UCLouvain

ÉCOLE POLYTECHNIQUE DE LOUVAIN

LELME2002 - Project in Mechatronics

14 May 2022

Final report

TutankhaBot

Group 4

Bernard, Théau	69111800
Bosschaert, Sébastien	11671800
Boulanger, François	12301800
Isenguerre, Nicolas	50041800
Lannoye, Diego	29591800
Mercier, Clément	55031800

eurobot

AGE of BOTS

Contents

CHAPTER 1

Introduction

Sébastien wrote this chapter.

As part of the Eurobot contest, a team of 6 students from the EPL set out to design a totally autonomous robot able to perform some actions while avoiding obstacles and opponents. At the beginning of the academic year, the rules and the theme of the contest "Age of bots" (relative to archaeology) were released. The 6 students then started the design of their beast: Tutankhabot.

Throughout the year, the 6 students have developed skills in the robotic domain, thanks to their courses, that helped them to build the robot. During the first semester of the year, the focus was on the mechanical as well as the electrical design of Tutankhabot. The mechanical aspect was dedicated to the dimensioning of the whole robot, but also to the design of all the mechanisms embedded in Tutankhabot. Then for the electrical part, it consisted in the cabling and control strategy of the whole electronics behind the robot as well as the design of additional PCBs to allow the interactions between the different electrical components. An introduction to robot computing was also made possible thanks to the provision of a minibot from the teaching staff. This helped to understand and use the elements needed to control the robot. Then, the second semester was dedicated to the physical construction of the robot and the implementation of the electronics. This aspect appeared to be the most difficult part of the project, since the mechanical parts and the electronics needed to fit perfectly. Despite those difficulties, Tutankhabot was finally born.

Regarding the academic purpose of the project, the main objective of Tutankhabot was to be able to pass the official homologation of the contest which consists of being able to detect an opponent and to stop, to respect the maximum dimensions and to perform at least one action. Tutankhabot also needed to avoid an opponent. At the end of the day, Tutankhabot was able to perform much more than that. Indeed, it was able to reach a score of 93 points in one match due to the realisation of several actions.

In the following report, we will first cover the general presentation of Tutankhabot through its functions and strategies. Then we will describe the different functions implemented in the robot. This will concern the mobility as well as the functions to perform actions. Then we will come back to the electronics and the programming inside Tutankhabot. Before concluding this report, we will make a critical review of the robot but also of the team itself in the final assessment chapter. We wish you an interesting reading![1]

[1] You may find the requested online archives of the project in [1].

CHAPTER 2

General presentation of the robot (functions and strategies)

This chapter was written by Clément and Sébastien.

After a deep analysis of the rules of the Eurobot contest [2] at the beginning of the project, the Tutankhabot team has decided to opt for several actions to realize during the matches. This initial strategy was meant to be defensive, and the goal was to maximize the following ratio: $\frac{\text{score-efficiency}}{\text{Difficulty to implement}}$. Beyond the essential actions of riding and detecting opponents/obstacles, the purpose of the robot was to perform the following actions:

Table 2.1: Summary of the score for an action.

Action	Score
Leaving the campsite and placing the statuette and the display cabinet	5
Bringing 6 samples to the campsite	6
Swapping the statuette with a replica	15
Storing samples in the workshed	12
Displaying the statuette in the display cabinet	20
Flipping all the excavation squares	25
Returning to the campsite	20
Estimating the score dynamically	+30% of the score (at most)

If the robot was able to perform all these actions in the 100 seconds, it would have obtained a final score of 134 points. To be able to realize these actions, Tutankhabot has been designed as follows: Figures 2.1 and 2.2 show global renders of the designed robot, containing every mechanism. The five different floors of the robot and the included parts are detailed in Figures 2.3, 2.4, 2.5, 2.6 and 2.7. The robot structure consists of four plates. The bottom plate is made of aluminium to ensure the rigidity of the whole robot. It has been machined at LAFAB, its layout has been discussed in the Technical Report[3]. The other plates for the floors are made of laser cut wood at Makilab. To support the different floors, the structure of the robot is composed of hexagonal brass spacers of different lengths. Indeed, we have the following heights for the floors:

- Floor 0-1: 5.7 cm
- Floor 1-2: 8 cm
- Floor 2-3: 8 cm
- Floor 3-Roof: 4 cm

However, the initial strategy had to be revised for several reasons. First, the action of bringing 6 samples to the campsite has been rethought. Indeed, because of the relatively small size of the robot and because of the time limitation of a match, Tutankhabot is only able to bring one or two samples to the campsite. Then, for the action of storing samples in the workshed, the Y-gripper needed too much precision to be successful every time, so it was abandoned for the matches. Finally, the action of flipping the excavation squares that has been implemented was also abandoned because it took too much time and the team didn't have the time to improve this.

After all these considerations, the final strategy that has been adopted is the following defensive strategy:

Table 2.2: Summary of the chosen actions with the corresponding score.

Action	Score
Leaving the campsite and placing the statuette and the display cabinet	5
Bringing 1 sample to the campsite	1
Swapping the statuette with a replica	15
Displaying the statuette in the display cabinet	20
Flipping one excavation square	10
Returning to the campsite	20
Estimating the score dynamically	+30% of the score (at most)

This final strategy is different and more simple than the first strategy in the sense that, instead of maximizing the ratio $\frac{\text{score-efficiency}}{\text{Difficulty to implement}}$, the focus had been placed on realizing the actions that are the most reliable from the existing mechanisms. In the end, by using this strategy, Tutankhabot was able to score a maximum of 93 points.

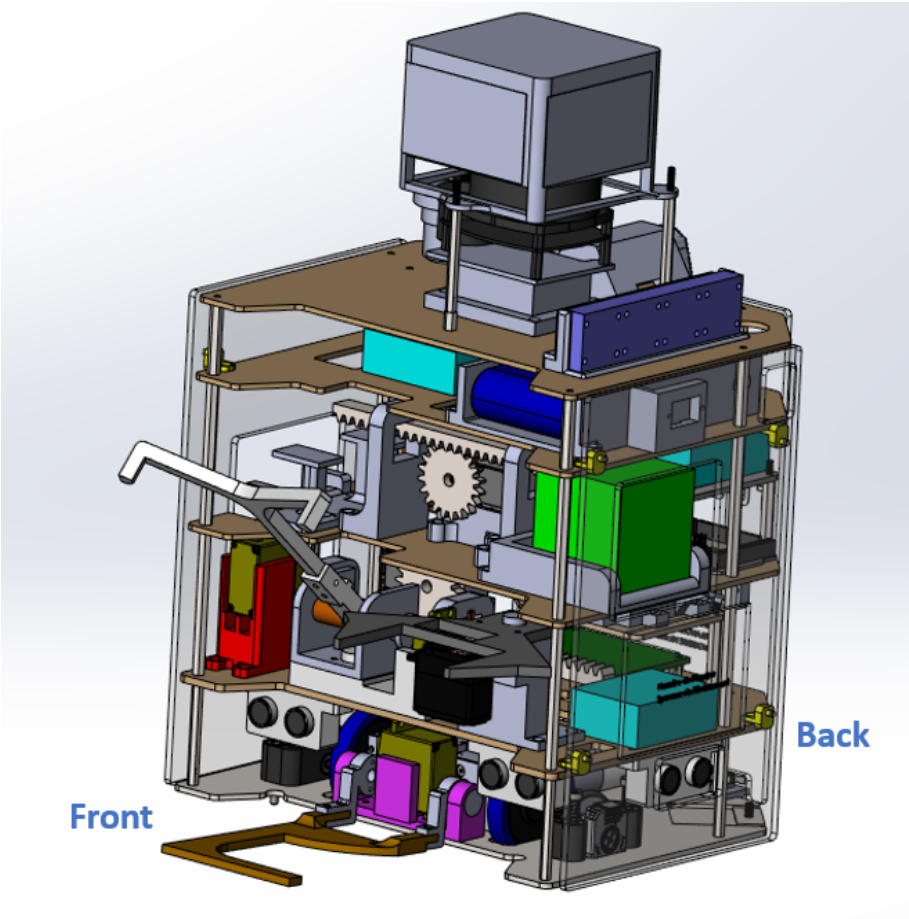


Figure 2.1: Global view of Tutankhabot.

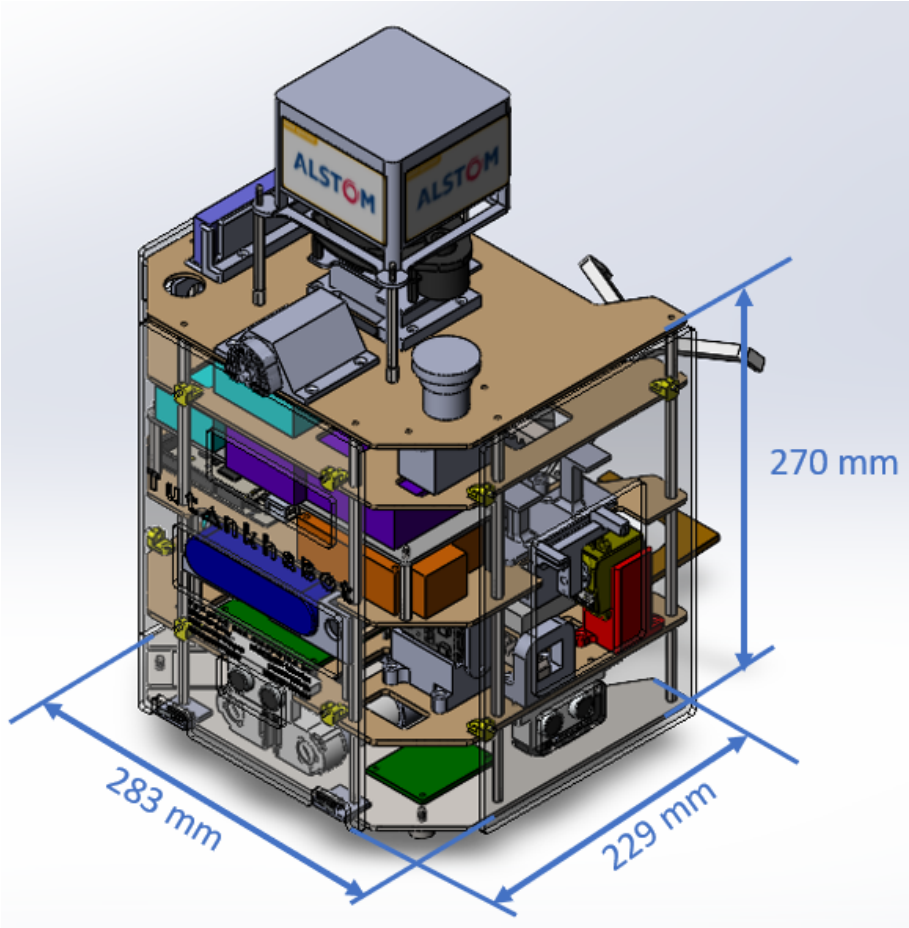


Figure 2.2: Global view of Tutankhabot with main dimensions.

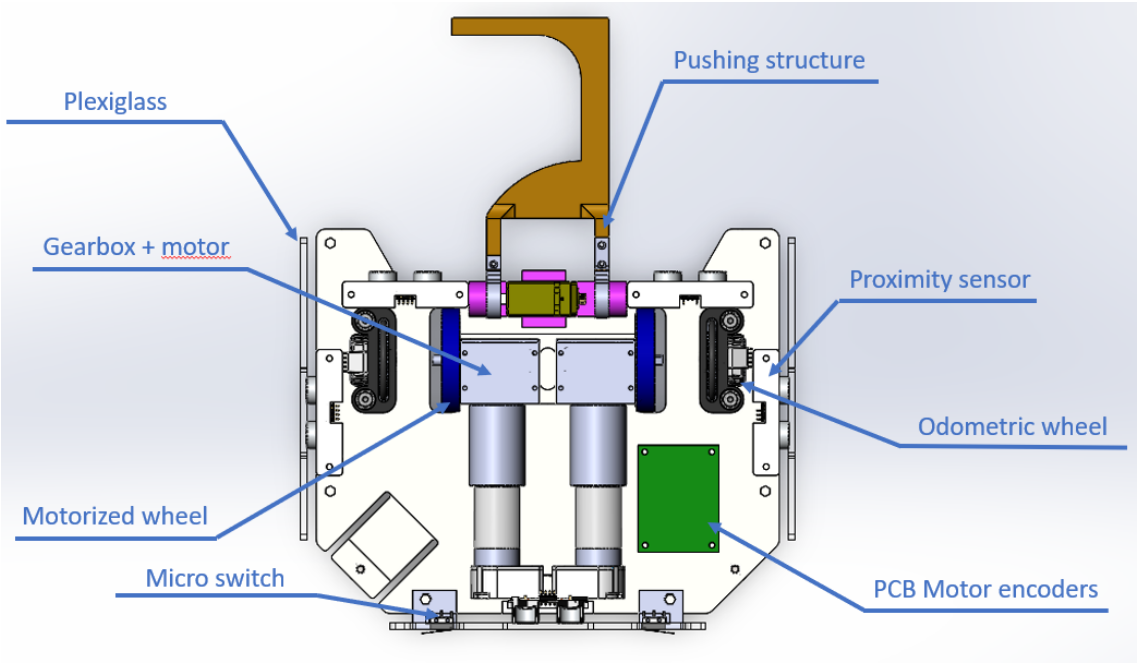


Figure 2.3: Floor 0 of Tutankhabot.

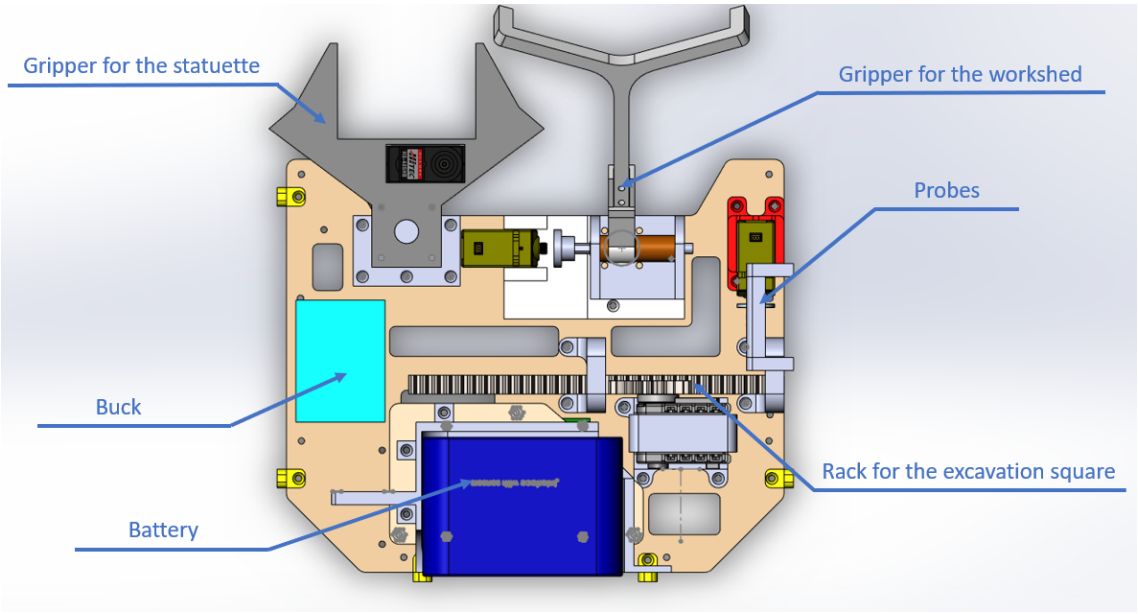


Figure 2.4: Floor 1 of Tutankhabot.

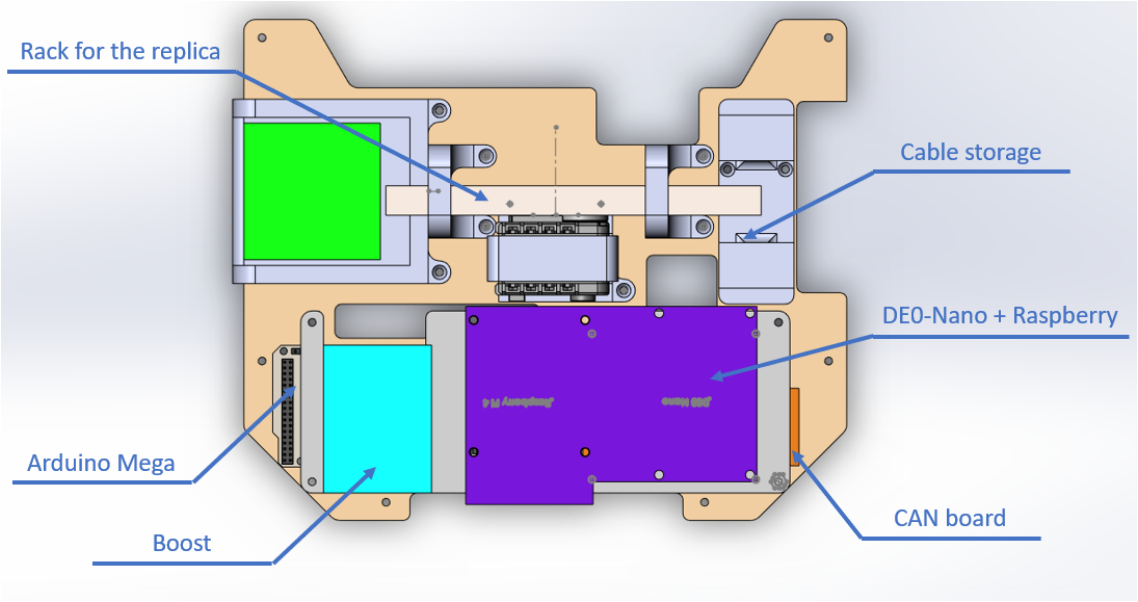


Figure 2.5: Floor 2 of Tutankhabot.

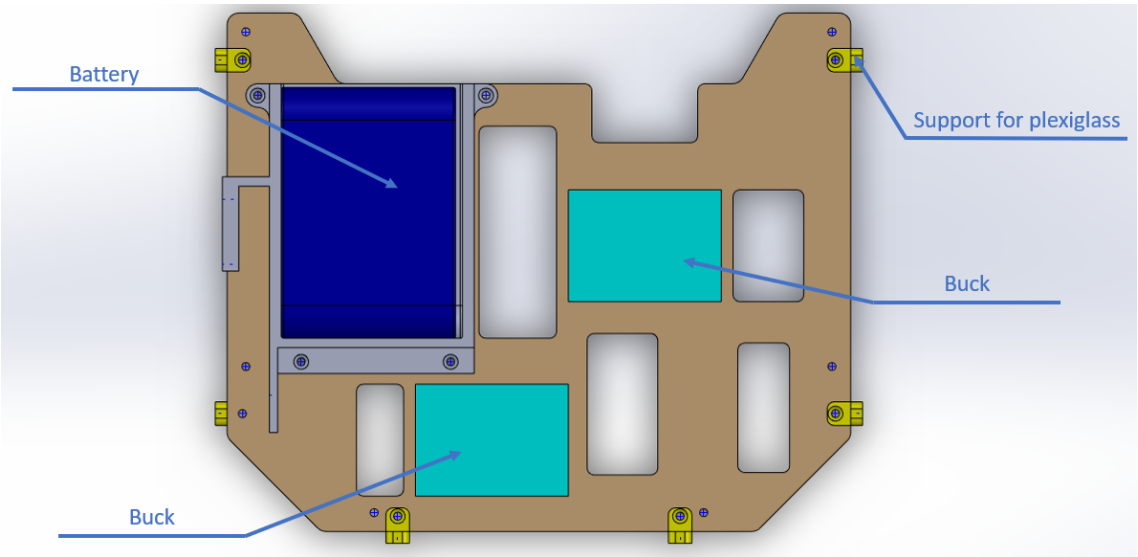


Figure 2.6: Floor 3 of Tutankhabot.

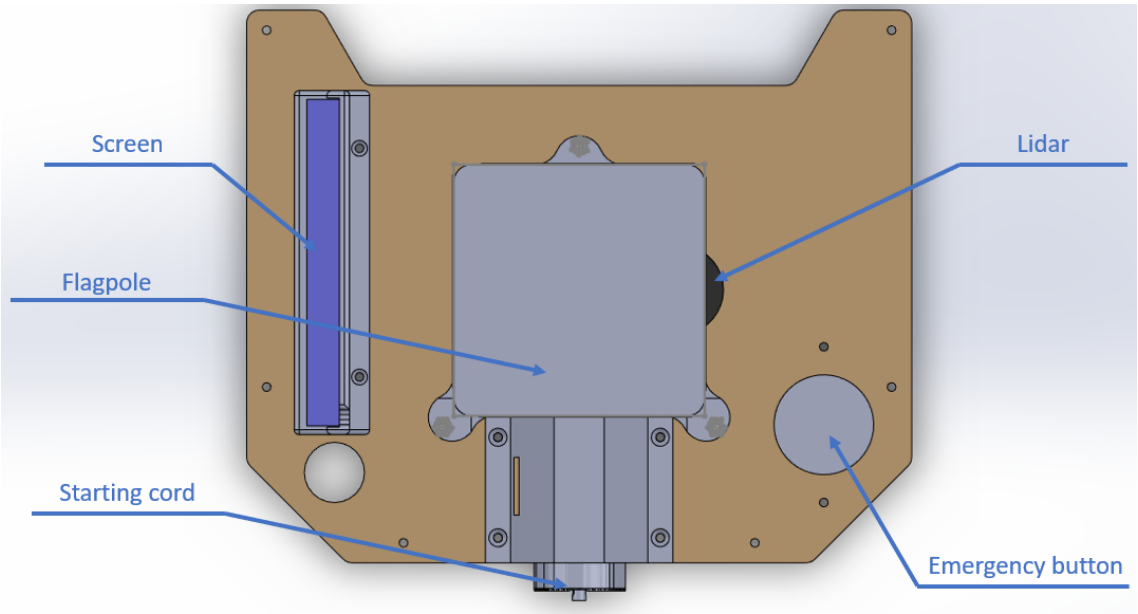


Figure 2.7: Top floor of Tutankhabot.

CHAPTER 3

Description of the functions

In this chapter, Clément and Sébastien wrote the sections referring to the mechanisms of the robot, Nicolas wrote the part referring to the LiDAR, the sonars, and the path-planning and obstacle avoidance in which Diego contributed for the tests and the figures. Finally, François wrote the parts referring to the odometry and the low-level control.

3.1 Mobility

3.1.1 Localisation

Localisation is based solely on odometry as the odometer wheels proved to be robust enough for what Tutankhabot has to achieve. On the other hand, opponent detection is done by a LiDAR placed on top of the robot. The specifications and further means of improvement for localisation will be explained in this section.

Opponent detection with the LiDAR

As previously mentioned, Tutankhabot uses a LiDAR put on top of it to detect the opponent. At this height, the only thing the LiDAR detects is the opponent's beacon support. As this is placed at the center of the robot, this allows Tutankhabot to detect precisely where the robot is by modelling the opponent by a circle whose center is the average of the detected points of the beacon support.

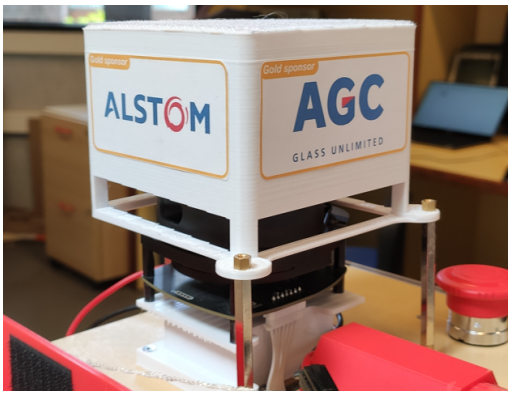
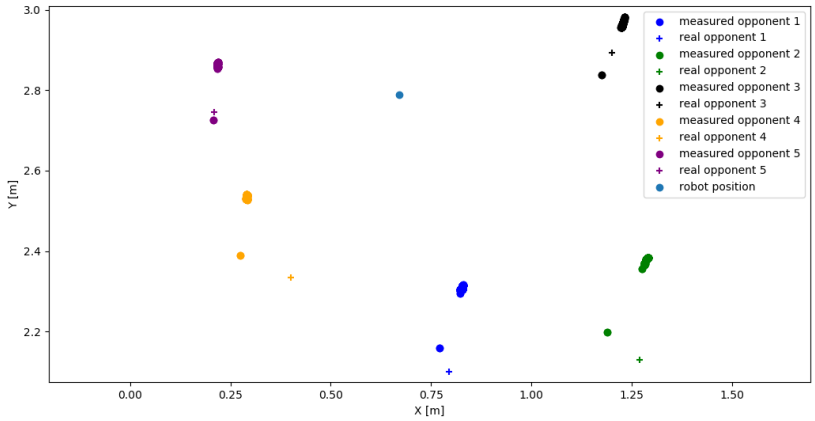


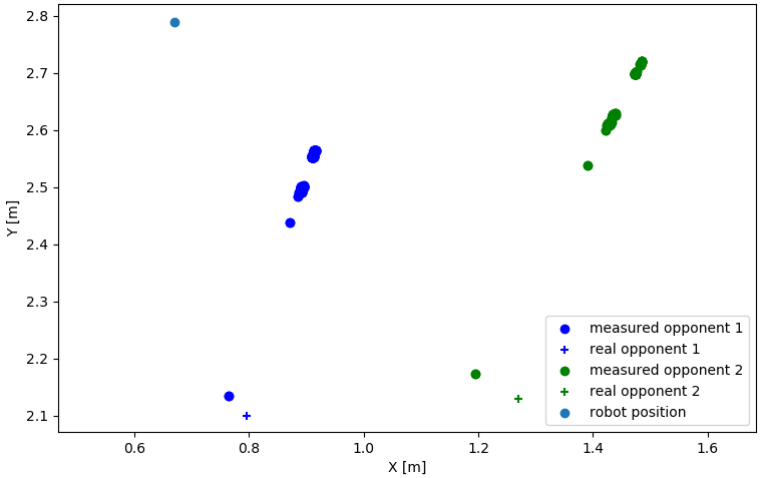
Figure 3.1: Tutankhabot's beacon support is supported by plastic beams, causing blind spots in the LiDAR.

This LiDAR suffers from blind spots, due to the four plastic beams supporting the beacon support, one at each corner of the rectangular support (visible in Figure 3.1), which results in an oscillating behaviour when trying to go around the opponent. Indeed, it initiates the avoidance, but soon looses the opponent as the beams hide the opponent's beacon support. This is due to the repetitive appearance and disappearance of a strong repulsive force due to the close presence of the opponent, acting like an obstacle for the potential field algorithm, which will be explained in subsection 3.1.3.

These blind spots make the static detection of the opponent harder at certain angles, which results in an error of detection. The following figures illustrate this problem. Also, the LiDAR has a harder time detecting certain beacon support shapes. Indeed, if a perfect cube with flat surfaces is easier to detect, a thin tube will lead to more errors, as illustrated in Figure 3.2a and Figure 3.2b. Each point represents an average of all the points measured at time t by the LiDAR. Several tests are represented in each figure.



(a) Test with a flat surface: the points indicate where the LiDAR thinks the opponent is, in average, at each time t. There are some errors in the measurement, mainly in the y-axis. For the yellow and green dots, there are also errors in the x-axis: this is due to the blind-spots, which makes the detection harder for certain angles.



(b) Test with a thin tube: the points indicate where the LiDAR thinks the opponent is, in average, at each time t. There are bigger errors in the measurement, principally in the y-axis. The error is bigger for the green dots, once again due to the blind-spots.

Figure 3.2: Results of tests of the LiDAR's performance for a flat surface and a thin tube.

Table 3.1: Summary of the test for the LiDAR whose graphical results have been shown in Figure 3.2a. The position of the robot is redefined as (0.0,0.0). The angle 0° is defined in the opposite direction of the y-axis.

		Test 1	Test 2	Test 3	Test 4	Test 5
Real x position	[cm]	12	60	53	40	21
Static x error	[cm]	3.03	1.4	2.69	-10.9	0.7
x standard deviation	[cm]	0.52	0.89	0.5	0.17	0.1
Real y position	[cm]	-69	-66	10.4	-45.5	-4.5
Static y error	[cm]	20.6	24.5	7.2	20	11.6
y standard deviation	[cm]	1.35	1.65	1.24	1.32	1.21
Real angle	[°]	10.3	42.3	111.1	-30.7	-84.4
Static angular error	[°]	7.53	13.7	6.5	-25.2	-14.5
Angular standard deviation	[°]	0.84	1.39	1.14	1.06	1.515
Real distance	[cm]	70.12	89.2	54	52.9	46.3
Static distance error	[cm]	-19.2	-15.0	4.4	7.1	-0.43
Distance standard deviation	[cm]	1.12	0.4	0.77	0.96	0.1

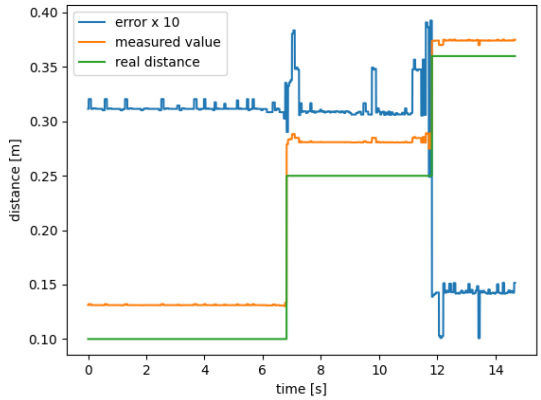
From the Table 3.1, it is deduced that the implementation of the LiDAR suffers from important static inaccuracies but offers a great precision (high static error but low standard deviation). The static error in distance decreases as the opponent approaches, this offers more accurate measurements when the situation is more critical.

Usage of sonars for proximity localisation

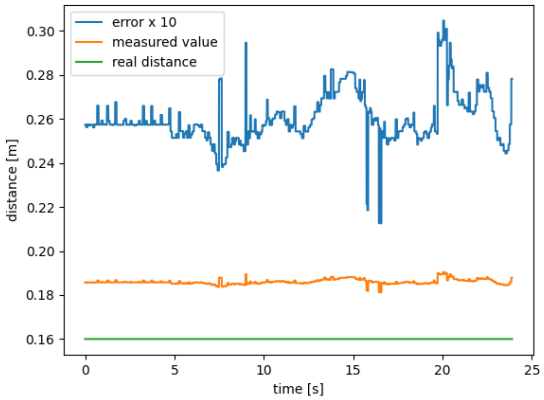
The sonars' functioning is explained in details in section 4.4. These sonars could only prove useful in close proximity to detect opponents, as borders are avoided using a potential field algorithm (see subsection 3.1.3). Several tests were carried out to assess their correct working. Only flat and smooth surfaces were used for these tests. Their results are visible in Figure 3.3a, Figure 3.3b, Figure 3.3c.

- **First test**: for the first test, several surfaces were put a known distance apart from each other, starting at 10 cm from the robot, and every 5 s, the closest surface to the sonar is removed. The other surfaces are 25 cm and 35 cm from the sonar.
- **Second test**: the robot is moved along a line, parallel to a wall. In the outward direction, it is not disturbed. On the way back, the robot is disturbed manually by shaking it gently.
- **Third test**: the same as the second test, but further away, to see if the distance increases a possible error.

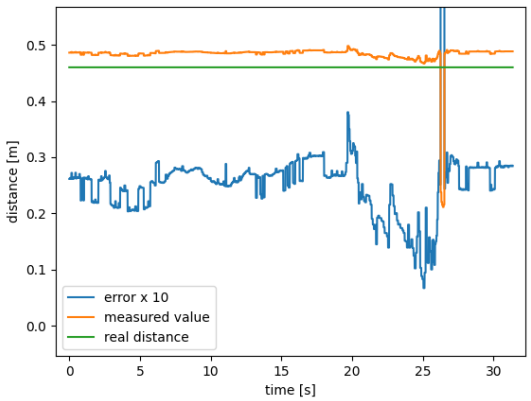
You may find an illustration of the measurement protocol of the third test here.



(a) Results of the first test.



(b) Results of the second test.



(c) Results of the third test.

Figure 3.3

In the first test, in which the robot doesn't move, it appears that the sonar has a static error of around 3 cm. The biggest errors appear when the tester removes the surface that is being detected to reveal the next one: there is a kind of settling time when the data is less reliable. This gets worse the further the surface, as visible on the right of Figure 3.3a. The static error seems smaller for a distant surface. A resume of this test is presented in Table 3.2.

Table 3.2: Summary of the first test for the sonars.

Real distance [cm]	Static error [cm]	Standard deviation [cm]
10	± 3.1	± 0.035
25	± 3.1	± 0.23
35	± 1.4	± 0.11

Table 3.3: Summary of the second and third tests for the sonars.

	Real distance [cm]	Static error [cm]	Standard deviation [cm]
Test 2	16	$\pm$ 2.6	$\pm$ 0.12
Test 3	46	$\pm$ 2.3	$\pm$ 2.5

In the second test, the static error is judged smaller but more variable. In the third test, the static error is more variable, indicating that the sonar has more difficulties to measure further distances when disturbed. As visible in Table 3.3, the standard deviation globally increases for these tests. It appears that the further the object whose distance to the robot is being measured, the smaller the static error proportionally to the real distance, but the more variable it will be when the robot is disturbed in its path.

However, as mentioned in the datasheet of the sonars [4], it is highly recommended to use them for detecting flat surfaces as smooth as possible, oriented parallel to the sonar. This is illustrated in Figure 3.4, where it is made clear that the sonar struggles to detect an oblique border.

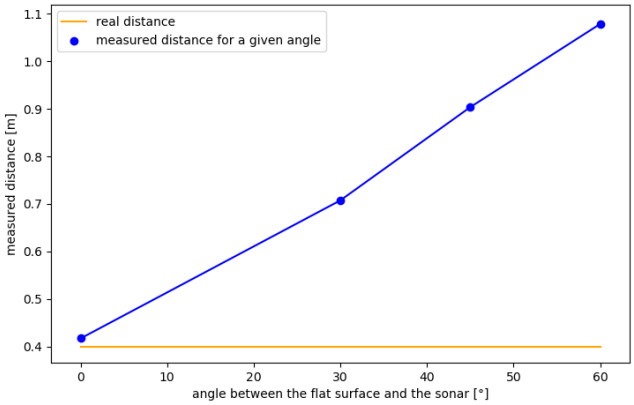


Figure 3.4: Distance measured for different angles. The obstacle was at a known and fixed distance of 40 cm. The error increases roughly linearly with the angle, proving that the sonars are not usable in practice, as most of the time, the surfaces to detect will not be perfectly flat and parallel to the sonars.

There were initially 5 sonars planned. At the end, only 4 were mounted on the robot, and 1 of them was not even cabled. The reason behind this is that, although working fine as explained above, they were deemed not useful enough to justify their final integration to the robot. In fact, their disposition in Tutankhabot could prove to be useful to detect close obstacles. However, as borders are avoided using a potential field, and they would only be useful to detect the opponent arriving from the front or the sides, zones where the LiDAR does not suffer too heavily from errors. And if they may correct the data for these situations, as soon as Tutankhabot would start avoiding the opponent, the sonars' data would be heavily disturbed by the non-flat surface and would not help improve the LiDAR measurements, possibly even further degrading them. This is why the group decided not to use them in the end.

Odometry

To locate itself, the robot uses odometers that will permanently keep track of the motion of the robot. The controller of the robot will then integrate it in order to compute the exact position at any time of the game.

The odometers are wheels that are placed on the same axis as the motorised wheels to avoid kinematic conflicts with the latter when the robot is undergoing displacements. The wheels of the odometers are also placed on really soft suspension in order to ensure the contact with the ground without supporting the weight of the robot, which is not the purpose of these wheels. The odometer system is also composed of an encoder similar to those used for the motors which is coupled in rotation to the wheels, allowing to evaluate the rotation speed of the wheel.

To compute the position of the robot, the controller uses an incremental method consisting in incrementing the current position according to the velocities of the odometers. The formula used is the following :

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta s \cos\theta + \Delta\theta/2 \\ \Delta s \sin\theta + \Delta\theta/2 \\ \Delta\theta \end{bmatrix}$$

In which $\Delta s = \frac{\Delta s_r + \Delta s_l}{2}$, $\Delta\theta = \frac{\Delta s_r - \Delta s_l}{b}$, b is the distance between the two odometers and Δs_r and Δs_l are the increment of position of the right and left odometers computed via their speed and the time of the iteration loop.

We can see that the algorithm for the position depends on the radius of the odometers and the distance between them. An error in the measurement of these latter could lead to an error in the position. That is why these quantities have been recalibrated by performing a well-known longitudinal motion with the robot and comparing the computed and the real length

of this motion, allowing us to recompute the radius of the wheels. Secondly, the same procedure has been performed to recompute the distance between the odometers by performing a circular motion.

The performances of the odometers are limited by several factors. Firstly, the algorithm that computes the location of the robot is based on the hypothesis that the velocity of the robot is kept constant during the iteration loop which is, in practice, not always the case when the robot is undergoing accelerations. Moreover, another source of noise directly comes from the odometers, which is the quantisation noise since the encoder is a digital sensor. To be able to evaluate this uncertainty, we have separated the error in two categories, the longitudinal error and the angular error. Experimental tests lead us to the conclusion that the longitudinal error, occurring during rectilinear displacements, is of 2 mm/m. The angular error is of about 1 degree/revolution and occurs during rotation motions. By integrating those values, we can obtain that the total error at the end of a match on the location of the robot is of a few centimetres, depending on the displacements performed during this match.

Possible improvements to localisation

There are several means by which Tutankhabot could improve its localisation:

- **Re-adapt the bottom plate:** the design of the bottom plate could be improved by taking the odometers and the wheels away from the centre, improving the accuracy. The system of fixation of the odometers could also be revamped to improve the contact with the ground of the odometers, which was not always guaranteed when the ground was not perfectly planar.
- **Triangulation**: using the LiDAR and three beacons placed on the supports provided on the borders of the map, the robot could correct its localisation using this technique. With only two beacons at a time and by knowing the distance between these two, by measuring the angles of the robot to these points, the robot can precisely know where it is. The third beacon brings redundancy and higher accuracy if detected. However, this would be hard to implement in practice, as the LiDAR is error-prone, and a slight error would cause the robot to lose its location.
- **Include the sonars in the final implementation**: they could be added to improve localisation by feeding the robot information about nearby borders, obstacles, etc. which could be used to estimate if the current location might be wrong or not.
- **Kalman filter**: although it was deemed not necessary, this filter would highly improve the limitation of the error on the odometry.
- **Recalibration**: the best way to correct any possible error is to make the robot do another calibration, like at the start of a match. However, this is really costly in time, and would severely penalise the overall performance of Tutankhabot in a match, although deleting all possible localisation error. Another recalibration protocol that would not take too much time could be implemented, but it would always lead to losing time.

3.1.2 Low-level control and actuation

As shown in the previous chapter, the robot we implemented is a differential drive robot with two motorised wheels powered by DC motors taking as input a constant voltage. The voltage is given by the motor controller, which will be fully described later in this report. The function controlling this controller take as input a command ranging from -100 to 100 respectively corresponding to the minimal and the maximal level of voltage.

To control these motors in speed, a proportional-integral (PI) controller has been implemented, taking as input the reference and the actual speed of the wheel that is controlled. After facing issues with the encoder of the motor and the PCB that was dealing with the signal of these latter, we decided to control the robot with the odometers, considering that the wheel that is the closest to this odometer is actually controlling it. This implementation is risky, but we have never faced any complication due to this implementation.

The coefficients of the controller have been tuned experimentally after a few trials. The robot was first controlled with a proportional controller to tune the proportional gain. Then, the integral action was added to adjust the response time of the system. As you shall see in the programming section, the time of the loop is rather high (about 20 ms). That is why we decided to limit the step response time of the system in order to limit the issues. The step response of the two wheels is given in Figure 3.5. The proportional and integral gains of the controller are finally and respectively 3.0 and 15.0. The 95 % step response time of the system is 0.76 s.

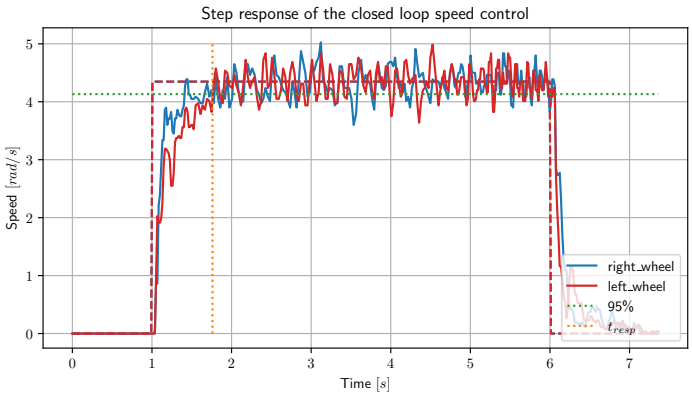


Figure 3.5: Step response of the low level controller (reference speed dashed).

3.1.3 Path planning and opponent avoidance

As already stated in [5], the method used for path planning and obstacle avoidance is the Potential Field algorithm. If the basics are the same as explained in this report, many things have been improved in order to adapt the theory to reality. All the changes and obtained results are detailed in this section.

The formulas

If the formula for the attractive force detailed in [5] remains the same, the formula used to compute the repulsive force changed. This formula is now given by:

$$F_{repulsive} = k_{rep} \cdot \left(\frac{1}{\rho(q)} - \frac{1}{\rho_0} \right) \cdot \frac{1}{\rho(q)^2} \cdot \frac{q - q_{obstacle}}{2}$$

where k_{rep} is the repulsive coefficient, $\rho(q)$ is the shortest distance between the obstacle and the robot, q is the position of the robot and $q_{obstacle}$ is the position of the obstacle. This is not the original formula from the theory, where the last $\frac{1}{2}$ is replaced by $\frac{1}{\rho(q)}$. The reason is explained right below.

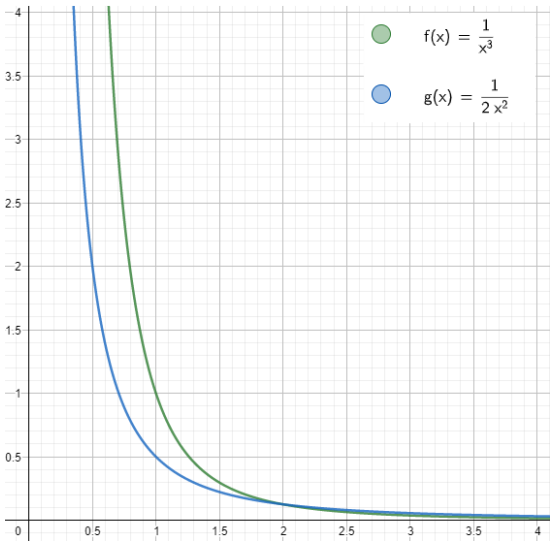


Figure 3.6: Comparison of the slope of the two functions.

Map modelling

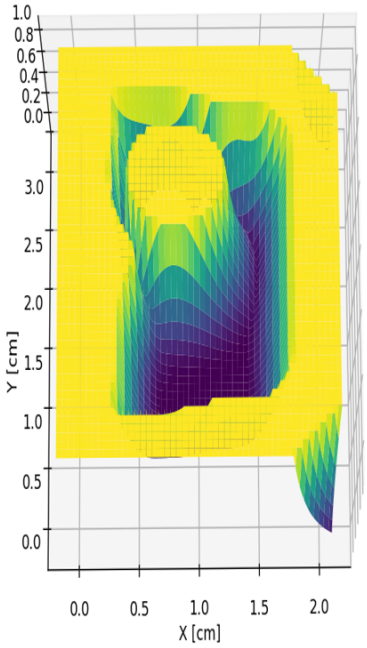
When the code produced in the LELME2732 course and working in a virtual environment was integrated in Tutankhabot, the robot showed heavy oscillations when dealing with an opponent. They were already present in the simulation, however, if the virtual robot took them well, in reality, the oscillations were too strong and it was impossible for the real robot to go around its opponent.

After some visual tests, it appeared clearly that the slope of the force was too steep. Very roughly speaking, the original formula had a $\frac{1}{x^3}$ slope, whereas the new formula[1] has a $\frac{1}{2x^2}$ slope. The difference between the two is shown in Figure 3.6.

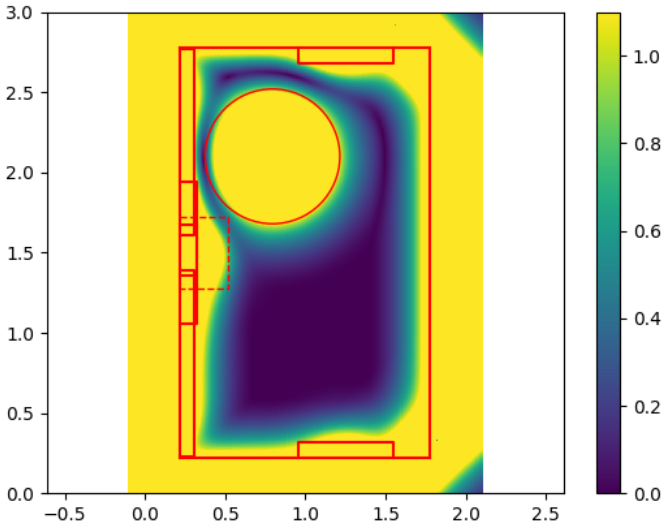
The slope of the blue curve is indeed gentler than the green curve in the range $x \in [0.5; 2.5]$, which is exactly what is wanted. With the rest of the function acting globally as multiplicative constant, this gives a slope that increases more slowly.

But that was not the only problem. Initially, the map was modelled as having borders represented by an infinite line and the rest of the obstacles by a series of points. Better results were obtained in the regions where many points were present in the computations, whereas at the borders, the force was still rising way too quickly. So, the map was remodelled by hard-coding its outline by points 5 cm apart[2]. The result was that, by taking every point in a certain radius, computing the repulsive force due to them and then summing everything, the furthest points helped to smooth the curves. The obtained results are shown in Figure 3.7a and Figure 3.7b.

[1] Taken, with permission, from group 5.
[2] Once again, many thanks to Simon Van Roy from group 5.



(a) 3D plot of the force seen by the robot at each point. The presence of an opponent is modelled by a circle.



(b) 2D plot of the force seen by the robot at each point. The red lines indicate the "real border" of the map: it is modelled as the border plus taking into account the radius of the robot. The red circle models the radius of the robot + the radius of the opponent.

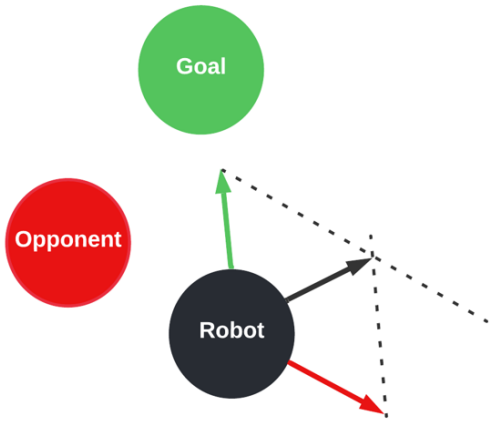
Figure 3.7: Results obtained with the coefficients presented in Table 3.4 and with the force limitation explained in subsection 3.1.3.

Force limitation and speed adjustment

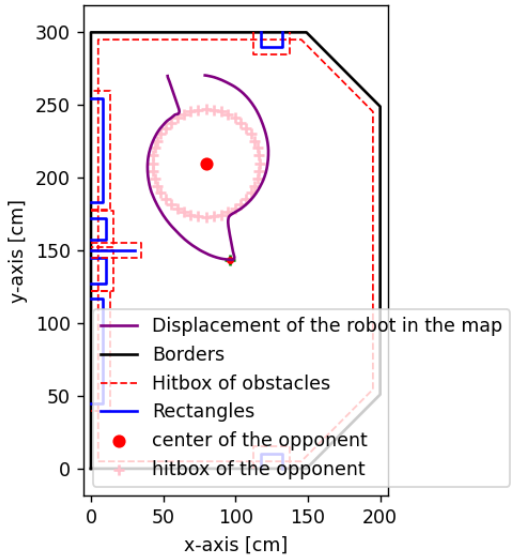
These changes alone were not enough, however. The weight and distance of influence of the obstacles had to be tuned in order to get smooth results. The final results are given in Table 3.4. Moreover, to avoid a strong oscillating behaviour, the attractive force is limited to at most 1 N and the repulsive force to 1.1 N. This way, when summing up the two vectors to get the final force acting upon the robot, one force is not taking over the other one too harshly and if both forces saturate, the resulting direction of force will be oriented globally perpendicularly to the two force vectors, as shown in Figure 3.8a. Finally, a speed limitation has been implemented so that when the robot gets closer to the opponent, it slows down, thus reducing further the possible oscillations and avoiding getting a point penalty for running at full speed into the opponent.

Table 3.4: Final weights (k_{rep}) and distances of influence (ρ_0) of the obstacles.

Type of obstacle	k_{rep} [/]	ρ_0 [m]
Map border	0.019	0.6
Opponent	0.02	0.8



(a) Illustration of the force composition. The green vector indicates the attractive force due to the obstacle, the red vector the repulsive force due to the opponent. The black one is the summation of the two, indicating the force acting on the robot. Both forces are assumed to be at their maximum value.



(b) Displacement of the robot on the map when confronted to a hardcoded obstacle straight in its path to the goal. Tutankhabot goes from the base to the goal and back to the base.

Figure 3.8

Obtained results

After these modifications, the robot now has a smoother behaviour on the map. Its displacements avoid obstacles and the opponent. Using these links, you can see videos of the robot's behaviour when confronted to an opponent (here, a box put in front of it) or to an obstacle hardcoded on the map. The resulting path when confronted to an obstacle that is encoded in the program is shown in Figure 3.8b.

3.2 Function 1: opponent detection and avoidance

3.2.1 Description

Since the matches in the context of the Eurobot contest are contactless, the robots must be able to detect and avoid the opponents. That's why Tutankhabot is equipped with a LiDAR that detects the beacon support of the opponent, since the LiDAR is itself located at this position on Tutankhabot. Regarding avoidance, it has been described in subsection 3.1.3. The LiDAR is placed on the top of Tutankhabot on a support that rectifies the angle of the robot, as it can be seen in Figure 3.1. Indeed, due to the difference of height between the motorised wheels and the freebear wheels, the whole robot is tilted by 1.2°. This angle can pose a problem to detect the beacon's support of the opponent, that's why the support corrects this angle. The position of the LiDAR in the robot implies also that it is at the same height as the robot beacon's support. To be able to detect opponents, holes have been made in the beacon's support. However, and since the beacon's support must be higher than the LiDAR, there are small supports to maintain it. As explained subsection 3.1.3, this creates dead zones in the vision of the LiDAR.

3.2.2 Experimental characterisation

To characterise this feature, A trial and error procedure has been realised. After a tuning of the parameters as explained in subsection 3.1.3, the robot is able to detect at any time opponents on the map, and able to avoid them when they are near to the robot, with a security margin.

3.2.3 Improvements

There are two possible improvements for this function. The first one is to think about a new beacon's support that has smaller supports than the current one in order to minimize the dead zones. The second improvement is to put the Lidar in another place inside Tutankhabot. Indeed, if it was placed at the lower floor of the robot, the LiDAR could have been able to detect the walls and map's elements in addition to the opponents. However, there would have been the same problem of dead zones as with the current design.

3.3 Function 2: statuette

3.3.1 Description

Our implementation for this function is composed of a parallel gripper operated by a servomotor. This servomotor is controlled by an Arduino Mega. Concerning the statuette, we have made cuts in the lower part of it so that it does not slide vertically. We also made sure that these cuts have an angle, as can be seen in Figure 3.9. This has the advantage that when the robot closes its gripper to take the statuette, this latter rises about 1 cm. This allowed us to save one degree of freedom for the gripper and thus keep the gripper design simple, since it should not be able to move up and down vertically. Being able to raise the statuette by 1 cm allows the robot to be sure not to hit the bottom of the display cabinet when placing the statuette in it, but it also allows the robot to be robust to small differences in level between the display cabinet and the pedestal.

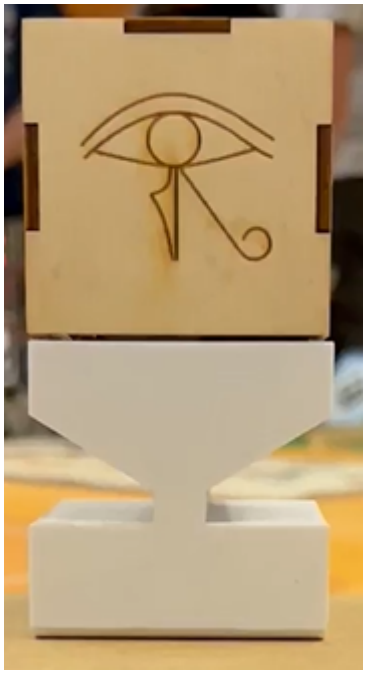


Figure 3.9: The statuette.

3.3.2 Experimental characterisation

The time taken to achieve this action is 8.5 s. This time takes into account the positioning of the robot and the taking of the statuette. Then, we analysed the reliability of the action by running it 10 times. The robot succeeded in taking the statuette 9 times, which allows us to affirm that this action works correctly. This good reliability is mainly due to the fact that the opening width of the gripper is much larger than the width of the statuette thanks to the cutouts. This allows the robot to succeed in the action even if it has a slight misalignment.

3.3.3 Improvements

Even if it has a good reliability, there are some points that can be improved. The first one is the centering of the gripper in the robot. Indeed, for now, the gripper is on the right side of the centre when looking at the robot from the front (as can be seen in Figure 2.1). This made it very difficult to transfer positions from one side of the table to the other when we implemented this action. The second point to improve is the fabrication of a guide to place the statuette on the pedestal. This would allow the statuette to be placed correctly and precisely every time.

3.4 Function 3: display cabinet

The structure of our display cabinet is made of wood plates that have been laser cut at Makilab in order to be able to fit them into each other and thus create the desired shape. Then, in order to have an animation of the display cabinet when the statuette is put on it, we have a strip of addressable LEDs controlled by the Arduino Mega as well as a set of infrared transmitter-receiver. The whole thing is powered by a 5 V USB battery that is placed inside the structure of the display cabinet. The system is quite simple, as long as the infrared beam is not blocked, the LEDs remain off. It is only when the statuette is placed in the display cabinet that the beam is blocked and the animation of the LEDs starts. The animation does not stop as long as the statuette is there, as we can see in Figure 3.10, or as long as the emergency button is not pressed.

3.4.1 Experimental characterisation

This function worked 7 times out of 10 mainly because of alignment problems of the robot with the display cabinet. It takes 11 seconds to do. This time takes into account the positioning of the robot and the placement of the statuette in the display cabinet.

3.4.2 Improvements

In order to improve the reliability of this function, the display cabinet could be wider. This would allow the robot to have more room for manoeuvre and small alignment error. We could also enlarge the detection area in order to be able to place the statuette further in the display cabinet. Finally, with the good weather we have had these last few days, we realized that the system did not work when there is too much sun. Indeed, the sun emits an electromagnetic radiation in which are notably the infrared, which creates interferences with the sensor. This has as consequence that the LEDs do not light up

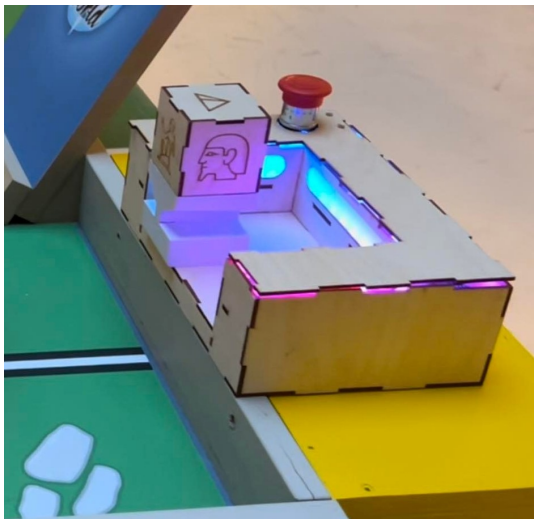


Figure 3.10: The display cabinet.

when the statuette is placed. To solve this problem, it would be necessary to add a roof to protect the sensor from direct light.

3.5 Function 4: replica

The implementation of this function relies on a system of rack and pinion, as can be seen in Figure 3.11 or in a more distinct way in Figure 2.5. A 16 cm-rack is driven by the pinion, which is directly actuated by a Dynamixel controlled by an Arduino Mega. At the start of the match, the replica is placed on a support and when the robot is aligned with the workshed, the Arduino gets the information from the Raspberry Pi to push the replica on the pedestal by actuating the Dynamixel.

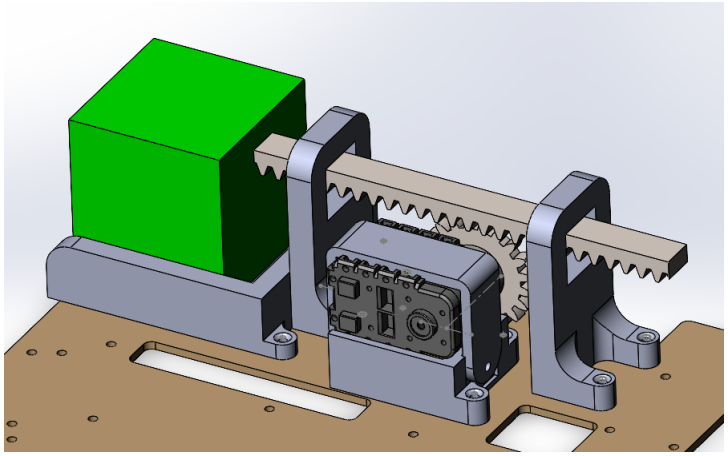


Figure 3.11: Replica mechanism.

3.5.1 Experimental characterisation

The time taken to achieve this action is 10.2 s. This time takes into account the positioning of the robot and the pushing of the replica on the pedestal. Then, we analysed the reliability of the action by running it 10 times. The robot succeeded in pushing the replica 8 times, which allows us to affirm that this function works quite correctly. This good reliability is mainly due to a good positioning of the robot when dropping the replica. The success of this action is also due to the fact that the replica is relatively small compared to the pedestal, which can make up for mistakes in positioning. The reliability would have been better if the whole mechanism was placed one floor lower so that the replica wouldn't have fallen from that high, as depicted in Figure 3.12.

3.5.2 Improvements

The major improvement that can be made to increase the reliability of this action is to place the whole mechanism one floor lower so that the replica doesn't fall when it is pushed. However, this is not easy because most of the other mechanisms are already on this floor. In fact, the mechanism was on this floor when Tutankhabot was first designed in 3D, but due to the placement of some electrical components, this mechanism was delocalised one floor higher. So the most interesting

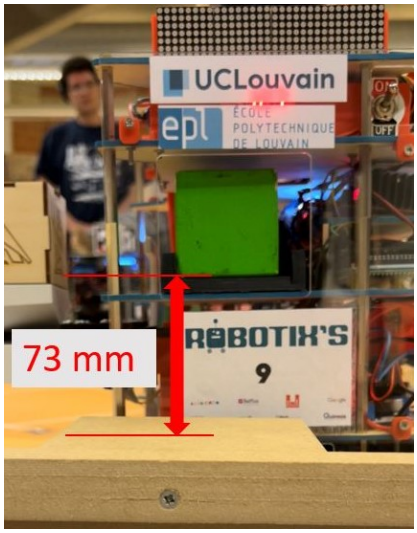


Figure 3.12: Difference of height between the pedestal and the replica in the robot.

solution would have been to opt for a system like a flipper that takes less space than a rack and pinion. This way, the support of the replica would have had the possibility to level the replica at the perfect height to optimise the reliability of the action. The last thing to improve concerns the wiring, indeed some problems of reliability have been encountered due to the fragility of the standard wires to control and power the Dynamixel.

3.6 Function 5: excavation squares

3.6.1 Description

To realise this function, Tutankhabot relies on two subsystems that are represented in Figure 3.13. The first one consists of two probes to measure the resistance of two excavation squares, as it can be seen in Figure 3.14. These probes are each composed of a piece of flexible copper that are connected to an Arduino Mega that can determine the resistance thanks to a voltage divider. The flexibility of these copper parts is very useful to compensate a slight misalignment during the measurement. These filaments of copper are fixed on a 3D printed piece that rotates thanks to a servomotor controlled by an Arduino Mega. The measurement of these two resistances allows Tutankhabot to know which excavation squares it can flip with the second subsystem. The second subsystem is a rack and pinion actuated by a Dynamixel which is also controlled by an Arduino Mega. This subsystem is on the bottom left of Figure 3.13. Due to the lack of time to improve this mechanism but also due to the limit of a match duration, an alternative mechanism has been found to flip the excavation square that always belongs to the team (so it doesn't need to be measured). This method consists of pushing this excavation square by using the gripper of the statuette that protrudes from the robot.

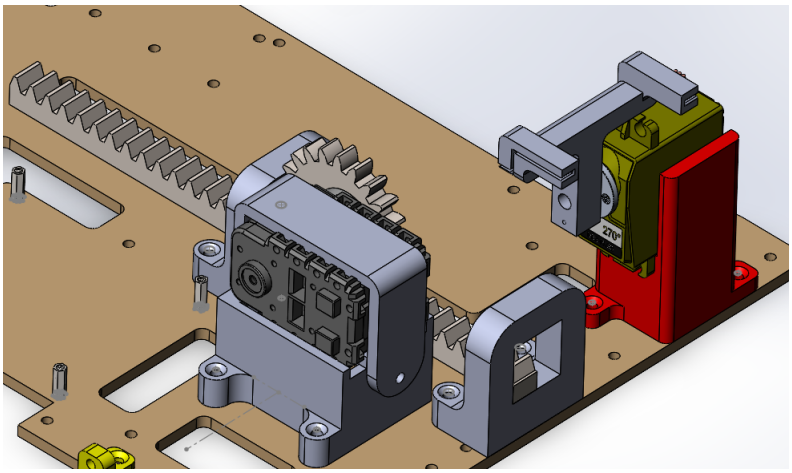


Figure 3.13: System for the excavation squares.

3.6.2 Experimental characterisation

Since the mechanism of probes and rack and pinion was not reliable enough because the team didn't have enough time to improve it and also because the realisation takes quite a lot of time, this mechanism has never been used in a match. However, from the few tests that have been carried out, this mechanism could have been totally sustainable after some

Figure 3.14: Probes that measure a resistance.

tuning. As explained in the description of the function, an alternative mechanism has been put in place by using the gripper of the statuette. This mechanism was reliable since it didn't need a lot of precision.

3.6.3 Improvements

Since the space inside Tutankhabot is a major concern, a possible improvement of the mechanism composed of the rack and pinion could have been replaced by a simple flipper. Another issue encountered that needed to be improved is once again linked to the wires of the Dynamixel that created a lot of faulty contacts.

3.7 Function 6: dynamic score

3.7.1 Description

For this function, we have a screen composed of 4 matrices 8x8 of LEDs. It is controlled by the Arduino Mega through a MAX7219 controller. This screen allows us to display the robot's score live, as can be seen in Figure 3.15, but also to increment it live after having performed an action.

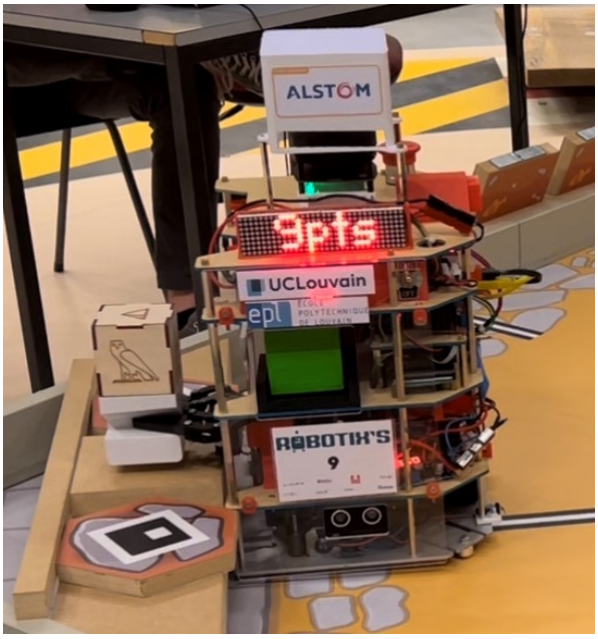


Figure 3.15: The screen.

3.7.2 Experimental characterisation

Due to wiring problems, we had a lot of problems with this screen. But by changing the strategy to power it through digital ports of the Arduino Mega, the screen is now working fine. For the reliability of the function, however, we do not reach a 100% reliability. Indeed, the score displayed is, in some situations, false. It is for example the case because of an action that is missed by the robot but since there is no verification of the fact that the action is well done, the score is still incremented with the score of this action. Finally, we observed a reliability of 6/10.

3.7.3 Improvements

To resolve our problems for this action, we have to implement a verification that the action has been performed correctly and then, only if it is the case, increment the score.

CHAPTER 4

Electronics

The technical aspects of this chapter were developed by Diego and Nicolas. This chapter was written by Nicolas, Diego contributed for the section 4.4.

4.1 Global electrical scheme

In this section, the global electrical scheme is explained in detail. Any changes with respect to the technical report [3] are mentioned and justified.

Overview of the schematic The global electrical scheme of Tutankhabot is given in Figure 4.1.

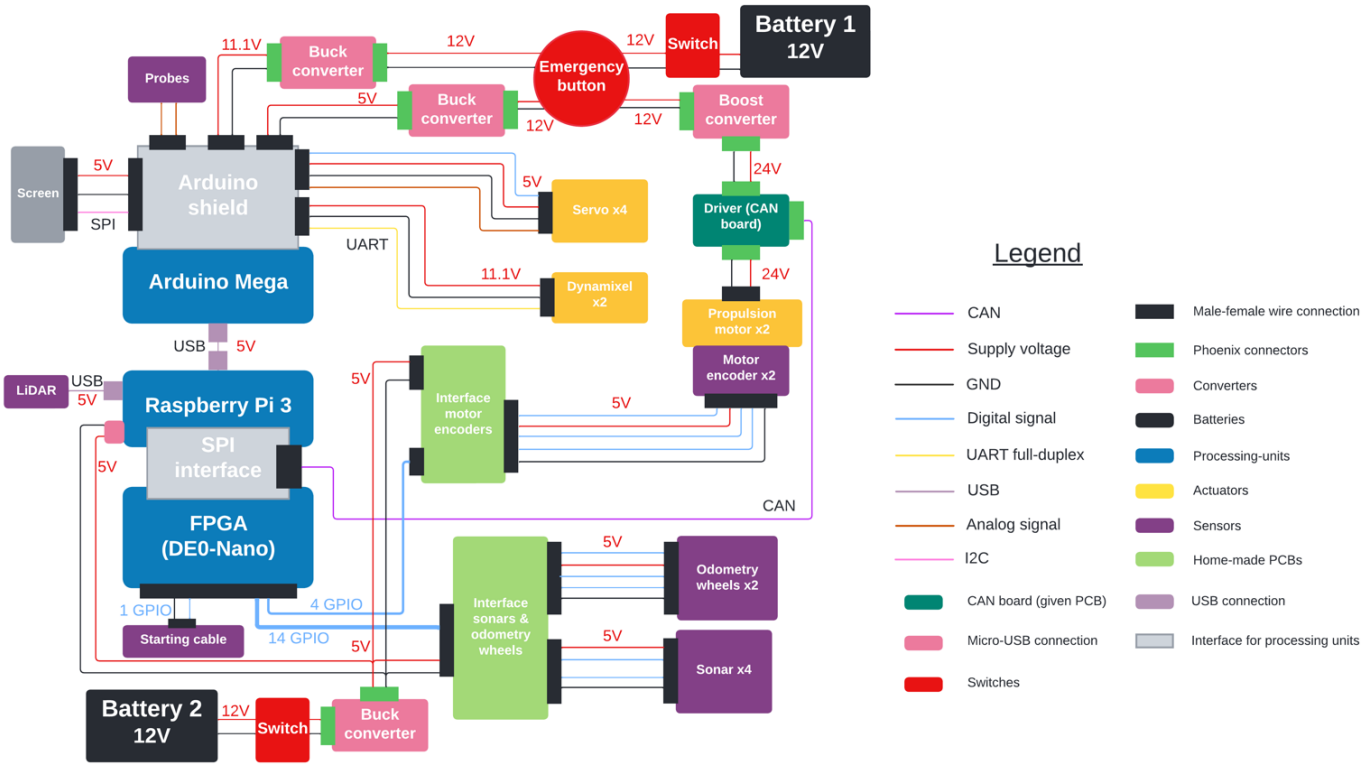


Figure 4.1: Global electrical scheme of the robot.

Many things have changed with respect to the global electrical scheme shown in the technical report [3]. These changes appeared as the assembly of the robot was ongoing. These changes concern:

- the screen
- the probes
- the Arduino board
- the home-made PCBs
- the addition of switches and a starting cable
- the total number of sensors

In the following paragraphs, each aforementioned change is specified.

The screen

The original idea was to use an I2C screen connected to the FPGA, which would in turn communicate with the Raspberry-Pi in SPI. However, the ordered screen came with no instructions and didn't use standard communication pins. Therefore many difficulties were encountered when trying to program an I2C communication on the FPGA.

It has been decided to reuse a LED matrix module from an old project as the screen. Its communication has also been moved to the Arduino for ease of use. By doing so, the part of the *Interface sonars & odometers* PCB converting the 5 V logic level used by the screen to a 3.3 V logic level for the DE0-Nano (or the other way around) is no longer useful[1]. Moreover, this new screen uses now SPI instead of I2C.

The probes

It has been decided that it was more compact to put the probes on the Arduino board and to use the integrated ADC to measure the resistances instead of going through the FPGA, the Raspberry-Pi and then the Arduino board (which controls the actuators). This is why the *Interface probes* PCB is no longer used.

The Arduino board

The original Arduino Nano board has been replaced by an Arduino Mega board. This is due to the fact that the screen, the probes and the Dynamixels were now directly connected to the board. Indeed, the PCB called *Interface motor encoders* was originally intended to perform a conversion between the full-duplex UART communication protocol coming from the Arduino and the half-duplex UART communication protocol coming from the Dynamixels. Though the Dynamixels are used for simple, short and relatively rare actions in the robot, the feedback from the actuators is therefore not critical. The PCB is then no longer required and data can be sent one-way directly from the Arduino. It allows for a more robust setup, as we know that self made PCBs can be less reliable than commercially available hardware.

Another advantage of the Arduino Mega board is that it offers multiple serial ports [6]. The Arduino uses serial to communicate with the Raspberry Pi, which would take the one and only serial port of the Arduino Nano. The Mega allows for simultaneous communication between the Pi and the Arduino and between the Arduino and the Dynamixels by using two different serial ports.

To be sure that the Arduino had enough pins, a shield has been used to make the connections. This shield is then plugged into the pins of the Arduino. The data enters and is sent through this shield, and it also does the power management for the actuators. The power management of the whole robot will be explained in further details in section 4.3.

The addition of switches and starting cable

The starting cable was added to meet the requirements for the Eurobot contest. Moreover, switches were added to safely cut the power supply to the rest of the robot. Two switches are used : one after the battery supplying the actuators and propulsion motors and one after the battery supplying all the sensors and processing units. This allows to shut down cleanly the actuation part and the digital part of the robot independently.

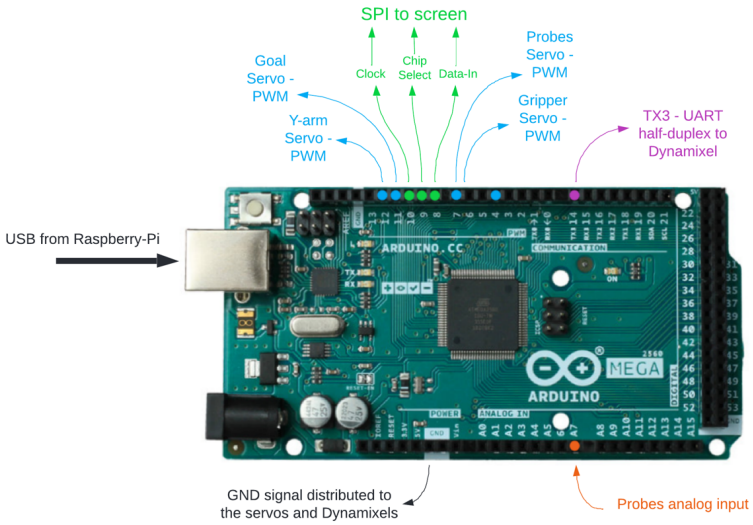
The total number of sensors

Originally, 5 sonars were planned. But, as already explained in subsection 3.1.1, when testing the sonars, it has been decided that they were of no use. Since the pins used on the DE0-Nano by the already assembled sonars were not needed and as the tests were done using only 4 sonars, the fifth one has never been added to the robot but the connections to the board remain.

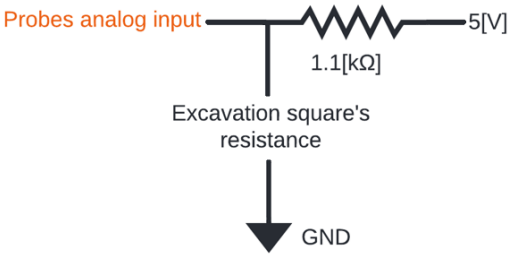
4.1.1 Zoom on the Arduino board

The Arduino board used is The Mega 2560 board [7]. The connections on the board are summarised in Figure 4.2a.

[1] As for the *Interface motor encoders* later, this is why their names have changed to only mention what is used in the interface.



(a) Schematic of the connections on the Arduino Mega board.



(b) Voltage divider implemented on the shield.

Figure 4.2

All these signals, except the USB from the Raspberry-Pi that supplies the board and allows serial communication between the two, go through a shield first. The shield acts as a relay between the Arduino and the outside world for a simpler and cleaner cable management. As already explained, the communication between the board and the Dynamixels goes only one way, and the same goes with the servos which position control is done internally. These are controlled via Pulse Width Modulation. The GND signal from the Arduino is distributed to the Servos and the Dynamixels to have a common ground. The analog input is connected to a voltage divider implemented with a 1.1 kΩ resistance on the shield that is itself connected to a 5 V signal, as visible in Figure 4.2b.

4.2 Description of the home-made functional blocks

Originally, there were three PCB boards that were designed for the project. By now, only two are used and many parts of the remaining PCBs are no longer used, as already explained in section 4.1. This section goes over a more detailed description of each PCB and justification of the used chips. Only a global view of each schematic is shown in this report: for a more detailed viewing of the schematics, please refer to section 8.1.

Overview of the connections

The connections between the DE0-Nano and the peripherals of the robot are shown in Figure 4.3. All interfaces are connected to JP2. The connections between the boards and the FPGA are made with jumper wires and Dupont connectors.

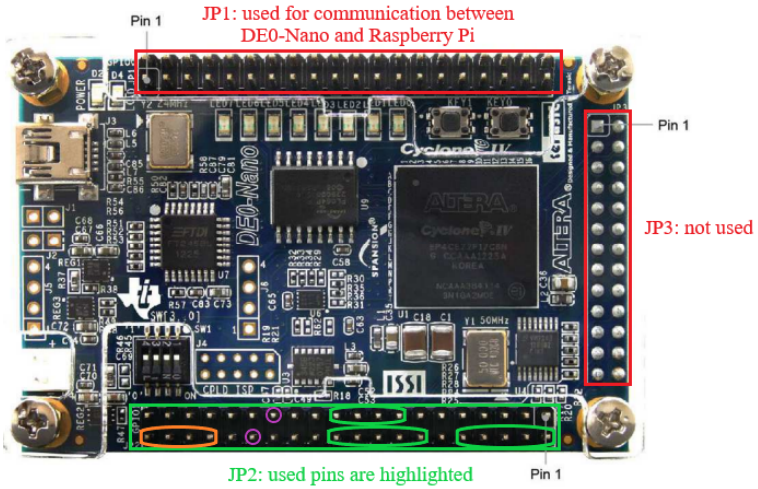


Figure 4.3: View of the connections on the DE0-Nano board. On JP2: in orange: pins used by the interface 1; in green: pins used by the interface 2; in purple: pins used for the starting cable.

4.2.1 Interface 1: *Interface motor encoders*

The schematic of the interface is given in Figure 4.4.

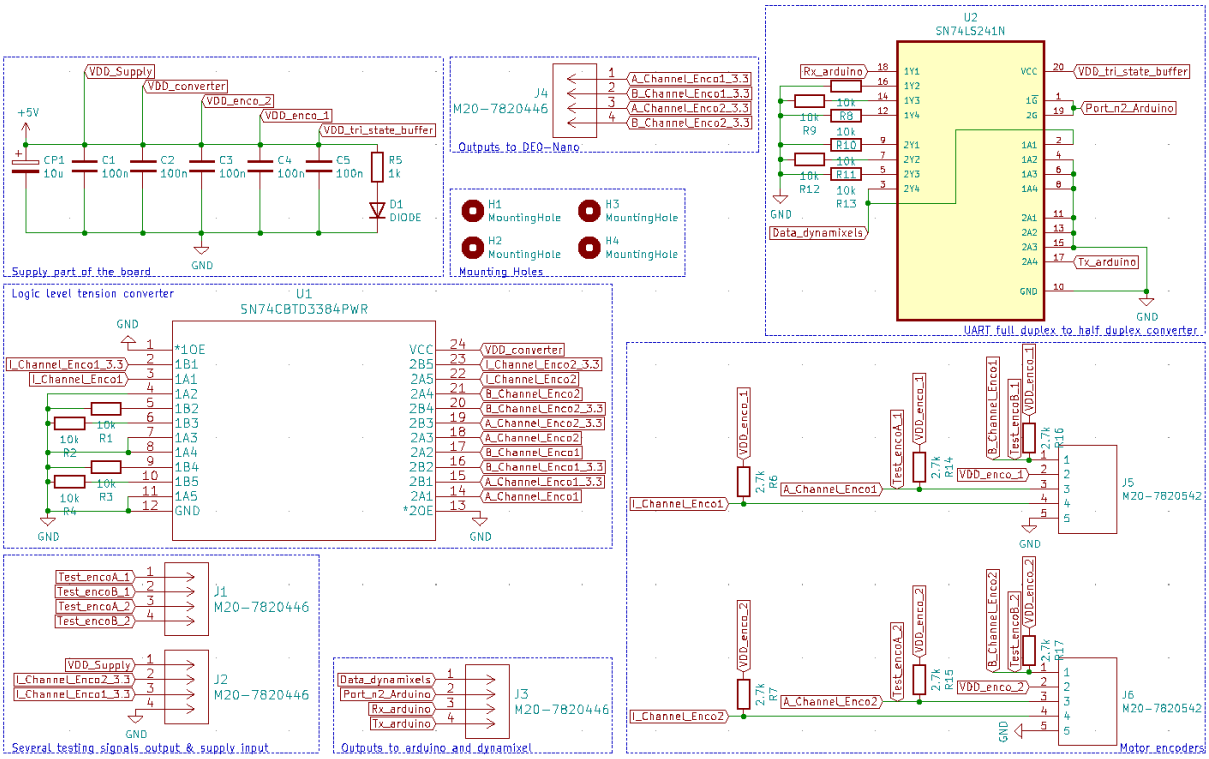


Figure 4.4: KiCad schematic of the PCB board *Interface motor encoders*.

The board is supplied by a 5 V voltage, which in turns supplies the motor encoders with the required 5 V and GND signal.

The encoders

Each signal channel (A, B and I) has a pull-up resistor of 2.7 kΩ to avoid floating signals, as required by the datasheet, because the lines are normally high.

The main chips

- **SN74CBTD3384PWR**: this chip [8] ensures that data coming from the encoders do not damage the DE0-Nano by converting the logic level from 5 V to 3.3 V.
- **SN74LS241N**: as explained in section 4.1, the PCB does not need anymore to perform the communication between full-duplex and half-duplex UART, which this chip is used for. Please refer to section 8.1 for an explanation of the chip's working.

Connections to the FPGA

This PCB has 4 GPIO of output. The I signals from the encoders are not used, as they were deemed not needed for the computations. A jumper wire of 4 pins' large is used to connect the board to the FPGA. The connections are summarised in Table 4.1.

Table 4.1: Summary of connections between the PCB *Interface motor encoders* and the FPGA.

Devices	Signals on PCB	Pins on FPGA
Encoders 2x	A_Channel_Enco1_3.3	GPIO_133
	B_Channel_Enco1_3.3	GPIO_131
	A_Channel_Enco2_3.3	GPIO_129
	B_Channel_Enco2_3.3	GPIO_127

4.2.2 Interface 2: *Interface sonars & odometry wheels*

The schematic of the interface is given in Figure 4.5.

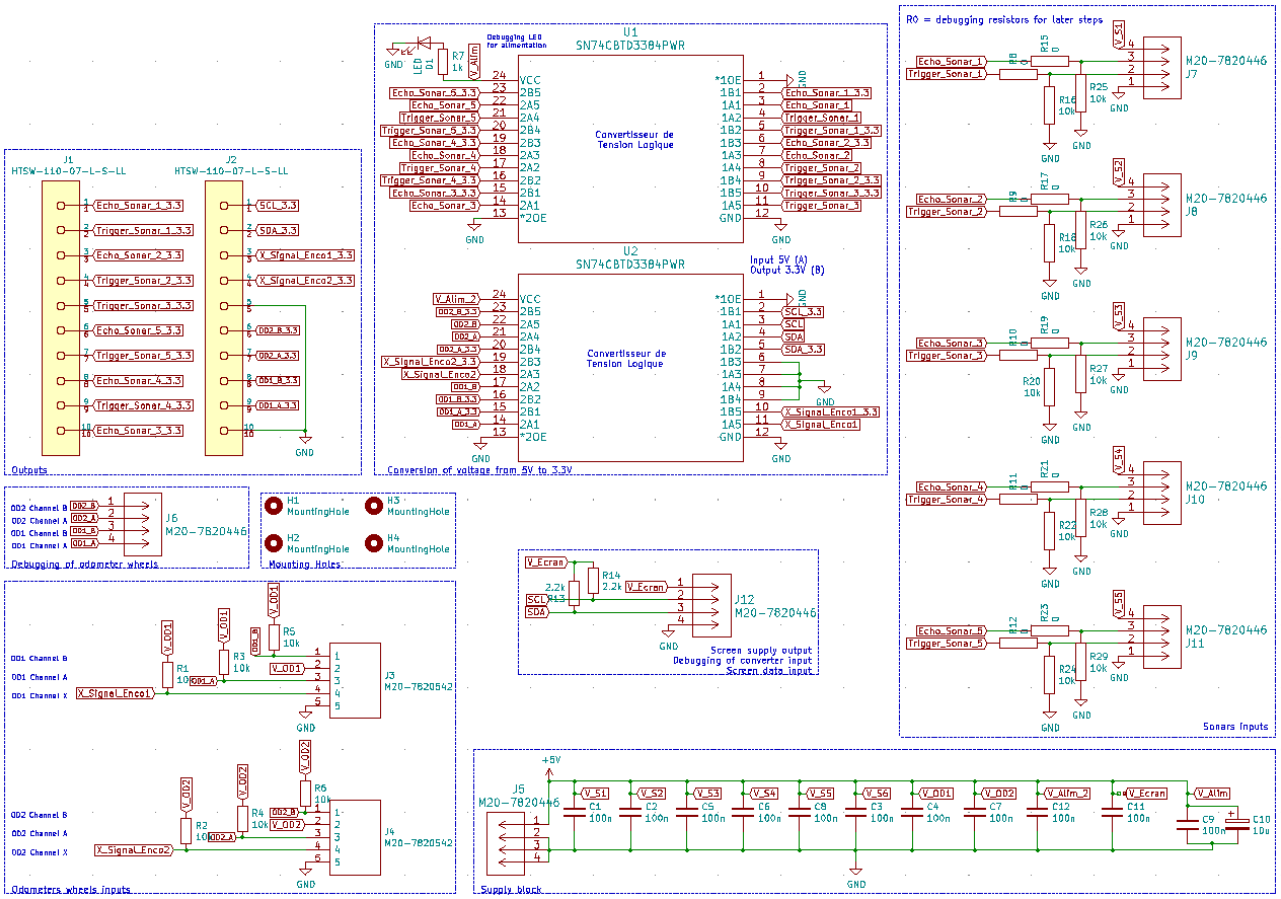


Figure 4.5: KiCad schematic of the PCB board *Interface sonars & odometry wheels*.

The board is supplied by a 5 V voltage and distributes this voltage to the sonars and odometry wheels, alongside the GND signal. Please note that although the sonars are not used in practice, they are described in the main report and not in section 8.1 because they are physically connected to the FPGA and the PCB board and could be used if deemed necessary[2].

The sonars

Five sonars were planned in the robot. As already explained, the fifth one was never added on the robot for ease of use, but all the connections are available and working. Pull-down resistors of 10 kΩ are used on the echo and trigger signal lines, as the lines are normally low. 0 kΩ resistors were added on the line for debugging matters.

The odometers

Just like the encoders, each signal channel (A, B and X) has a pull-up resistor of 10 kΩ to avoid floating signals, because the lines are normally high.

The main chips

Only one type of chip is used, twice, on the PCB: the **SN74CBTD3384PWR**, already presented in subsection 4.2.1, used to protect the FPGA from the 5 V logic level signals coming from the sonars by converting them in 3.3 V signals (for the ECHO signals) and by converting the TRIGGER signal sent by the FPGA at 3.3 V to a 5 V logic value as the sonars work properly at that logic level.

Connections to the FPGA

This PCB has 14 GPIO of output. Just like for the encoders, the X signals from the odometry wheels were deemed not needed. Two jumper wires, one of 11 pins' large for the sonars (to account for the GND pin in the middle of the pins used

[2]Unlike the *Interface probes* which has simply been removed or the Dynamixel part of the first interface, which is not physically connected to anything.

for the sonars on the FPGA) and one of 4 pins' large for the odometry wheels signals are used to connect the PCB to the FPGA. The connections are summarised in Table 4.2.

Table 4.2: Summary of connections between the PCB *Interface sonars & odometry wheels* and the FPGA.

Devices	Signals on PCB	Pins on FPGA
Odometry wheels 2x	OD2_A_3.3	GPIO_110
	OD2_B_3.3	GPIO_112
	OD1_A_3.3	GPIO_114
	OD1_B_3.3	GPIO_116
Sonar 1	Echo_Sonar_1_3.3	GPIO_115
	Trigger_Sonar_1_3.3	GPIO_117
Sonar 2	Echo_Sonar_2_3.3	GPIO_111
	Trigger_Sonar_2_3.3	GPIO_113
Sonar 3	Echo_Sonar_3_3.3	GPIO_19
	Trigger_Sonar_3_3.3	GPIO_10
Sonar 4	Echo_Sonar_4_3.3	GPIO_11
	Trigger_Sonar_4_3.3	GPIO_13
Sonar 5	Echo_Sonar_5_3.3	GPIO_15
	Trigger_Sonar_5_3.3	GPIO_17

4.2.3 Interface 3: *Interface probes*

As this interface is no longer used, its description can be found in the Appendices in section 8.1.

4.3 Description of power management in the robot

The robot is supplied by two $LiFePO_4$ batteries of 12 V and 3200 mAh. The division is made so that one battery supplies all the motors and actuators and the other battery supplies the digital part, including the PCB boards, the sensors and the processing units.

The digital part works at 5 V, so a buck-converter is used to convert the 12 V from the battery to the required 5 V, which is then distributed to both the Raspberry-Pi (via micro-USB) and the home-made PCBs (through a jumper wire), which then supply the motor encoders for interface 1 and the odometry wheels and sonars for interface 2. The Raspberry-Pi supplies the DE0-Nano board through their interface and the Arduino Mega via a USB connection, which is also used to transfer data serially. The Arduino itself then supplies the LiDAR, the screen and uses the 5 V it receives from the Raspberry-Pi for the voltage divider of the probes.

The actuators and motors part is first connected to the emergency button, which ensures an easy stoppage if the robot starts going mad. The voltage is then distributed to three converters: two bucks and one boost. The boost drives the 12 V voltage to a 24 V voltage to supply the driver, which in turns supplies the motors with the appropriate voltage. One buck converts the 12 V to 11.1 V to supply the Dynamixels, while the other one converts it to 5 V to supply the Servo motors. Both the 11.1 V and 5 V go through the Arduino shield, which acts as a central relay of the power.

As already stated in section 4.1, both batteries are separated from their respective part of the circuit by a switch, which allows to safely turn on and cut the power supply when at rest.

The rest of this section goes over an estimation of the energy consumed by the robot.

- **Methodology**: first, a baseline when nothing is activated has been measured. The current consumed is then measured in series with each sub-circuit. Using the baseline, the current consumed by each part is obtained. Furthermore, some actuators seemed only to provoke temporary peaks of current, e.g. for the servo opening the gripper. This nuance is indicated in the caption of the result table. Then, knowing the terminal voltage of each actuator, the measured power is obtained and then integrated over the activity time of each part to get the consumed energy.
- **Results**: they are visible in Table 4.3.

As the batteries supplying the robot can output 3200 mAh, the total energy of one battery can be found:

$$E_{battery} = 12[V] \cdot 3200[mAh] \cdot 3600[s] = 138240[J] \tag{4.1}$$

Using the total energy from each part, it is found that Tutankhabot's battery for the motors and actuators part can last around 31 matches and the one supplying the digital part can last 170 matches.

Table 4.3: Summary of the power consumption of Tutankhabot. For the actuator and motor part, for the current, a red number indicates that the value measured is valid only during displacement and a blue number indicates that the value in the table stands for all the duration of the action.

Robot's part	Devices	V_{supply} [V]	$I_{measured}$ [A]	$P_{measured}$ [W]	Time [s]	Energy [J]
Actuators and motors	Goal Servo	5	0.4	2	5	10
	Probes Servo	5	0.25	1.25	3	3.75
	Y-arm Servo	5	0.11	0.55	3	1.65
	Gripper Servo	5	0.05	0.25	2	0.5
	Dynamixel replica	11.1	0.04	0.444	2	0.888
	Dynamixel excavation square	11.1	0.09	0.999	8	7.992
	Propulsion motors	24	0.654	15.696	280	4394.88
	Total					4419.66
Digital part	Raspberry-Pi, FPGA, Arduino & sensors	5	0.580	2.9	280	812

4.4 Specific description of the functionalities implemented in the FPGA

The following subsection details the modules implemented in the FPGA. A global view of the code architecture is shown in the Figure 5.1.

The FPGA was dedicated for the sensing tasks of the robot, because these tasks, although not very computationally demanding, need to be constantly activated, and using a thread for each of them in the FPGA would be a waste of computational resources. The number of GPIO pins needed to connect the sensors was another advantage of the FPGA, as there were not enough pins on the Raspberry-Pi alone.

Two sensing tasks are not in the FPGA:

- **The LiDAR**: neither the connection which needs to be in USB nor the input/outputs (return a list of distances and angles) of this sensor are compatible with the FPGA.
- **Reading of the resistance**: this module was transferred to the Arduino to decrease the number of cables in the robot.

Two other modules not directly connected to the sensing tasks are implemented in the FPGA.

- **The SPI module**: which permits data flow between the Raspberry-Pi and the FPGA.
- **The pre-computation for the encoders and odometry wheels**: because having it on the FPGA gives a better robustness in time (the clock in the FPGA is exposed to less variation and the clock to generate the inputs and the outputs are the same).

Communication with the Raspberry

SPI Address	Data sent to Raspberry (32bits)
0h	Odometer 1 counter
1h	Odometer 2 counter
2h	Encoder 1 counter
3h	Encoder 2 counter
4h	Speed odometer 1
5h	Speed odometer 2
6h	Speed Encoder 1
7h	Speed Encoder 2
8h	{Starting cable, 7'b0 , distance measured by sonar1}
9h	{8'b0 , distance measured by sonar2}
Ah	{8'b0 , distance measured by sonar3}
Bh	{8'b0 , distance measured by sonar4}
Ch	{8'b0 , distance measured by sonar5}

Figure 4.6: Summary of the SPI communication addressing through the interface.

The communication between the FPGA and the Raspberry is done in SPI. The Raspberry being the host of the main loop, it needed to be the master. The transfer happens 5 bytes at a time.

The management of data transfer between the Raspberry and the FPGA is the following: the Raspberry sends a message in SPI containing an address corresponding to the required data in the last byte of the message (the other 32 bits have no importance). Figure 4.6 shows the table of registers' address in the FPGA.

The odometry wheels and encoders

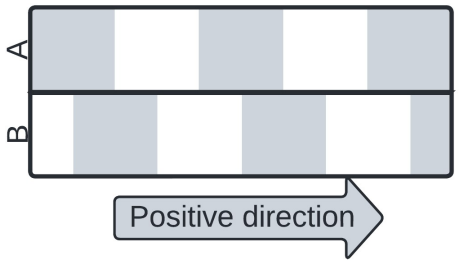
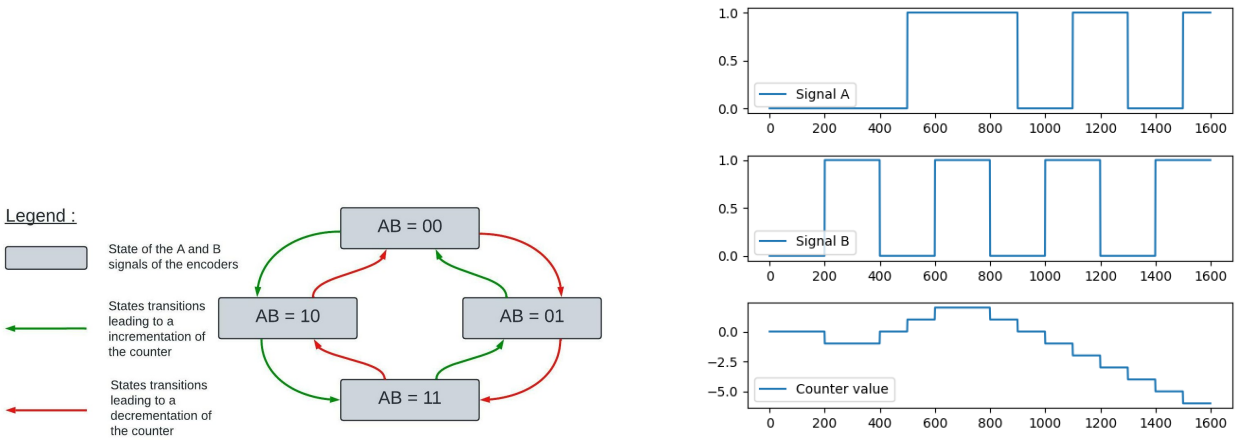


Figure 4.7: Coding of the odometry wheels.

The encoders' and odometry wheels' module returns a signed counter value that is related to the distance travelled based on two inputs signal A and B. They use a special coding to distinguish the direction of rotation. This coding is illustrated in the Figure 4.7.



(a) Encoders and odometry modules' architecture.

(b) Evolution of the counter for given A and B signals.

Figure 4.8

The design of the encoders' module implementation is shown in the Figure 4.8a and an example of the counter evolution is given in Figure 4.8b.

The relation between the counter value and the distance travelled by a wheel is given by the following expression :

$$\textbf{distance} = \frac{\textbf{counter value} \cdot 2\pi \cdot \textbf{Radius of the wheel}}{\textbf{N}_{\textbf{number of color's steps in one turn}}} = \textbf{k} \cdot \textbf{counter value}$$

The k factor being a constant that does not vary over time, it can be determined by a simple test:

- Moving the robot over a given precise distance (generally two to three meters).
- Dividing the distance by the value of the counters.
- Repeating the process and doing an average to reduce errors from non-perfect straight line trajectory, imprecision in the starting and ending positions, and other error sources.

Note: the test must be done on a floor covering as close as possible to the one used in the competition to take into account slippage and the clinging of the odometry's wheels on the covering.

Speeds module

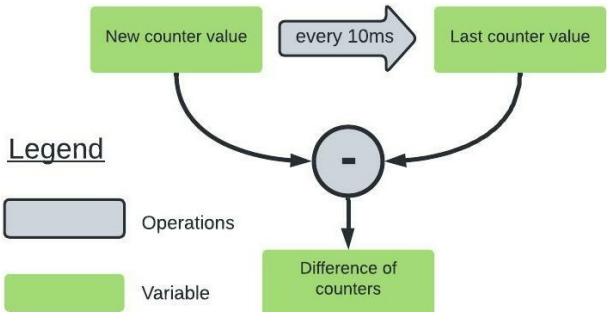


Figure 4.9: Encoders and odometry modules architecture.

The implementation of the speed module is illustrated in Figure 4.9. It compares the values of the encoders' and odometry wheels' counter 10 ms before with the actual value of the counter.

The sonars

The sonars work with two signals: the TRIGGER signal, sent from the FPGA to the sonars, and the ECHO signal, sent by the sonars and received by the FPGA. The trigger signal used in Tutankhabot is 12 µs, as the datasheet requires at least 10 µs high level signal. This signal starts the ranging. It is then followed by an 8 cycle burst of ultrasound and then the module receives the ECHO. This ECHO triggers a high level on the ECHO line, which is used to trigger a counter that will run as long as the ECHO line is high. When it is done, the distance encoded on 32 bits is sent via the SPI interface with the address of the concerned sonar concatenated as the MSB byte. The Finite State Machine used for the sonars is showed and explained state by state in Figure 4.10. The cycle repeats every 61 µs, where a new TriggerLowStart signal is sent to initiate the trigger step.

Explanation of the FSM

The initial state is the STAND_BY state. It is automatically left every 61 µs when a TriggerLowStart signal is sent. The TRIG_LOW state prevents the TRIGGER line from being at a high logic level by forcing it to 0 during 2 µs, after which a TriggerLowEnd signal is sent, allowing the FSM to move on to INTER_TRIG which is only used to reset the general counter of the FSM. For the next 12 µs, TRIGGER is high, and then the TriggerHighEnd signal is sent, stopping the TRIGGER signal and the FSM enters the INIT_MODE state where the counter is once again reset, and it waits for an ECHO signal. If something is measured, then the high logic level time of ECHO is counted in the COUNTING state, which in turn either is followed by the initial state waiting for a new cycle or by an overflow, indicating a wrong measure. In either case, the counter is reset and if the measured distance is valid, it is stored and ready to be sent to the Raspberry-Pi. If nothing is measured, the state is left when a new cycle begins, just like the OVERFLOW state. The FSM uses only one counter, which is reset between each time it is used, and the 61 µs cycle is managed by a counter in the top level module. The architecture of the code in the FPGA will be explained in chapter 5.

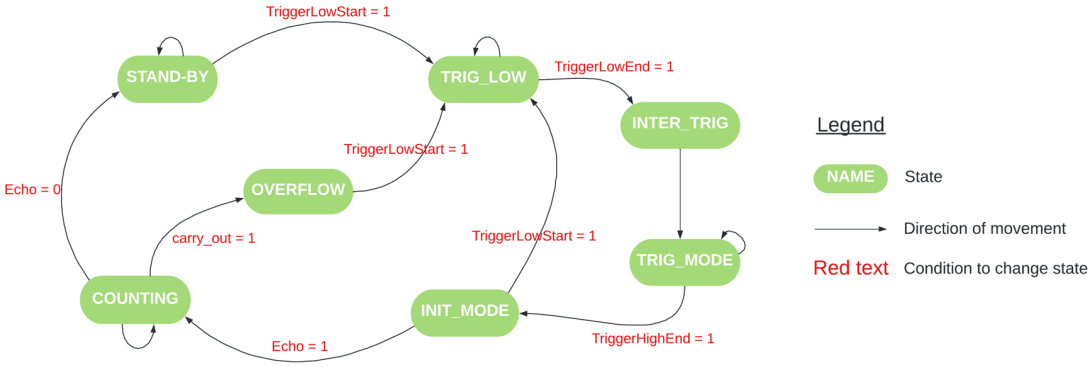


Figure 4.10: Schematic of the FSM controlling the behaviour of a sonar.

Starting cable

The state of the starting cable is sent to the Raspberry Pi via SPI as a binary value, to avoid using a specific address for a one bit register, the starting cable state is sent at the same time as the distance of the first sonar.

CHAPTER 5

Programming

This chapter was globally written by Diego. The section 5.3 was written by François. As this chapter is transversal and connects a lot of parts, the attribution of the tasks' realisation will be done at the end of the section

5.1 Description of the general structure of the program and task management

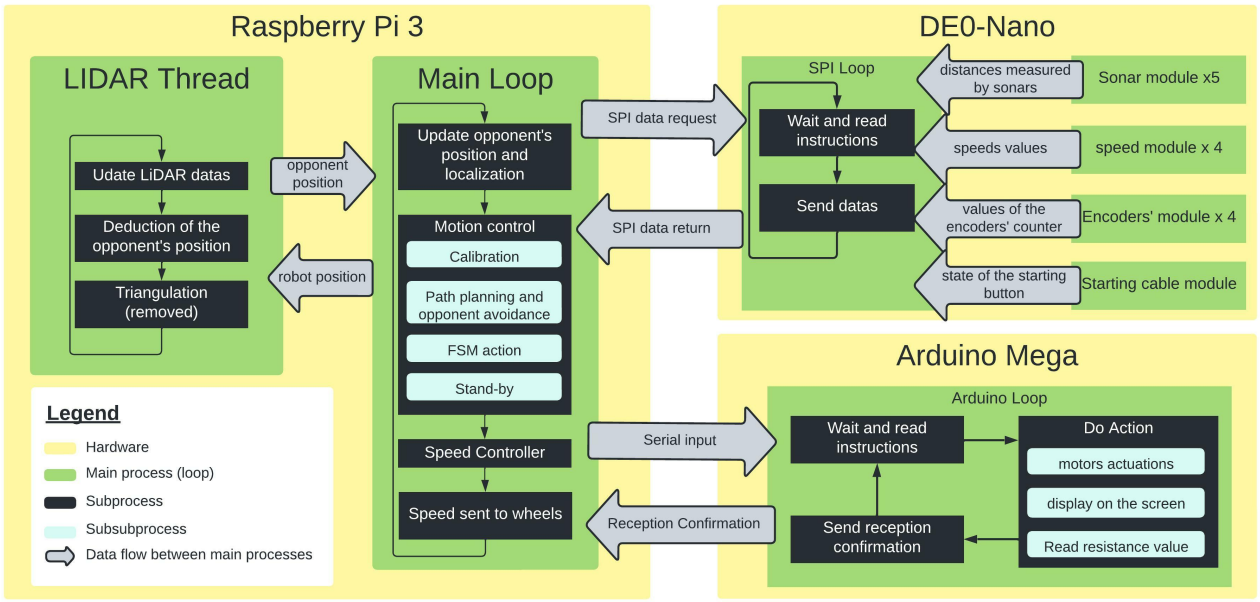


Figure 5.1: Overview of the program's architecture.

As shown in Figure 5.1, the program is shared between three different devices, a Raspberry-Pi, a DE0-Nano (FPGA) and an Arduino Mega. The DE0-Nano is dedicated to the sensing tasks. It permits to take advantage of its high number of pins, high speed processing and its ease of process' parallelism without entering into too complex designs. The Arduino is dedicated to actuators and display (the only exception is the probes' module). It allows to take advantage of the high number of libraries available, simplifying the implementation of Servo motor control and UART and SPI communications.

The Raspberry Pi, used as the central device, runs two main processes:

1. the LIDAR thread receives the data from the LiDAR, takes the list of points seen by it and the current position of the robot and sends the current position of the opponent;
2. the main loop that uses the data received in SPI from the DE0-Nano to perform the localisation task, sends the speed command outgoing from the speed controller to the motor driver, sends to the Arduino in USB the state of the motors and the score to display on the screen.

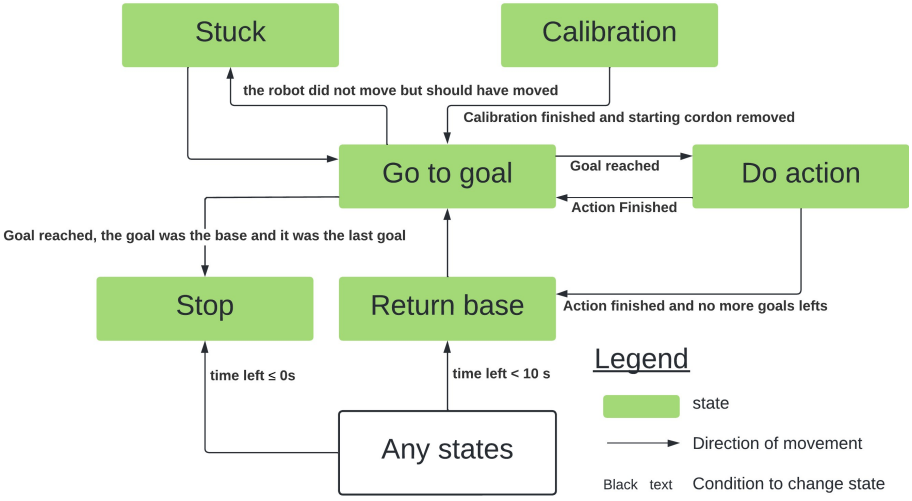


Figure 5.2: Motion Control FSM architecture.

5.2 Motion control

As shown in Figure 5.2 the motion control is implemented as an FSM. This part of the code takes as inputs the localisation of the robot and the localisation of the opponent and, as output, it sends the speed references to the controller expressed in angular and forward speed.

The different states are :

- **Calibration** : initial state, depends on the side of the board (either : purple or yellow), sends the reference speeds by using the Calibration FSM.
- **Go to goal** : sends the reference speeds based on a path planning and trajectory control algorithm (Potential Field), the potential field also ensures the opponent avoidance (see subsection 3.1.3).
- **Do action** : sends the reference speeds by using the Action FSM.
- **Stuck** : changes the current goal to avoid being stuck for too long. During this operation, the reference speeds are equal to zero.
- **Return base** : sets the current goal to the base. During this operation, the reference speeds are equal to zero.
- **Stop** : sets the reference speeds to zero. Being in this state leads to ending all processes and to freeing all variables before closing the program.

5.3 Speed Controller

The speed controller structure is used to control the speed of the wheels in order to give the robot its desired translation and rotation speed given by the high level controller. Figure 5.3 shows how this controller is structured. This controller is in charge of measuring the speed of the wheel (via the SPI interface), computing the error between the reference speed and the actual speed, and computing the output command of the PI controller. An anti-windup mechanism has also been implemented in the controller saturating the command at 40 % of the maximal voltage and stopping the accumulation of the integral term in case of saturation.

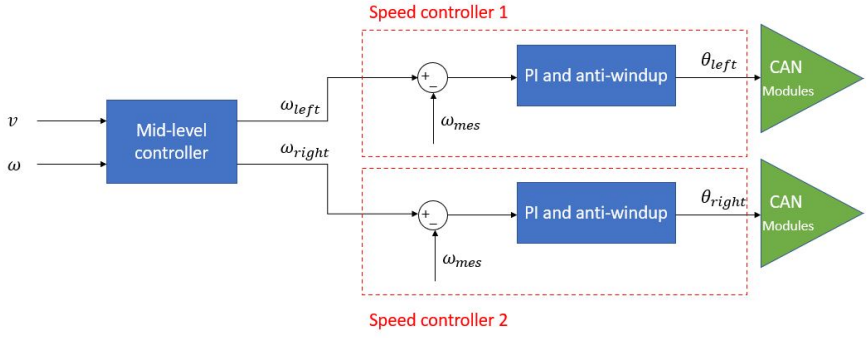


Figure 5.3: Speed controller block diagram.

5.4 Computational time required by each part of the main loop

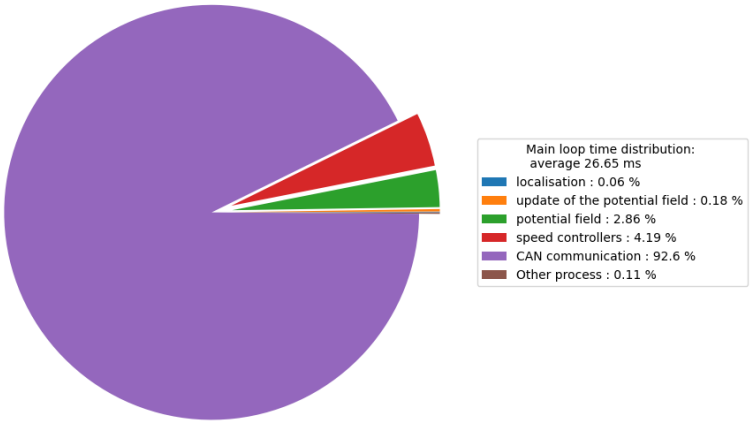


Figure 5.4: Illustration of the computational time distribution in the main loop.

A big difficulty in the project was the enhancement of the main loop speed. As illustrated in Figure 5.4, the main part of the computational time is used for the communication in CAN. A lot of work has been done to speed up this part of the code. A new version of the code was developed with the help of the group 5, but the deployment caused a lot of issues and no solutions have been found.

To generate the figure, a timer has been set around each process and the results were directly printed on the terminal, and the critical case is when the robot is in displacement to reach a goal. The time of the printing operations has been removed. The method to collect data was simple and could lead to small inaccuracies, but as only order of magnitudes were needed, those inaccuracies are acceptable.

5.5 Programming task distribution

Table 5.1 shows the task distribution in the programming of Tutankhabot.

Table 5.1: Task distribution in the programming of Tutankhabot.

	Clément	Diego	François	Nicolas	Sébastien	Théau
Multi Threads		x				
Localisation	x		x		x	
Calibration	x		x		x	
Motion Control FSM		x				
Path Planning and opponent avoidance		x		x		
FSM action	x		x		x	
Speed Controller	x		x		x	
Opponent detection			x			x
CAN communication		x	x			
Encoder and speed module	x		x		x	
SPI	x		x		x	
Sonars module (functional but not used)		x		x		
Starting cable		x	x			
Arduino communication						x
Motor actuation						x
Probes' module (functional but not used)				x		x
Screen				x		x
Code maintenance and integration		x				

CHAPTER 6

Final assessment

This chapter was written by Nicolas.

6.1 Pros and cons of Tutankhabot

In this section, the strengths and weaknesses of the robot are described.

6.1.1 Weaknesses of Tutankhabot

- **It takes its time**: Tutankhabot is a robot that takes its time: it drives slowly around the map, slowing down when near an opponent. This means that it does not have the time to do many actions, which in turn lowers its competitiveness. When it is asked to go faster, Tutankhabot starts making a fuss when trying to avoid an obstacle. This is due to the CAN protocol, that is too slow to react. An improvement to the protocol, that would have made the communication 100 times faster, has been implemented, but although it worked fine on one wheel (the other one being with the "old version of CAN"), when implemented to control both wheels, the robot starts going crazy. This problem has been investigated for many days and may be due to an impedance problem on the CAN board or another internal problem, but it could not be solved, so the "old version" of the CAN protocol, slower but working, is still the one used.
- **No real recalibration**: Tutankhabot only relies on odometry to know where it is, and nothing has been implemented to correct the errors (like a Kalman filter), as it was deemed not useful as explained in subsection 3.1.1. A more robust implementation would have been to use a triangulation process with the LiDAR to better locate itself on the map, or to implement a filter for the odometers.
- **No vectorial control**: For its path planning, Tutankhabot uses a potential field algorithm, as already explained in subsection 3.1.3. Although working fine, when working near an opponent, the robot starts having an oscillating behaviour. It may probably be due to the fact that there is no vectorial control in the robot: the control is done in forward and angular speed only, leading to some struggles when it arrives from the front on an opponent. The current control may also lead to "overshoots": when asked to reach a point behind it, it will do a large loop instead of turning on itself then moving forward. We resolved partially by limiting the forward speed when the angular speed is important.
- **Lack of versatility**: Tutankhabot is a robust yet limited robot. Indeed, only one strategy, essentially happening in our part of the map, has been encoded in the robot. It does its job, but only that and nothing else.

6.1.2 Strengths of Tutankhabot

- **A robust worker**: Although Tutankhabot is limited in its possible actions, it does them very robustly. The actions it does are simple ones, and its moderate speed allows it to reach its goal and be precise with high reliability. This robustness compensates some of its lack of versatility.
- **Compact**: Tutankhabot is a very compact robot, with packed cabling and proximity of plenty of actions. The placement of some mechanisms (e.g.: for the excavation square) allows it to chain actions quickly. Also, its smaller size helps it deal with obstacles, e.g. passing between an opponent and a wall.
- **Parallelism**: The CAN protocol may be slower than what could have been obtained, but the implementation of parallelism in the architecture helps to speed up the control loop, thus allowing a faster response to obstacles and a smoother path in general.

6.2 «If you had to do it again»

During the project, we encountered many unexpected problems, ranging from the breaking of a 3D printed mechanism to a cable breaking in its middle without warning. Those problems are a part of the game, and nothing can really be done to prevent them. However, there are several things that could be changed.

6.2.1 Concerning the mechanical aspects

The biggest change that could be done concerning the mechanical aspects of Tutankhabot is its size: indeed, we did not understand correctly the imposed limitation for the contest, thinking that the maximum perimeter is to be measured when everything is deployed. Thus, we restricted ourselves a lot so that when all the mechanisms are fully deployed, the perimeter does not exceed the 130 cm maximum limit. This way, a mechanism could have been added, e.g. a hydraulic suction cup and a storage room for samples. Moreover, even stronger mechanisms and a better cable management throughout the robot would be achieved.

6.2.2 Concerning the electrical aspects

We encountered a lot of problems with the home-made PCBs, ranging from a component burning and short-circuiting an entire PCB because of the oscillations in the voltage when we first supplied the board[1] to a lack of components. In general, the usage of SMD components was not a good idea for burnable components like the chips to convert the logic level of the data. An idea is to put the important chips on sockets so that it is easy to change them. Another mistake was making an order of components in insufficient number, and we lost valuable time waiting for new components.

6.2.3 Concerning the programming aspects

Several things could be done to improve the code:

- **A faster loop**: with more time, a solution might have been found for the faster implementation of the CAN protocol and with it, we could have been faster on the map and smoother in the opponent avoidance.
- **Re-calibration**: to limit positioning errors, many things can be implemented, e.g. triangulation or a Kalman filter as already mentioned previously in subsection 6.1.1.
- **Better control**: as already discussed, Tutankhabot starts having an oscillating behaviour when confronted to an opponent. By implementing the vectorial control, this could be improved: e.g. when the goal is behind the robot, instead of commanding the robot to "move forward and turn", with a vectorial control, it would first turn then move forward. Also, damping could be added to the control, making its movements smoother.
- **Better versatility**: with more implemented actions, something that could be done is change its strategy dynamically and not waste time for nothing, e.g. for the probes if the robot detects that there is no excavation square to measure because they are already turned down, it could decide immediately to do something else. This decision could also be based on the opponent's position, to choose an action where the robot would not enter in direct conflict with the opponent which would disturb its functioning.
- **Improve the path planning**: although a robust algorithm, it has several weaknesses, like local minima. If it was to be redone, we would go for another path planning strategy, like an A* or D* algorithm.

6.3 Evaluation of the way the group was operating and managed

The project started by a lot of brainstorming as a team. But when the different subparts of the project really started, around the time the minibots arrived, the group split into three groups: Théau would do the programming of the minibot, Clément, François and Sébastien would work on the mechanical aspects of Tutankhabot and Diego and Nicolas would work on the electronics. These groups stayed together until the end, working on their respective subparts. Concerning the programming aspect, it got split between the members as explained in chapter 5.

The main criticism that we could give for the group management is the lack of communication between the subgroups. Although there was a high level of trust between the members of the group concerning the quality of the work each person would do, the lack of communication lead to some surprises when building up the robot, and changes had to be done to the different floors of the robot, which made us lose time in the big picture.

[1] The capacitance concerned could afford 6 V at most. When we supplied it with 5 V, by flicking the switch, oscillations bigger than 6 V appeared, effectively destroying the capacitance and the PCB.

This lack of communication has several explanations: the group globally worked as three entities: Théau, often working with Sébastien, François and Clément, and then Diego and Nicolas. The members of each subgroup knew each other well long before the project, and it was more comfortable for them to work together. As these entities had different schedules, they usually were not there at the lab at the same time. Initially, regular meetings allowed for everyone to be up-to-date on what was to be done and who would do it, but soon these meetings were abandoned when we started having too much to do.

Despite this, the final product fulfils most of what was expected and this project, overall, was very enjoyable.

CHAPTER 7

Conclusion

The conclusion has been written by Nicolas

This report went over the detailed description of Tutankhabot, from its abilities to how it moves, locates itself, and acts on the environment. The mechanical, electrical and programming aspects of the project have been either summarised or detailed, depending on what had already been explained in previous reports. It concluded by a recap of the pros and cons of Tutankhabot, an evaluation of the team's functioning and an answer to the question "what if we had to do it again".

This is the end of a year-long project for us. We had our ups and downs, some moments where nothing was working, and we wanted to quit it, other moments where it would start working and re-ignite the flame of passion for the project. This is the first time any of us had to work on such a big scale project, spanning over such a long period of time. It has been challenging, we sometimes struggled to work as a unit, but at the end, Tutankhabot meets most of our expectations, and we are happy with what we have achieved. Never before have we been challenged this much, learning on the spot almost everything, and struggling with the -sometimes stupid- problems we would encounter. We come out of it grown, enriched in experience and skills.

This project has also been a fantastic display of mutual aid, laughter and overall friendly time with the other groups. It was really motivating to work in such a friendly and helpful environment. We would like to thank every group for the help, advice and friendliness shared throughout the year, and especially group 5, Carbot14, with which we shared the most. Finally, the team would like to warmly thank the teaching staff, the assistants especially Louis Devillez, whom we (gently) harassed with our many questions- and the technical staff linked to the project, especially Thierry Daras for his help with the components and Souley Djadjandi for his help with the PCBs.

Until next time !

The Tutankhabot team

A video showing steps of the development of Tutankhabot and its capabilities can be found here: `https://youtu.be/ghdG_GZrbP0`

CHAPTER 8

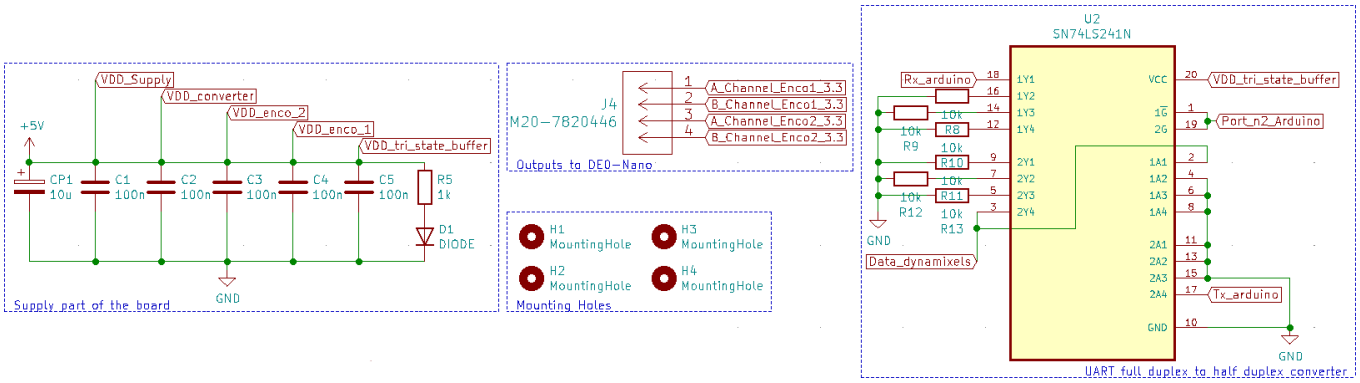
Appendices

Nicolas worked on the general form of the report, later reworked with François, Clément and Sébastien. The cohesion of the report was checked by Nicolas, François, Clément and Sébastien. For these Appendices, Nicolas wrote the Electronics chapter.

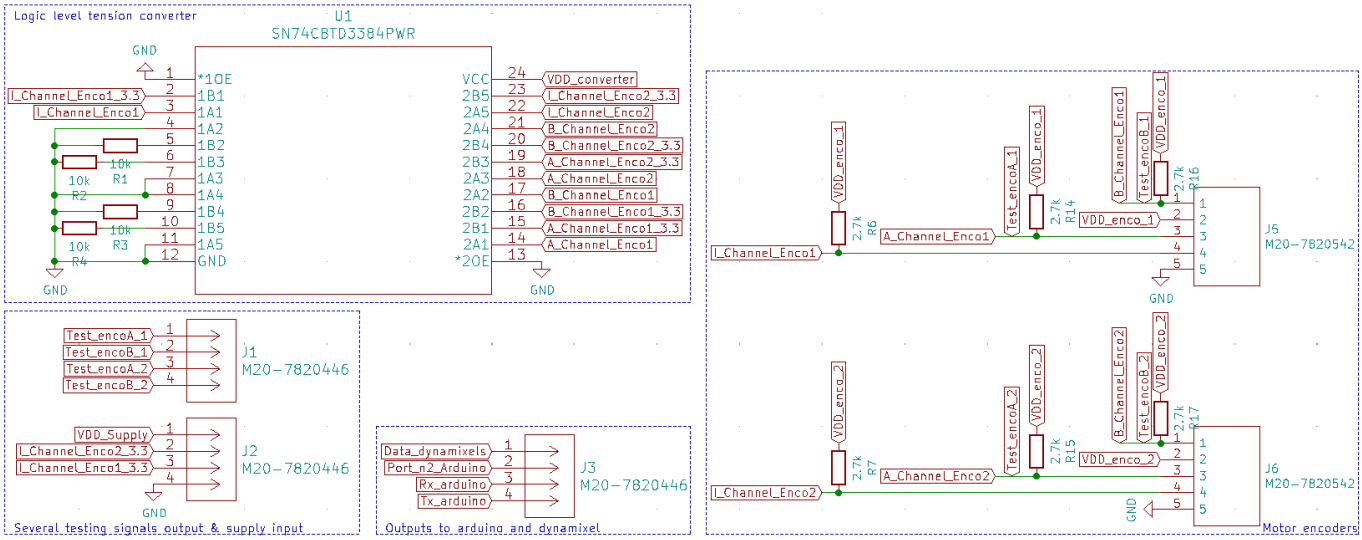
8.1 Chapter 4

8.1.1 Interface 1: *Interface motor encoders*

Zoom on the schematic



(a) Zoom on the upper part of the schematic for interface 1.



(b) Zoom on the bottom part of the schematic for interface 1.

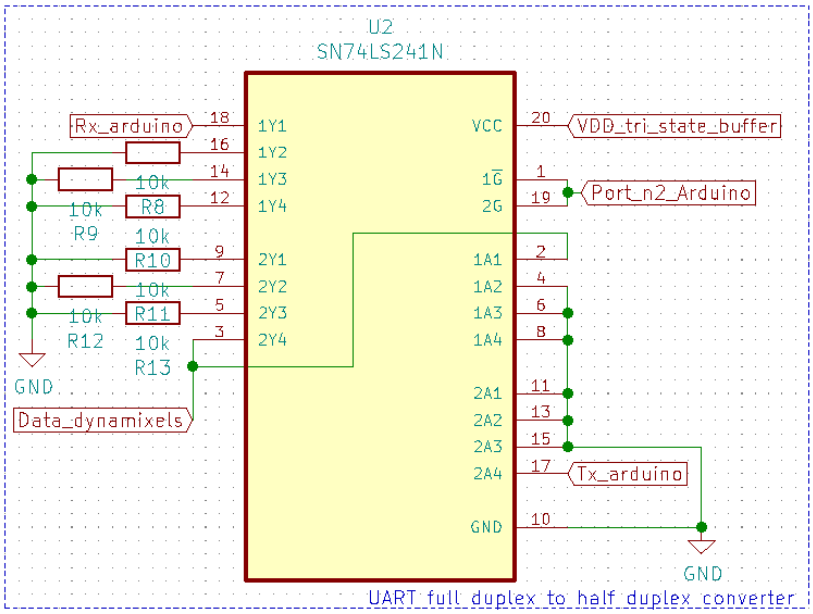
Figure 8.1

The encoders

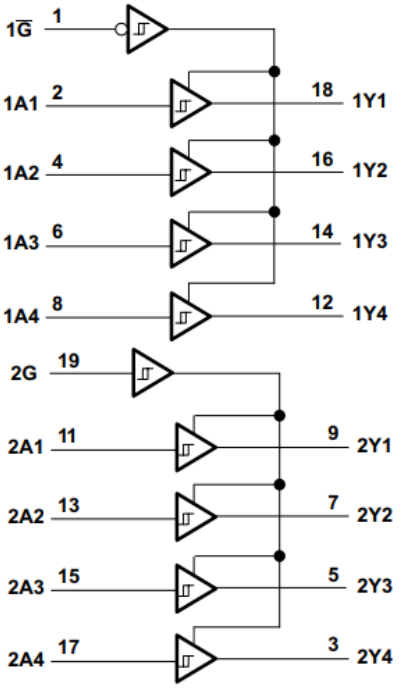
Each signal channel (A, B and X) has a pull-up resistor of 2.7 kΩ, as required by the datasheet, because the lines are normally high.

The main chips

- **SN74LS241N**: this chip [9] allows a correct communication between a full duplex and half-duplex UART. It works with two tri-state buffers, one for each direction, that have opposite enable signals $1\bar{G}$ and 2G. In the chip, they are assigned to *Port_n2_Arduino*. The *Rx_arduino* signal on 1Y1 (output of the chip) is connected through a tri-state buffer to 1A1 which receives the feedback data from the Dynamixels : this allows communication from the Dynamixel to the Arduino. The *Tx_arduino* signal on 2A4 (input of the chip) which receives the command from the Arduino for the actuator is connected through a tri-state buffer to 2Y4 which transfers the command to the Dynamixel : this allows communication from the Arduino to the actuator. As the two tri-state buffers implied in this action are controlled by an opposite signal, only one way is active at a time.



(a) Zoom on the part of the PCB dealing with UART communication.

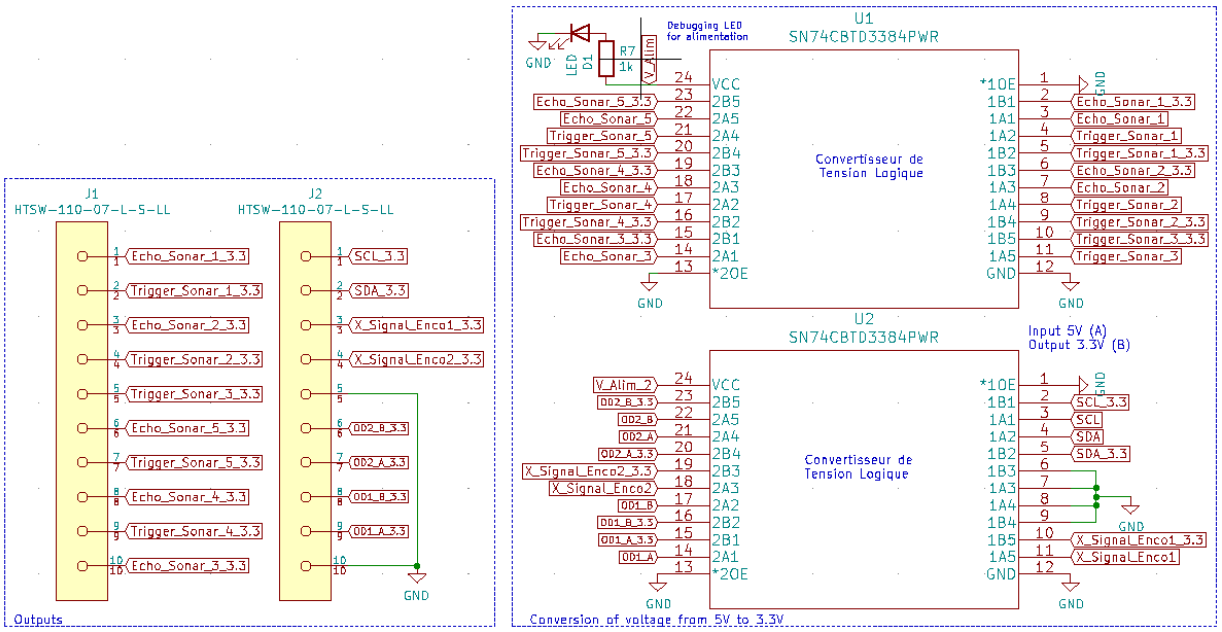


(b) Schematic representation of the chip's interior [9].

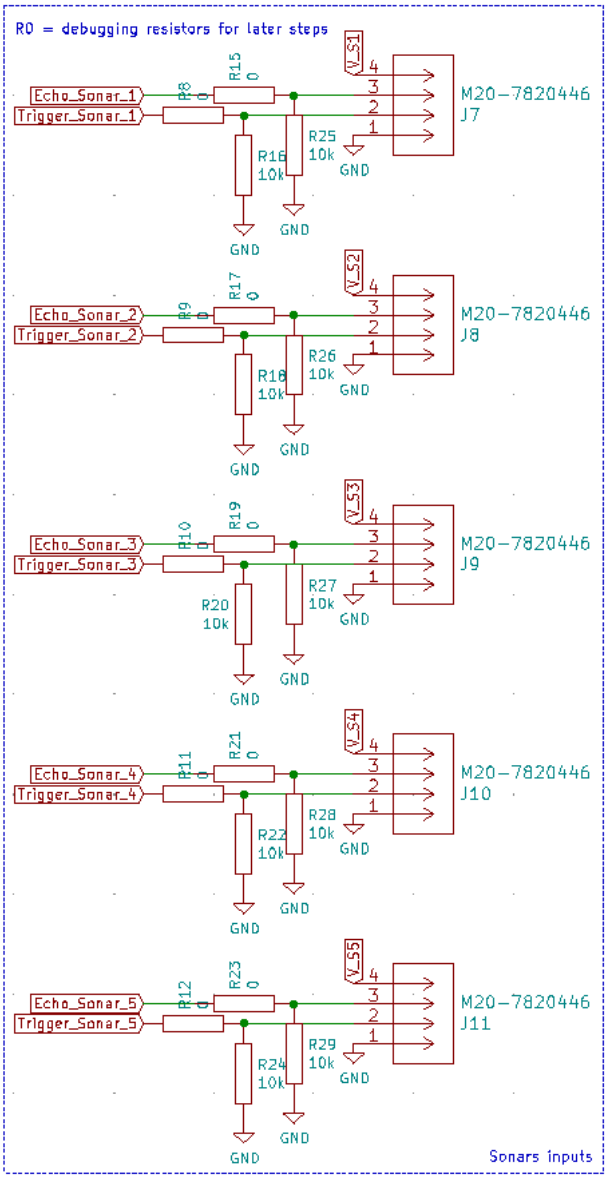
Figure 8.2

8.1.2 Interface 2: *Interface sonars & odometers*

Zoom on the schematic

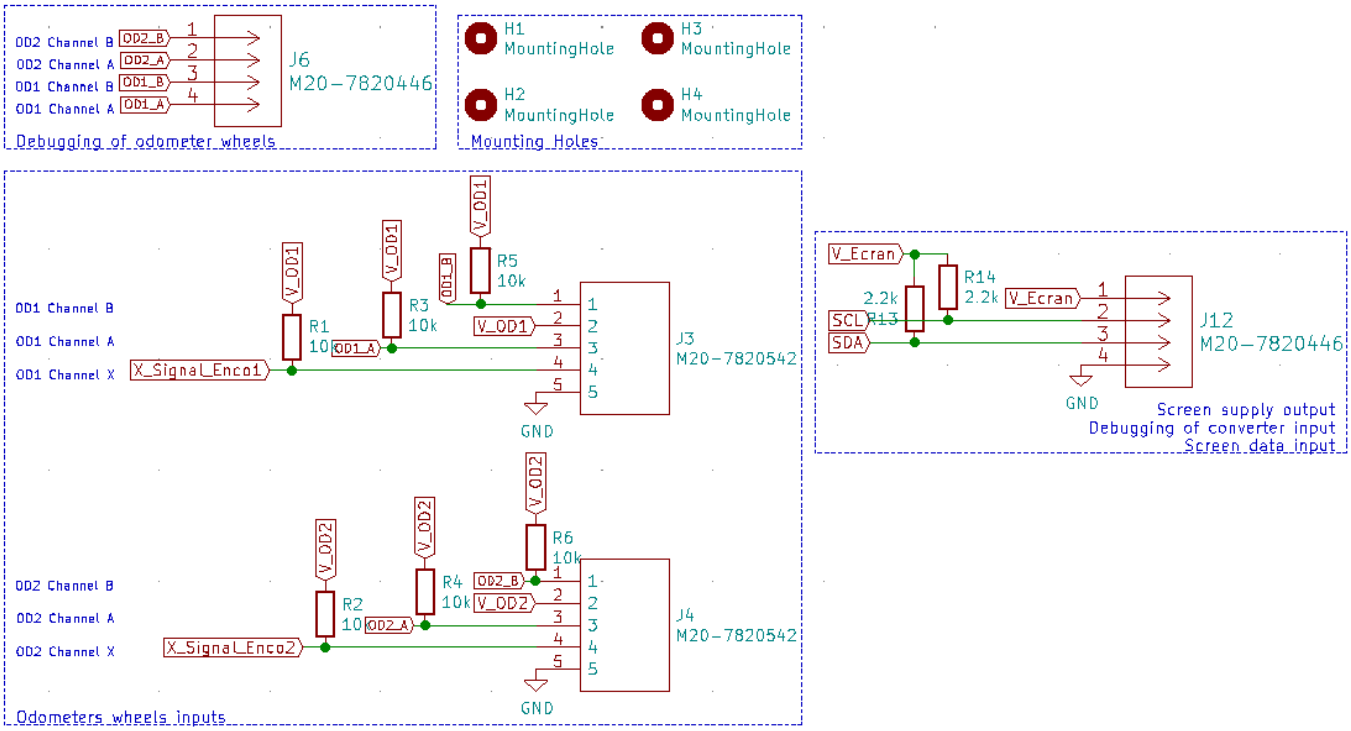


(a) Zoom on the upper left part of the schematic for interface 2.

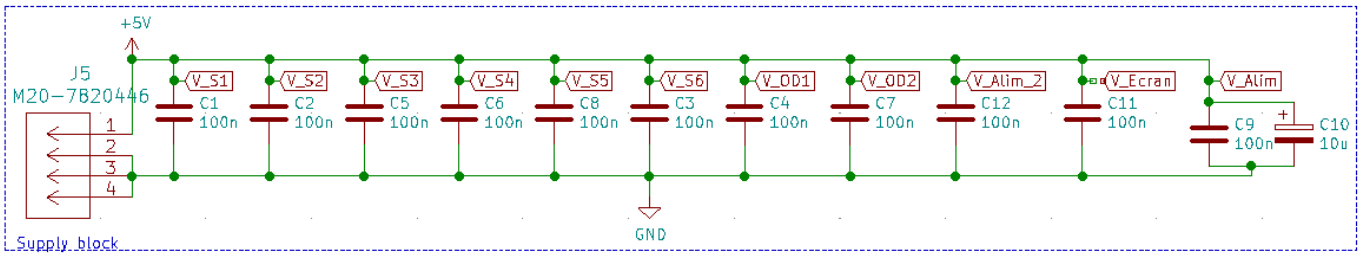


(b) Zoom on the sonars input part of the schematic for interface 2.

Figure 8.3



(a) Zoom on the lower left part of the schematic for interface 2.



(b) Zoom on the supply part of the schematic for interface 2.

Figure 8.4

8.1.3 Interface 3: *Interface probes*

The schematic of the interface is given in Figure 8.5.

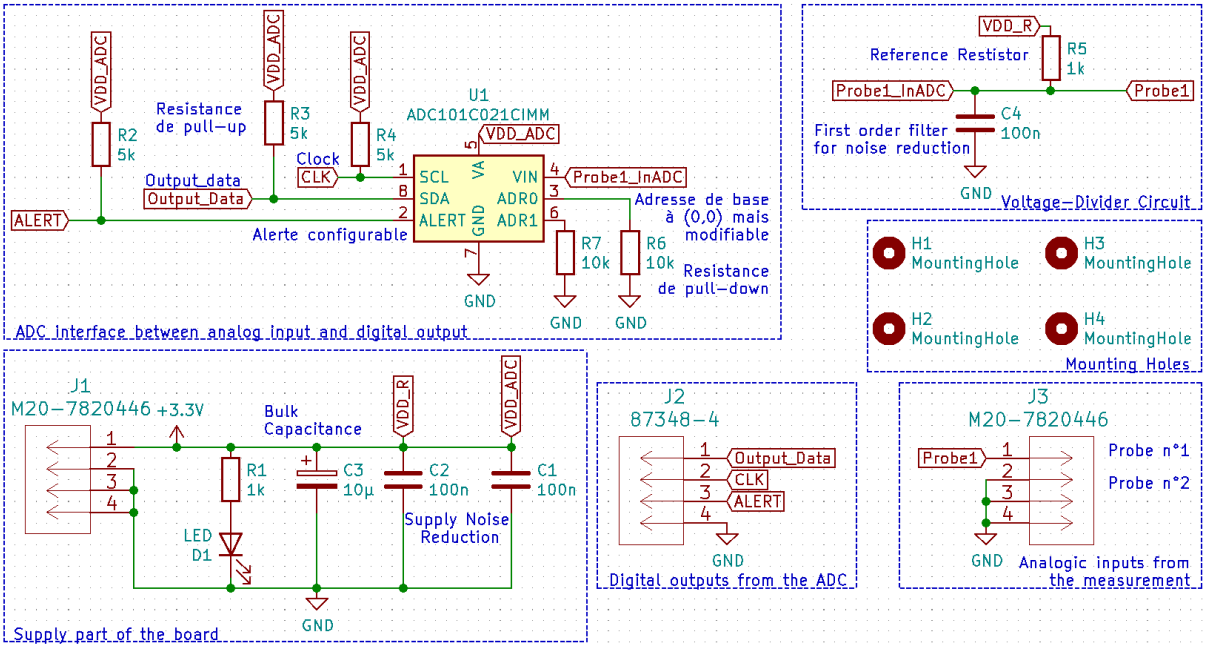


Figure 8.5: KiCad schematic of the PCB board *Interface probes*.

The board is supplied by a 3.3 V voltage used as supply and as reference for the computations of the resistance. The working principle is simple: due to a voltage divider (visible in the upper right corner of Figure 8.5), an analog input is fed into an ADC which converts it into a digital data sent to the FPGA via an I2C communication.

The main chips

The only chip used in this PCB is an ADC, the **ADC101C021CIMM** [10], which has a resolution of 10 bits, which is way enough for our usage as there are only three resistors to differentiate and their values are very different. It has a high SNR of 61.8 dB and has a typical power consumption of 0.26 mW.

Bibliography

[1] T. Bernard, S. Bosschaert, F. Boulanger, N. Isenguerre, D Lannoye, and C. Mercier. *Online Archives - final project*. May 2022. URL: https://uclouvain-my.sharepoint.com/:f:/g/personal/nicolas_isenguerre_student_uclouvain_be/EizlNRF6hchKoZU9qQnohc4BohAasB7WRDDx7atFBeNRmw?e=QIr5Ia.

[2] Eurobot organisation committee. *EurobotOpen 2022 Rules*. [Online]. 2022. URL: https://www.coupederobotique.fr/wp-content/uploads/Eurobot2022_Rules-EN.pdf (visited on 05/12/2022).

[3] T. Bernard, S. Bosschaert, F. Boulanger, N. Isenguerre, D Lannoye, and C. Mercier. *Technical report - LELME2002*. Last consulted on 12 May. December 2021.

[4] Elecfreaks. *Ultrasonic Ranging Module HC - SR04*. [Online]. 1999. URL: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf (visited on 05/13/2022).

[5] T. Bernard, S. Bosschaert, F. Boulanger, N. Isenguerre, D Lannoye, and C. Mercier. *LELME2732 : Robot modelling and control - Final report*. Last consulted on 13 May. April 2022.

[6] Arduino Docs. *Use Multiple Serial Ports on the Arduino Mega*. [Online]. May 2020. URL: https://create.arduino.cc/projecthub/maurizfa-13216008-arthur-jogy-13216037-agha-maretha-13216095/modbus-rs-485-using-arduino-c055b5 (visited on 05/13/2022).

[7] Arduino. *Mega 2560 Rev3*. [Online]. Modified: 13/05/2022. URL: https://docs.arduino.cc/hardware/mega-2560 (visited on 05/13/2022).

[8] Texas Instrument. *SN74CBTD3384*. [Online]. May 1995 - Revised on the 15th Jan 2004. URL: https://www.ti.com/product/SN74CBTD3384?HQS=ti-null-null-verifimanuf_df-manu-pf-octopart-wwe (visited on 05/13/2022).

[9] Texas Instrument. *SN74LS241*. [Online]. April 1985 - Revised on the 10th Oct 2016. URL: https://www.ti.com/product/SN74LS241 (visited on 05/13/2022).

[10] Texas Instrument. *ADC101C021*. [Online]. February 2008 - Revised on the 19th Feb 2013. URL: https://www.ti.com/product/ADC101C021 (visited on 05/13/2022).