assemblies of the robot: the conveyor and the transmission and the last one is for other smaller features such as the arms or the shoes.

Each part is subdivided as follows: the problem definition section poses the problem and the requirements, the developed solution and the Solidworks model is described in the design description and the tests and results are interpreted in the evaluation section.

2.1 Conveyor

Problem scope  The scope of this section is to provide a solution that will collect the bottles, store them and drop them in the recycling area by means of a conveyor belt.

Technical review  Although conveyors are often used in industrial plants and combine-harvesters (example for this is provided in Figure 2.1 on the next page), they could not be used as it was without adaptation. No conveyor belt with divisions and with the desired size has been found on the market. The conveyor designed here should be able to carry around 9 or 10 PET bottles of 500ml. It should ideally collect the bottles without external device (no lift or anything else to carry the bottles to the conveyor).

The conveyor consists of three parts (conveyor belt, pulleys and divisions) that are further detailed separately. One pulley is actuated while the other one rotates freely. A system is also designed to put the conveyor belt under tension.

Only two pulleys were used. A larger number of them would have softened the angles for the bottles but the shape would then be less convenient and the number of mechanical component would be increased.

2.1.1 Pulleys

Design requirements

- **Weight:** as the robot is an embedded system, the weight should be minimised for all the parts. This requirement is implicit and won’t be repeated further in this report.
- **Easily fixed** on the body of the robot: the actuated pulley should have a fixed part in order to carry the motors.
- **Size:** the size should be adapted so that the bottles can go around the pulleys without any problem.
Figure 2.1: The grain conveyor (number 4 on the picture) of the combine-harvester was a source of inspiration for the conveyor of the robot. [Credits: http://www.sunyfarmmachine.com]

Figure 2.2: System that applies tension on the conveyor

- **Adherence**: the pulleys should have enough surface in contact with the conveyor belt to avoid slipping.
- **Compatibility with the divisions**: depending on the way the divisions are fixed, the pulleys should not hinder them.

**Design description** The idler (free rotating pulley) is shown in Figure 2.3. It consists of three 20mm-thick discs linked by three bars. The spacing between the two external discs corresponds to the width of the conveyor belt. The discs remain equally spaced thanks to braces (PVC tubes) that are placed on the aluminium bars. The idler is one whole rigid body and the parts are held together only by fitting. The central axis is free to rotate in a hole in the robot shell. The two other ones care for the mechanical cohesion. Additional holes in the discs are there for two reasons: to lighten the system and to allow to reuse the discs for the actuated pulley.

The tension on the conveyor is applied by moving laterally the idler. A flat angle with a screw, two nuts and a shoe for the bar is the system used as depicted in Figure 2.2.

The design is slightly different for the tensioner since actuators should also be included.

Figure 2.3: Final design of the idler
in it. The motors are directly fixed on the shell with a 3D printed support piece. This support also serves as a shaft for the pulley. The width of the pulley should not be larger than the shell. The motor therefore needs to be inside the pulley. A drawing of the tensioner is shown in Figure 2.4a. The mobile part consists of three discs assembled together with 4 aluminium bars. The external discs were made in POM, as opposed to the others that are 3D printed, because they need to transmit the torque and should be more robust. The joint with the fixed part is realised with ball bearings. Those bearings were preferred to journal bearings since the conveyor should support some tension. The choice between ball bearings or other bearings such as roll bearings was only led by the price argument. Two motors are used to actuate the conveyor. This was for symmetry reasons (in actuation and for the weight repartition in the robot) and it allowed a larger security factor for the motors since the efforts needed are difficult to evaluate.

2.1.2 Conveyor belt

Design requirements

- **Suppleness**: should be supple enough to bend correctly around the pulleys
- **Tension**: should be able to support the tension necessary to guaranty the good adherence between the pulleys and the belt
- **Tear resistant**: since no belt with the divisions already included was found, the divisions needed to be added. As discussed in Section 2.1.3 on the next page, the divisions are screwed inside the belt. The belt material should be chosen so as not to tear under the applied tension because of this.
- **Rigidness**: combined with the fixation system, the conveyor should allow to hold the divisions at an angle closed to $90^\circ$ so that the divisions could push the bottles.

Design description  The conveyor belt has been designed out of an old jeans. A first approach, more conventional with a polyurethane belt was first considered but it did not meet the requirement about gashes and was quite expensive. Jeans offers elasticity, is quite resistant and can support some tension. It is not rigid enough to hold the divisions at a right angle but under pressure and with the fixation system explained in Section 2.1.3 on the following page, it offers good results.

The width of the belt was decided to be 160mm. It is smaller than the length of a bottle but it is enough to support them and allows to offer an easier access to the electronics...
A zip fastener has also been added to ease the assembly of the robot.

2.1.3 Conveyor Divisions

Design requirements

- **Fixation**: should be compatible with the belt chosen, for the width as well as for the way it is fixed.
- **Rigidness**: should be rigid enough to push the bottles.
- **Interaction with the bottles**: regardless of the way the bottles should come to the conveyor, they should not be wedged due to the divisions. All the spaces between the divisions or between the division and any other component of the robot should be smaller than the smallest dimension of the bottle (its diameter).
- **Interferences with other components**: the divisions should be small enough to not interfere with the arms of the robot or with the fixation bars or even with the shell.

Design description The final design of the conveyor division is shown in Figure 2.5. The component is 3D printed, allowing good rigidity and a large scale of possibilities in the shape. The height is slightly bigger than the radius of the bottle. Two screws are fixed in each leg on a base that is larger than the rest of the component. On the other side of the belt, small Plexiglas pieces are fixed to press the belt. They are used to maintain the divisions closer to being perpendicular to the belt. A reinforcement bar is present between the two legs. The divisions are fixed on the belt with a certain angle regarding the border of the belt. That way, if a bottle comes with a right angle through the robot, it is deviated when the robot goes forward.

Design evaluation The conveyor has been tested and realised. The realisation is shown in Figure 2.6 on the following page. The tests were quite convincing. Almost all the constraints for each part are met. The rigidity that guarantees that the divisions push correctly on the bottles is a bit critical but, under a good tension and with the Plexiglas pieces, the bottles are able to go all the way round in most cases. Several trials were conducted to reach this results. As an example, the divisions presented here show the fourth prototype.
2.2 Transmission

**Problem scope**  Due to the conveyor and the chosen wheels, direct drive was not possible. The choice was made to design a transmission system although it is not ideal in terms of complexity and yield. The current version of the system has a ratio 1:1. Several reasons motivate this choice:

▷ The motor axis needs to be held in two points so that there is no radial effort on it. A system needs to be designed anyway.
▷ Larger wheels with their axes corresponding to the height of the motors on the robot could have be envisaged. The Wild Thumper wheels have nevertheless been chosen because they were suggested and a larger radius would have required larger torques from the motors.
▷ The reduction ratio could be adapted with a larger granularity. Even though calculations were made to choose the right motor, adaptation could be needed in order to overcome effects that can not be taken into account during simple modeling.
▷ The last reason for the transmission is a didactic one. This project was also an opportunity to test and realise new things. The transmission was one of them.

**Technical review**  This transmission has no particular requirements and does not necessitate state-of-the-art technologies. It just needs to be a transmission adapted to our application.

**Design requirement**

▷ *Adaptation to the existing components:* the wheel and the motor are those used from the catalog of the competition. The transmission needs to fit with them.
▷ *Rigid link:* The transmission should ensure a rigid link between the motor and the wheel shaft.
▷ *Support for the motor:* The transmission should support and protect the rotor of the motor against lateral efforts and impacts that could occur on the wheels.
▷ *Independent part:* The transmission bloc should be an independent part that is fixed on the robot. It means that the components should be fixed between them and not on the robot itself. The reason for this is to ease the assembly of the robot.

Figure 2.6: CAD model of the conveyor
(a) Sub-assembly for the transmission between the motors and the wheels  
(b) realisation

**Design description**  The transmission bloc is shown in Figure [2.7b](image). It is a chain transmission. This was a cheap solution (compared to gears and belts) that was also reliable, easily adjustable and precise enough to be used for odometry. The motor shaft was elongated to permit it to be sustained in two different points. A circlip prevents axial translation. The wheel shaft was replaced by a longer one. Pinions for the chain are fixed to the shafts with planes and clamping screws. The wheel shaft is inserted in the 6 sided hole of the wheel while the motor shaft is secured with a clamping screw on the rotor. To fix the motor (its stator), two screws with a large head are used. They are first screwed in the motor and the screw heads enter two holes in the transmission bloc. Bearings isolate the moving shaft from the fix bloc. A small tensioner wheel is placed on an axis, made of a bolt in an oblong hole, to regulate the tension on the chain. The whole bloc is fixed with screws on the shell. Larger holes are placed on the opposite surface, in front of them to make the screwing possible.

**Design evaluation**  In this assembly, the pinions and the bearings were bought. Their dimensions were not totally adjustable. During the conception, the components were supposed to be retouched (mainly the thickness of the pinion and the depth of the smaller part of the bearings). They were finally not retouched because of technical issues. Adaptations were made but the assembly remained more difficult to realise. An example of this is that the chain needs to be removed in order to fix the clamping screw of the motor (otherwise the screw drive could not face the hole properly).

To ease the assembly, two U shapes could have been used rather than a rectangular one. All the components would have been adjusted inside a U shape and the second U would have been added after, closing the box.

Plastic pinions have been chosen for cost reasons but the use of a clamping screw is not recommended in this material. Other materials would have probably been better for longer use. For this project, plastic pinions were nevertheless enough.

Clamping screws on the motor shaft have caused some issues because the screw thread could easily be damaged. The screw thread’s length was quite short and this can lead to two issues: the force is distributed over a small surface and is then more intense and the screw could be placed sideways which also damages the thread. Other solutions with a hole inside the rotor could have been studied (a small part could be inserted through the shaft and the rotor or special screws could have been used).
2.3 Global Structure

The structure of the robot depends mainly on the conveyor belt. The electronics is placed in the free space between the pulleys of the conveyor. Thin aluminium sheets are used for the shell. Because the lateral plate supports the global structure, they are thicker (2mm) while the one under just serves as a guide for the bottles and is 1mm thick. To reinforce the structure between both pulleys, a bar is screwed on the sides. No rigid part is needed for upper side and a net is used to close the robot. This allows to see inside the robot and is very good in terms of weight and heat. Large holes are placed on the side to allow the access to the electronics. The bottom of the robot is as smooth as possible so that the robot does not grip on the grass.

An additional plate rigidifies the link between the two wheels as shown in Figure 2.8.

![Support piece to rigidify the link between the transmissions](image)

Figure 2.8: Support piece to rigidify the link between the transmissions

Only two wheels are used for the driving. They are placed on the sides, at equal distance between the front and the rear of the robot to make the control easier (it allows the robot to turn on itself). Three additional shoes have been placed under the robot. The one at the rear is a basic plastic piece while those at the front are mounted with springs to guarantee a good contact with the ground at any moment. A plastic tube guides the axis so that it stays well vertical. One front shoe is shown in Figure 2.9. The final version of the shoe had no spring because the friction increased when the shoe was not perfectly vertical and the shoe was difficult to hold in that position with the spring.

In order to align the bottles for their entry in the conveyor, two arms are placed in front of the robot. They overlap slightly to guarantee that the bottles do not go out while closing them and need therefore to open and close asynchronously. Hinges were supposed to support the arms so that all the weight was not supported by the servo. This was finally

![Front shoe](image)

Figure 2.9: Front shoe
not necessary and they were removed.

A thin blade is attached to the bottom of the robot to make a gentle slope when the bottles come inside the robot. The flexibility of this blade is only guaranteed by its fineness.

Bumpers are used for different reasons. To detect on large areas, thin blades with a spring system are used. The system is shown in Figure 2.10. Bumpers were placed at the back and on the sides of the robot to detect obstacles while turning or going backwards. The back bumper is also used for localisation, as explained in the section 4. There are also two bumpers in front (cf figure 2.11), inside the robot, to detect whether a bottle has entered the robot or not.

Figure 2.10: System used for the bumpers to enlarge the detection zone.

Figure 2.11: Whiskers

2.4 Evaluation

Compared to the specifications, the robot is a bit heavier than expected (4.96kg rather than 4.00kg). The choice of the conveyor has two main disadvantages, it reduces noticeably the modularity of the mechanical design and requires a transmission or a special system for the positioning of the motors. If this had been taken into account at the very beginning, the choice matrices would have probably changed. For the rest, except for the transmission that caused some problems due to poor use of the CAD software and the challenging assembly, the mechanics worked quite fine. The conveyor system, that was daring, gave very good results.
CHAPTER 3

Development of the electrical functions

This section presents the different parts of the electrical and electronic parts of the robot. The key points are the choice of the motors, their power supply and their control, the other electrical parts (such as the bumpers or the servomotors) and the integration of all these components on a PCB. Finally, many updates had to be done to adapt the different components to the reality of the robot.

3.1 Motors

3.1.1 Choice of the motors

All the electronic devices that have to be defined depend on the need of energy to run the robot. Four motors are used: two for the wheels, and two other ones for the conveyor belt. Different calculations and tests have been processed to determine roughly which motors should be chosen.

Fundamental dynamic laws First, some parameters have to be chosen, and other estimated to know the torque the motors have to provide. The total weight of the robot has been estimated to $4\text{kg}$, and the calculations have been done with a weight of $5\text{kg}$. The maximum acceleration and the maximum speed have been determined to easily cover the whole easy zone during the allocated 10min : the maximum chosen acceleration is $0.5\text{m/s}^2$, and the maximum speed is $0.5\text{m/s}$ as mentionned in Section 1.3 on page 10.

With these data, the needed torque can be estimated with the fundamental dynamic laws:

$$m \gamma = F_{\text{motor}}$$

with

$$F_{\text{motor}} = \frac{C}{R} N_{\text{wheels}}.$$  

Thus

$$C = \frac{m \gamma R}{N_{\text{wheels}}} = \frac{5\text{kg} \cdot 0.5\text{m/s}^2 \cdot 60\text{mm}}{2} = 75\text{mNm}.$$  

The friction force has been estimated with a quite simple setup : two wheels were fixed to a platform, on which $4\text{kg}$ of bricks have been added. Then, the platform was placed on carpets, similar to those composing floor of the final arena, and which were inclined.

![Figure 3.1: Setup of the experiment evaluation the friction equivalent angle](image)
The aim was to define the equivalent angle of the friction force on the floor, in order to finish the calculations. Different experiments have been processed, and the equivalent angle found was $4^\circ$.

Adding it to the previous results, the calculations gave a minimum torque of $190\text{mNm}$ per wheel.

**Choice of the motors**  No datasheet could be found for the motors proposed for this competition. To determine their behaviour, some experiments have been done. The principle was to apply a known torque on the motor with a system of pulley and weight, and to determine, for a fixed voltage of $7.2\text{V}$, the speed of the motor, and check that the current was still under the limit given by the manufacturer. We did 5 measurements for each of the 6 different torques applied. The results for the motor Pololu 2286 are shown in the graph 3.2.

![Graph 3.2: Calibration of the motors](image)

3.2 **Choice of the electronic board**

The choice of doing a whole PCB was motivated by different points. First, from an educational point of view, it is interesting to develop a whole electronic circuit, and to understand how to do it from start to end. Then, from a technical point of view, having its own electrical circuit leads to a better understanding of all its features, and it is easier to find a problem and then to fix it than on a classical board. Indeed, the signal can be followed from the beginning to the end, and all the steps are mastered. Finally, it is also better from a financial viewpoint to build a circuit than to purchase several circuits for the same result.

The conception of the PCB started with a list of the different features that the circuits should be able to do, and choosing the components which can provide these features. Then, the electrical circuit was designed, the masks for the PCB were drawn, the PCB was printed and fabricated, and finally all the features are tested.

3.2.1 **Features needed for the PCB and chosen components**

The PCB was designed to integrate all the electrical and electronic parts of the robot. It means that it has to provide all the power needed for the other parts of the robot (Arduino, BeagleBoard, sensors...), to drive the motors, and to do an interface between the sensors and the command cards, or between the different cards. Of course, some wires could go directly from a part to another one (for example for the communication between...
the different boards), but it is easier, clearer and it avoids a lot of mistakes to use ribbon cable than several wires. Furthermore, it is more robust: it is harder to disconnect than single cables.

For the power, all the circuits were powered with a 7.2V NiMH battery. This battery has a capacity of 3000Ah, and the instantaneous power that could be delivered was really high (the battery could be emptied in 5min), so there was no power limit on this side. This power supply had to provide energy to the different boards (5V for the BeagleBoard, between 7V and 12V for the Arduino), to power the sensors (the IR sensors needed +5V power supply: an output pin of the Arduino set at 5V was not enough to really power it), and the motors. To create a 5V supply voltage, the voltage convertor L7805ACP was used, as no special power performances were needed so we didn’t care about the energy losses, and the 5V power required by the circuit was really low.

The L298N Full H-Bridge component was used to drive the motors. The 2 motors used for the wheels had to turn in both ways, so a H-bridge was necessary. As we thought of using two 75:1 low power motors suggested by the competition, we did some tests to check how much current they required: even if on the website it was explained that the maximum current was 2.2A, under 7.2V with a stall motor, it was closer to 1A. As the L298N was composed of two integrated H-bridge circuits, each one limited to 2A, this component could fit our needs. However, to be sure to avoid overheating problems, one component was used for each motor. Furthermore, as it was not possible to order less than 5L298N, it didn’t increase the budget. Then, a L298N has also been used to drive the conveyor. For this part of the robot, the two motors which were used could only turn in one direction, but is was easier to use the same component, because we didn’t have to check other datasheets to find a suitable component, and it was cheaper because the order could be grouped with the other ones. However, here only one component has been used to drive both motors, because the conveyor is not used during long periods, so there won’t be overheating problems.

For the interface between the different parts, PCB screw terminals were used to connect mainly the power supply and the bumpers. Then, PCB headers with 2.54mm pitch were used for connecting the motors and the command boards. Furthermore, a 100Ohm resistance was added before all the inputs and outputs that connect to the boards.

As the inputs of the BeagleBoard were limited to 3.3V, a voltage convertor LD1117V33C was used to supply the encoders of the motors.

Finally, is was interesting to add power switches: indeed, it could be useful to be able to stop the power supply without removing the battery, or stop the power supply keeping the logical inputs.

3.3 Design of the electrical circuit

The design of the PCB has been done with the software Altium. It was very complicated, because there are a lot of components, and it is hard to move a placed component when all the wires are in place. The first designed circuit is shown in annex. As we can see, the second H-bridge in the L298N is earthed, to avoid unwanted potentials on these pins. Then, some fast high power free wheels diodes have been added on all the h-bridges, because they are not included in the component.
To avoid potential shifts on the supply entrance, different condensers were added. The values of these components have been chosen regarding what were suggested in the datasheets.

In addition to the different entrances, several PCB Headers have been earthed, to plug the bumpers in, but also because it is always useful in a circuit to have spare grounds. In the first design, all the board entrances were designed to be plugged on the Beaglebone Black, because it was not planned at that time to use an Arduino.

However, many changes occurred after design of this PCB. The main one was the replacement of the motors: indeed, the 75:1 Low Power couldn’t deliver enough power to move the robot on the carpet. Consequently, they were replaced by 75:1 High Power motors, but these last ones needed more current to deliver more power. At first, the first PCB was modified: some lines were cut and others added, to open the second H-bridge on the L298N. These changes worked, but the circuit were less reliable. As some last minute problems were expected, in order to avoid contact failure, which is a problem that takes time to debug, it was decided to design a last minute new PCB. Even if it is based on the former one, the circuit was redrawn again from the beginning, to avoid forgetting to add or remove some contacts in the middle of the circuit. However, this step was faster because of the previous experience on Altium.

The new PCB circuit is shown in Figure 3.4 on the following page.

Figure 3.3: PCB Masks

Here, the two H-bridges in the components are open and driven by the same command. To know which is the best way to connect these bridges, the datasheet examples have been followed.
Figure 3.4: Schematic of the circuit

In this sketch, some connections with the Arduino have been added, and some with the BeagleBoard were still remaining. However, as there were some problems with the implementation on the BeagleBoard, these connections were designed to be connected also to the Arduino if needed. The encoder supply was changed from 3.3V to 5V as they
were computed by the Arduino, but there still was a 3.3V convertor needed to enable the communication between the Arduino and the BeagleBoard. However, we took care of not putting a capacitor close to this area. Indeed, as the communication between the boards was quite fast, we tried to avoid low pass filters.

3.3.1 Drawing of the PCB masks

The drawing on the PCB was a quite long step, as we had no experience in doing so, and the circuit is quite complex. As it was planned to print it in the EPFL workshops, it was designed to be a double layer PCB. The width of the lines depend of the power that will go through them: for high power lines, 2mm width lines were chosen. For the other ones, the minimum width were 0.256mm, because of the technology used in the workshop, but it could be increased if there was enough place. Moreover, the right angles on the lines have been avoided as much as possible, because it can add unwanted signal reflections.

The shape of the PCB was chosen to be easily integrated in the final robot, and the routing was done depending on it. The fixing points were placed on the PCB before the production, because it was easier to drill them at the beginning rather than after everything was soldered.

The components were placed in a quite logical way, to avoid long distances of wire or cross wires. As the ground line was needed almost everywhere, it was placed all around the circuit. It is nearly the same for the power line, which is placed near the H-Bridges. The space between the components had to be respected, to be sure to be able to integrate them after. Moreover, as the PCB was planned to be fabricated in a manual workshop, the vias were not metallized, and each via adds work for the rest of the production. Some tricks have been used, for example placing a line under a resistor, which avoids to place vias. The through-hole components (all the components except the diodes) were also really useful, as they can be soldered both on the top and on the bottom at the same time.

The schema of the PCB lines is shown in annex.

3.3.2 Production of the PCB

We fabricated the PCB at the workshop at the EPFL, on plates coated with 35µm of copper on both sides, because there are some high currents through them. The technology used is photolithography: after washing carefully and drying the plate, it is put in a photoresist bath, then heated at 80°C during 8 min 30. Then, it is exposed to UV light during 70s on both sides at the same time, through the mask designed before. The plate is placed on a revealing solution to remove the unexposed photoresist, and it goes through acid to remove the copper where there is no more photoresist. Then, the rest of the photoresist is removed with alcohol.

When the plate is ready, the different holes are drilled. The diameters of these wholes depend on the components. The width of the pins were checked on the datasheet, but also measured on the components, which permitted to avoid too tight holes. The different diameters chosen were 1.1mm for the PCB screw and the voltage converters, 0.9mm for the H-Bridges, the PCB headers, and 0.6mm for the resistor, diodes, and capacitors. To drill these holes, a machine was used, where a big viewfinder helped to be as precise as possible.

Finally, all the components were soldered on the board.
3.3.3 Checking of the PCB

When the PCB was finished, most of the connections were checked in order to avoid problems after the PCB integration with other things. One problem found was that the shunt resistances on the H-bridges were missing. In fact, we thought they were integrated in the component, but after checking the datasheet, it appears they were not. In the first circuit, the current sensing pin was connected to the ground, because we were sure the current was below the bridge limit, and the regulation of the motors was not done on the current, so there were no problem. As it was possible on the second circuit that the motors required extra current, it was needed to know the current which went through the motors. To do so, a small resistance was added after the production.

The final PCB is shown in Figure 3.3 on page 22.

3.4 Sensors

As encoders were included on the motors, they were used to know where the robot is on the arena with an odometry system, which is explained in the control part.

At the beginning, as the detection of the bottles and obstacles was planned to be done with image processing, no other sensors were needed. For a second check, some bumpers were added at the front, on the back, and on the sides of the robot. To connect them to the control boards, as we could activate pullup resistances inside them, we only connected one wire to the ground, and one wire to control board input through small resistances (100Ω).

However, as there were some problems to install libraries for image computation on the control board, other sensors were added: infrared ones in front of the robot just higher than the highest standing bottle, to detect obstacles, and ultrasound ones on the side of the robot close to the ground, to detect bottles, even if they are lying down. To supply the infrared sensors, the output of the Arduino with a high voltage couldn’t send enough current. As a consequence, we powered them directly with the PCB.

The IR sensors send an analogical signal, which has to be converted into a distance. As a consequence, a calibration table is needed to do the conversion. As the change to this solution were done only few days before the competition, there were not enough time to do that. We thank the Group 3, who gave us their conversion table.

The UltraSound sensors communicate through a digital signal, and the code to get the information of the distance was given on the competition website, so they were quite fast to implement.

3.5 Integration of the electronics on the robot

As the plate where the electronic was put was metallic, plastic headers were put under all the circuits, to avoid short circuits. The wiring was quite fast, as everything were ready before: the PCB was designed in this way. However, as we added the different sensors quite late, there were more wires than expected.
This subsection covers the software control of our robot. We had a first solution that we wanted to do, but changed 2 days prior to the competition, due to both some hardware and software problems, but mostly hardware. For some reason, the installation of Open CV on Beaglebone Black was proving to be harder than it should, and we didn’t manage to get it up and working on our second Beaglebone Black (since the first one burnt on its first try in the actual robot). Both solutions will be presented in the following sections. Some features were common to both solutions, which is why we were able to implement a second solution in such short notice. They are discussed after the presentation of the differences in each method.

4.1 First solution

The first solution was managed both with a Beaglebone Black (Figure 4.1a), a micro-processor using Debian as an operating system, and an Arduino Mega (Figure 4.1b), a microcontroller, that communicate with each other. The Arduino controls the wheels, the conveyor belt and reads their encoders, whilst the Beaglebone Black does all the rest.

A main program in C++ calls many functions that were implemented and tested separately for each "part" of the control, which are mostly shown in Figure 4.2. Parallel to this program runs a second program (using threads) which checks for bumper activations (in case of direct contact with an obstacle).

The different sections that were implemented in the Arduino Mega are:

- Wheel control (PWM, DC motor)
- Conveyor control (PWM, DC motor)
- Odometry (read encoders)
- Absolute localisation

Those implemented on the Beaglebone Black are:

- Arm control (PWM, servomotor)
- Bottle detection (visual extraction)
- Obstacle detection (visual extraction)
- Distance to bottles/obstacles (visual extraction, optical flow)
- Bumper detection (switches)
- Navigation (towards bottles, away from obstacles)

![Flow chart](image)

* For the small robot: Have I finished my area?

Figure 4.2: Flow chart

Since we stopped this solution, the code is not optimal and even in the features that are common for both solutions, it is possible that the version integrated in the code for this part is not up to date.

### 4.1.1 Bumper control

When a bumper is activated (bumper = 1), we want to stop and move out of the way of the obstacle. Since the Beaglebone Black does not have an easy way of implementing interruptions, we used threads.

A thread is a way for a program to split itself into two tasks that run simultaneously (cf. Figure 4.3). Our main program manages the whole control, and in a parallel a thread is running to detect if a bumper switch is activated.

To implement this thread, the poll() instruction is needed. It lets us use a single thread for all bumpers.

When a bumper is activated, we need to move away such that we do not damage the robot. Each trajectory is composed of one or 2 mouvements: move forwards/backwards for 1/3 of the length of the robot + go at 45°. This is not an optimal way of doing it, and has been changed for the second solution.
4.1.2 Beaglebone Black and Arduino Mega communication

To communicate between our two modules, we use a serial communication, the Universal Asynchronous Receiver/Transmitter (UART). The transmitter takes bytes of data and transmits each bit individually and sequentially, whilst the receiver reassembles the bits into bytes.

In our implementation, the Beaglebone sends the following information to the Arduino:

- Initialisation value (to check that we want to read the data at that point)
- Parameter deciding whether we want to move the wheels or the conveyor belt
- Movement parameter:
  - If we want to move the wheels: position to which we want to go (x,y) (from the point where we actually are)
  - If we want to move the conveyor belt: do we want to move 1 compartment or do we want to empty the belt

The Arduino returns the absolute position.

4.1.3 Visual extraction

A moving camera gives a sequence of time-ordered frames, each being a projection of a 3D scene on a 2D image.

The camera used for the visual extraction is the Logitech c270 (Figure 4.4), with the open-source OpenCV software for image capture and processing.

We need to extract from the pictures the height and weight of the objects in pixels, as well as the object’s centre of gravity. These values are then used to calculate the distance to an object:

\[
\text{distanceToObject}[\text{cm}] = \frac{\text{focalLength}[\text{mm}] \times \text{realHeightOfObject}[\text{mm}] \times \text{imageHeight}[\text{pixels}]}{\text{objectHeight}[\text{pixels}] \times \text{sensorHeightToGround}[\text{mm}]}
\]
4.1.3.1 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library.

Feature detection  The robot was planned to navigate with the help of a camera that works with floor plane extraction. The different steps of the image processing algorithm are justified in this section. This code was finally not tested on the robot because of issues with the installation of libraries on the embedded processor. Some adaptations may be required but the general idea is already developed here. An example is shown based on the source picture shown in Figure 4.5a.

Filtering  The first step of the image processing consists of a median blur. This filter is often used for edge detection algorithms because it allows to smooth the image and preserve the edges at the same time. This step enables to get rid of the reliefs on the carpet and the walls as well as high frequency noise. The filter applied here has a kernel of 59 pixels (found by tests, considered as a parameter in the code) and gives the result shown in Figure 4.5b.

HSV  The filtered image is then converted from the RGB domain into the HSV (hue-saturation-value). Only the hue information is used after this step. Because the filtering requires a lot of computational resource, it has been also tested on the hue information only rather than on the RGB image. Nonetheless, that option was not maintained because the results were degraded.

BackProjection  The function backProjection consists in histogram comparison. This is the key step of the algorithm. A reference histogram is taken and considered to be the one corresponding to the floor. The reference histogram is then compared to small histograms taken in the image. One parameter can be adapted here: the number of bins (categories in the histogram). Taking a smaller number of bins amounts to attach less importance to small differences in the hue value. The result of the back projection with 10 bins is shown in Figure 4.5c.

Threshold  A binary threshold is now applied on the result of the back projection in order to find the edges. All the points that have a value under 60 (out of 255, maximal value) are set to 0 and those that are greater are set to 255. This step enhances the edges in the picture. A canny filter was also tested for this function but the results are not as good as in the former version. A comparison of both transformations is shown in Figures 4.5c and 4.5d.
FindContours  The functions `findContours` and `convexHull` are then applied. They find respectively the contours and convex hulls associated with the image. For both cases, the centre of gravity is computed and displayed. The results are in Figure 4.5f and 4.5g. Taking only the contours does not give the same result and the convex hull is needed.

ContourArea  The image is now almost processed but the camera detects a lot of small parts in the image. The selection is done by taking the convex hulls that have an area bigger than a certain threshold. This method is possible here since the camera has a certain angle with the ground and looks at it. All the relevant objects will then fill automatically a large amount of pixels. The final result is depicted in Figure 4.5h.

4.1.3.2 Evaluation and discussion

To be fully operational, some checks should be done about the computational resources needed for the code. This aspect could be critical in an embedded system since the resources are limited and the list of operations should be schedulable on the desired processor. The large kernel of the median filter could raise some issues. About the `moments` function used to compute the centres of gravity, it is probably overkilled. A simpler (maybe a bit less accurate) self-made algorithm could be used for this, based on the extremities of the convex hulls.
(a) Source image
(b) Filtered image
(c) Back projection
(d) Application of a binary threshold
(e) Canny filter
(f) Find contours
(g) Convex Hull
(h) Destination image
4.2 **Second Solution**

For this second solution, we used exclusively an Arduino Mega to control our robot. We replaced visual detection by detection with ultrasound and infrared sensors.

4.2.1 **General concept**

Our program checks for the activation of a sensor every 1000 ms. There are 2 infrared sensors at the front of the robot, 1 ultrasonic sensor on each side just at the base of the arms, 1 bumper at the back of the robot, 1 on each side at the back, and 2 whiskers on the inside of the robot, at the base of the arms.

As long as we do not have 9 bottles, which is the maximum capacity of our robot, if a sensor is activated, it enters a new state to do a series of actions that depend on each sensor, which by default brings it back on a predefined path.

We also wanted to check if the robot was stuck, so if the encoders were blocked even though the robot was given the instruction to move, but did not have the time to implement that part, even though it is a possible state of our state machine.

The different sections that were implemented on the Arduino Mega are:

- Wheel control (PWM, DC motor)
- Conveyor control (PWM, DC motor)
- Odometry (read encoders)
- Arm control (PWM, servomotor)
- Bottle detection (ultrasonic sensors)
- Obstacle detection (IR sensors)
- Bumper detection (switch)
- Navigation (predefined path)

4.2.2 **Predefined path**

Our robot is programmed to follow a path (cf. figure 4.6), on which it will check for obstacles and bottles.

The path is defined per segment. We measure the absolute distance of our robot and compare it to the end of the segment (for the vertical segments, we only compare the vertical component and the same applies for the horizontal segments). Each segment is at ±90° from the other.

Every once in a while, the robot goes backwards until it bumps into a wall (bumper detection) to rectify the error in localisation. It bumps with the back of the robot and updates either the x or the y position.

The path ends in the recycle area, so we do not need to find another way of coming back if we do not have 9 bottles yet.

4.2.2.1 **Boundaries check**

Whenever the robot is given an instruction to go to a certain position, it checks that this position is not out of the arena boundaries, by checking the value of the absolute position added with this new position. Since we decided to go exclusively in the easy zone and maybe in the second zone (the grass), we had to check many boundaries.
The implementation is not integrated in the code, as it was not yet clear what action was to be taken in order to stay in the area that we wanted, all while getting back on our path without colliding with a wall or obstacle.

4.2.3 Obstacle detection and avoidance

Obstacles are detected using exclusively the IR sensors. We had to assign an actual distance to the sensor values, which we could then threshold, such that if a sensor is further than DETECT RANGE, we do not take it into account. We chose a range of 50cm.

Once an obstacle is detected, we compare the values of both sensors. If the left sensor has a greater value than the right sensor (i.e. the obstacle is closer to the left sensor), we will try to avoid the obstacle by the right, otherwise we will avoid it by the left.

To go around the obstacle (cf. figure 4.7), we calculate the distance between the IR and the centre of the obstacle, to which we add the half-length of the robot. We calculate the angle the robot needs to get on the side of the obstacle, at 5cm of it. Once we have this angle, we can calculate the actual distance needed to get there.

\[
\alpha = \arctan\left(\frac{\text{detectionRange} + \frac{\text{maxObstacleLength}}{2} + \frac{\text{robotLength}}{2}}{\frac{\text{robotwidth}}{2} + 5\text{mm}}\right)
\]

\[
\text{length} = \frac{\text{detectionRange} + \frac{\text{maxObstacleLength}}{2} + \frac{\text{robotLength}}{2}}{\cos(\alpha)}
\]

Now that the angle is known, we do the following sequence (which gets us past the obstacle and back in the same orientation):

- turn of $\alpha$
- go forwards of $\text{length}$
- turn of $-2 \times \alpha$
- go forwards of $\text{length}$
- turn of $\alpha$
4.2.4 Bottle detection and collection

The bottle detection can be done in two ways. Either a bottle comes straight into the 'mouth' of the robot (i.e. the bottle was on the robot’s predefined path) and we detect it using the whiskers or it is detected by an ultrasonic sensor.

We have two ultrasonic sensors, placed on either side of the robot, such that we detect the bottles on the side of us when we follow the path. An obstacle is detected if it is closer than DETECT RANGE. When one of these sensors is activated, we go forwards of the distance between the wheels and sensor (so as to put the centre of our robot at the same level as the obstacle). We then turn 90° and check the IR sensors. If the IR sensors measure a distance equal to the ultrasonic (plus a margin factor of 5mm), we actually detected an obstacle. Otherwise, we detected a bottle.

If it actually is a bottle, we go towards it and pick it up, as described in the following sequence:

▷ go forwards of the distance measured by the ultrasonic sensor + 10cm
▷ close arms
▷ turn conveyor belt 1 step
▷ go backwards of the distance measured by the ultrasonic sensor + 10cm
▷ open arms

We then get back on our track by turning 90° in the opposite direction as before.
4.3 Common features

4.3.1 Motor control

4.3.1.1 Pulse Width Modulation

To control the DC motors, we use Pulse Width Modulation (PWM), which is a way of getting analog results by digital means. It is a modulation technique that controls the width of the pulse. The motor receives, through the H-bridge controlled by this signal, a voltage value between 0V and the maximum battery voltage. This value can be modified by changing the duty cycle (ratio of time spent on and time spent off). Here, we implemented a simple PWM, where only two transistors of one side of the bridge are modulated, while the other ones are opened or closed all the time.

PWM offers the advantage of losing very little power. When a switch is off, there is nearly no current and when it is on, there is nearly no voltage drop. The main losses remain the switching losses.

On the Arduino, the duty cycle is defined using analogWrite(), where analogWrite(0) is a 0% duty cycle and analogWrite(255) is a 100% duty cycle (cf. figure 4.8).

![Figure 4.8: Duty cycle values in Arduino](image)

We do not use exactly the same duty cycles for the motors of each wheel, as the motors do not turn at the exact same speed for the same voltage command. In our case, the duty cycle for the right wheel is a little higher than that of the left wheel. For the conveyor belt, the same duty cycle is used for both motors, as the coupling is done by the pulley.

4.3.1.2 Servomotors

The movement of the arms is controlled using servomotors. A servomotor can be precisely controlled: it can be positioned at any angle, usually between 0 and 180°, using a PWM.

Since the arms move in opposite direction, when one arm’s duty cycle is increasing, the other arm’s duty cycle is decreasing, and vice versa.

In the Arduino, a library exists to control a servomotor: Servo.h. Using the write() function, we send as as parameter the angle that we want to reach. We implement the
arms so that they stop at 0, 45 or 90°. The intermediate stop helps the bottles to go in
properly.

4.3.1.3 Encoders
We used motors with integrated shaft encoders. For each encoder, we have 2 outputs. The
frequency of these outputs gives the speed of the motor, while the phase between the two
signals gives the rotating direction. The principle is explained in figure 4.9.

The encoders are read using interrupts. Every time a pulse comes in, the program
checks the state of the 2 channels and increments or decrements the ticks.

4.3.2 Localisation
It is important to know where the robot is during its movement, as we want to implement
a predetermined path. We want to measure the pose variation of the centre of mass of our
robot.

4.3.2.1 Principle
At the beginning of the program, we set the encoders to 0. Then, at the beginning of
each movement, we look at the actual value of the encoders, we update the pose of the
robot with the previous pose and the encoder values, and reset the encoders to 0. This
algorithm is executed each time we change the displacement command.

As the motor used have a 75:1 reduction rapport, and the encoders have an increasing
rate of 48 ticks per rotation, the encoders increase of 3600 per wheels rotation. Then,
knowing the diameter of the wheels (120mm), we could calculate the distance travelled by
the robot with the following equation:

\[
distance = \frac{\text{encoderCount} \times \pi \times \text{wheelDiameter}}{\text{numberOfTicksInOneTurn}}
\]

To measure the angles, we need to take into account that the wheel is at a certain
distance from the centre of mass. Since both wheels turn in opposite directions when the
robot turns (i.e. it only turns on itself), both wheels move of the same distance. The
rotated angle can be calculated in degrees using:

\[
\alpha = \frac{360}{2 \times \pi \times \frac{\text{distance}}{\text{distanceFromCentreOfMassToWheel}}}
\]
Since when we are going straight ahead, and when we are turning both wheels have
the same displacement, we only use the measures of the right wheel.

4.3.2.2 Calculation of the new pose

As explained before, we can easily calculate the new position of the robot. However, as we
didn’t have the time to implement a precise regulation, the command is quite simple: the
robot has to run a certain distance or angle, and then stop. In this case, we always have
a displacement higher than the command. To correct this phenomenon, we calculate the
new position after translation displacement regarding the displacement length and actual
orientation of the robot. The robot has three coordinates \((x, y, \alpha)\), which are represented
in the figure 4.10.

![Robot coordinates in the arena](image)

Then, the new position is calculated with the following equations:

\[
x_{\text{updated}} = x_{\text{previous}} + \cos(\alpha \cdot \pi) \cdot \text{DisplacementLength}
\]

\[
y_{\text{updated}} = y_{\text{previous}} + \sin(\alpha \cdot \pi) \cdot \text{DisplacementLength}
\]

Moreover, when we were going forwards, we are doing it until a fixed position, and when
we turned, the commands were given in absolute angular position rather than relative one,
to compensate the previous errors. So with this system, there is always an error on the
final position, but as the robot knows where it is and the commands are given in relative
absolute position, most of this errors is compensated in the following displacements.
Conclusion

Although the final demonstration of the robot was not so impressive, we could consider this project as a success for several reasons.

First of all, because of the huge amount of learning outcomes. As explained earlier in this report, some parts of the project (especially for the transmission and the PCB) could have been done in a easier way but were still realised as learning experiments. It was almost the first real concrete hardware project for all the members of our group. The fact that we had to build the project from scratch and until the end was also very rewarding. After this project, the take home messages could be:

- **About mechanics**, to be more careful about the way the parts are assembled and the way they could be damaged. A lot of time was spent on assembling and disassembling the robot to change parts that did not resist very well. The place for the electronics and the way it can be accessed also need to be considered from the very beginning. We might not have chosen a conveyor belt as a solution if we had taken that into account.

- **About electronics**, to take more time exploring the existing solutions. Some more efficient systems could be developed and they also have the advantage to be easily reusable for other projects.

- **About control**, to try to integrate the hardware sooner, test the different subfunctions separately as soon as the corresponding hardware is available. In general, trying to integrate as soon as possible after a good study of the problem. If we had followed this rule, we could have changed our solution of the camera and of the BeagleBone Black earlier, and have more time to implement our final solution.

- **In general**, to take larger security margins, more robust mechanical components, motors with higher torques and trying to concentrate on basic features before looking further. Modelling of the phenomenon is hardly ever accurate and higher security factors need to be taken for features with higher risks. Also, multiplying the sources of information, experts, papers, articles and books is essential, as the information contained in it is quite different.

The final robot still needs to be adjusted and the integration of the hardware and the software still needs some work. Nevertheless, each individual part is working and the functions needed for the autonomous navigation, such as the odometry, obstacle avoidance, bottle searching and collection, are operative. Localisation could also be considered as achieved since the robot should come back by itself to the goal, though some parameters still need to be tuned to have it well working.
We would like to thank the STI Section at EPFL and particularly Auke Ijspeert and Alessandro Crespi for giving us the opportunity of doing such a project.

We would also like to thank our assistant, Peter Eckert, for guiding us through this project, mostly at the start when we did not have a clue of what we were doing. He managed to listen to our (sometimes crazy) ideas and tried to look further into them while reminding us of some other solutions and teaching us some practical things.

The AEM made a very good job for our robot, all while teaching us some missing practical background and helping us to transform our ideas in some parts that were mechanically feasible. We would like to thank them for this.

Without the thoughtful assistance of Manuel Leitos, the PCB would have been much more complicated to realise.

We show our full gratitude to all the other participants of the competition, without whom the competition would never have been as interesting, as fulfilling and as enriching. The exchange of ideas and pieces of advice enriched this project remarkably, especially due to our different backgrounds.

We are also very grateful towards Marie-Madeleine, the grandmother of Jérôme Amiguet, who helped us realise the conveyor belt.


APPENDIX A

Gantt chart