École Polytechnique de Louvain

LELME2732 : Robot modelling and control
Final report

Group 4 :
Bernard Théau 69111800
Bosschaert Sébastien 11671800
Boulanger François 12301800
Isenguerre Nicolas 50041800
Lannoye Diego 29591800
Mercier Clément 55031800

September 22, 2022

Contents

1 Introduction

The goal of this report is to describe our implementation of the control of a mobile robot that will be used in a simulation of the Eurobot contest. During this contest, it will have to be able to localize itself as well as to move in a well known environment in order to catch some targets and to put them back in its base camp. All this will have to be done without any collision with the opponent that will be performing the same tasks.

We will start from the low level elements to the more high level one covering the independent control of the wheels and the conversion of the speed of the robot into the speed of the wheels, the system of localization, then the detection of the environment and finally the planning of the path that our robot will follow combining all the previously mentioned elements.

1.1 Preliminary remark

For the origin of the reference frame and the x and y-axis, we have made a different choice than the one given in the README. Indeed, for the sake of ease with the project in mechatronics, we decided to take the origin of our reference frame identical to the one used by the rules of the Eurobot contest. Then we chose the x and y-axis so that all distances on the map are positive. The configuration that has been taken is in Figure 1.

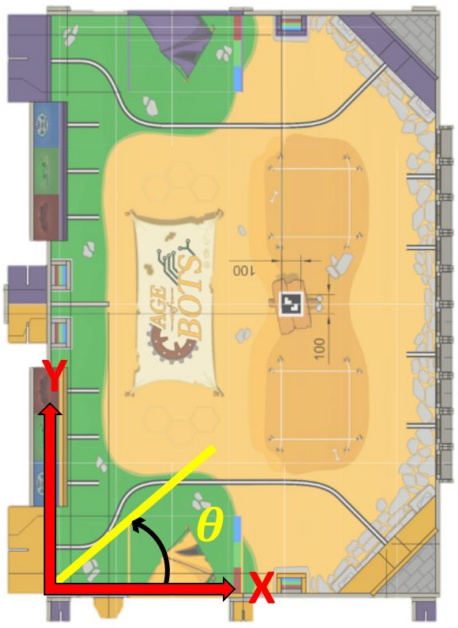


Figure 1: Reference frame

2 Low-level control and mid-level control

2.1 Low-level control

To reach the desired velocity, we have implemented a controller in a closed loop that controls the motors of the wheels. The block diagram of this controller is represented in Figure 2. We then simulated this controller, considering that the robot had to reach a translational speed of $v = 0.3\ m/s$ at time $t = 0\ s$. We can see in Figure 3 that the settling time is about 0.1 seconds and that there is no overshoot.

2.2 Mid-level control

We have implemented one of the actions of the mid-level controller: the conversion of the global speeds of the robot to the speeds of the different wheels. This conversion (seen during lecture 2) gives the following equations

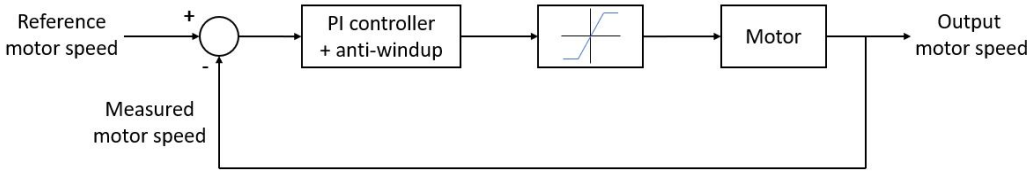


Figure 2: Controller block diagram.

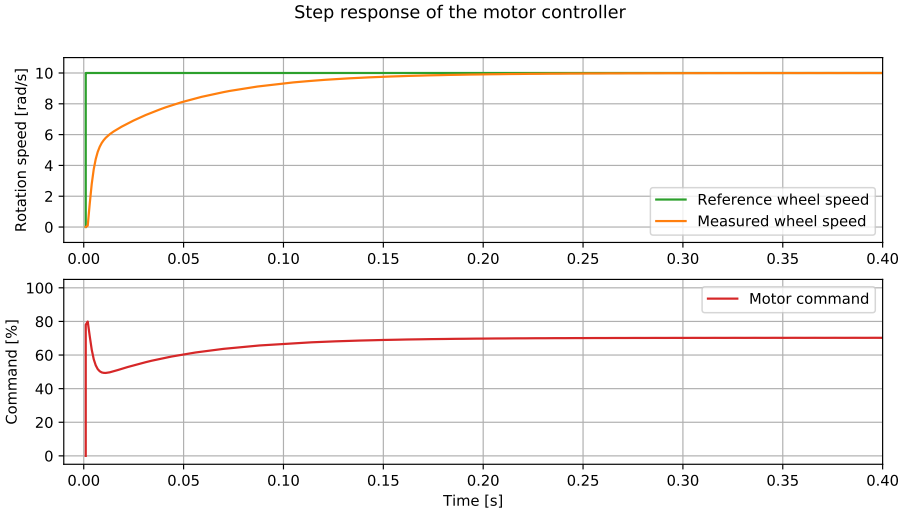


Figure 3: Illustration of the rotation speed (top) obtained by a given command (bottom).

that we have implemented in our code:

$$\dot{\phi_r} = 1/r \cdot (v + l \cdot \omega) \qquad \dot{\phi_l} = 1/r \cdot (v - l \cdot \omega)$$

Where v is the translational speed of the whole robot, ω is the rotational speed, r is the radius of the wheels and l is half the distance between the two wheels (distance to the rotation point of the robot).

3 Position tracking: odometry

For position tracking, we have implemented odometry. With this method, we integrate the motions of the wheels in order to find the location of our robot. This localization system will be the only one used to localize the robot. We made this choice because this is what we will use in our final robot and thanks to this simulation, we will be able to check that the odometry system is enough for the position tracking. If we conclude that this is not the case, we will have to implement other methods such as triangulation using the beacon and the lidar to allow the robot to locate itself properly.

To convert the speed of the odometers, we used the following equations:

$$p' = p + \Delta p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta s \cdot cos(\theta + \Delta\theta/2) \\ \Delta s \cdot sin(\theta + \Delta\theta/2) \\ \Delta\theta \end{bmatrix} \tag{1}$$

With:

$$\Delta\theta = \frac{\Delta s_r - \Delta s_l}{b} \qquad \Delta s = \frac{\Delta s_r + \Delta s_l}{2} \tag{2}$$

To show that the odometry is working properly, we have programmed the robot to follow a certain speed profile and we have plotted the results obtained with the odometry in Figure 4.

3.1 Calibration procedure

To calibrate the system, we have implemented a calibration procedure, composed of several steps, which is followed by the robot just before the start of the match. In that way, it allows us to give the exact initial position of the robot to the algorithm of the odometry.

First, the robot goes backward to be against the border of the map, this way we can set the y-axis origin. Then the robot goes forward, makes a 90° turn and goes backward again to get in contact with another side of the map border so that it is able to set the x-axis origin. Finally, it returns to its initial position before the 10-second timer expires so that it is not penalized. The robot also adjusts its orientation every time it meets a border during this calibration procedure, resulting in a calibration of the three coordinates describing the location of our robot :

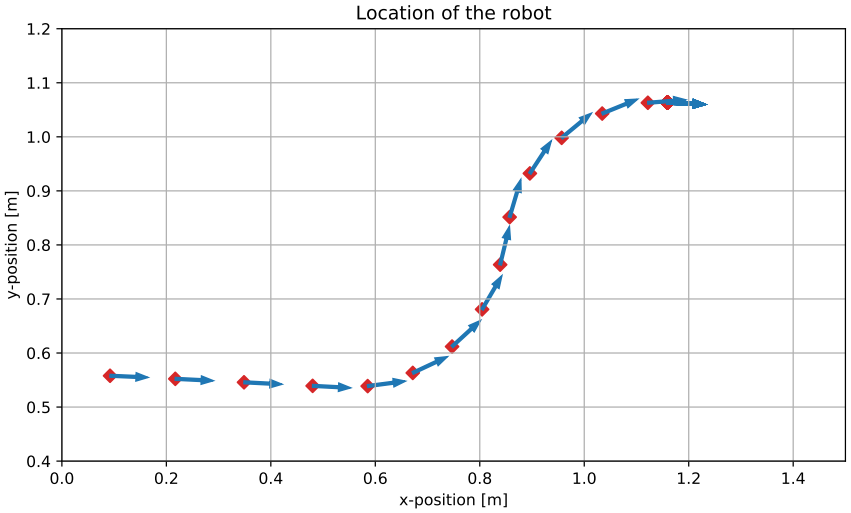


Figure 4: Illustration of the working of localization with odometry.

The calibration procedure is slightly different depending on whether we are assigned to the purple team or to the yellow team. The switching between the cases is handled in the file *Localization.cc*. We also made the choice that we would only use one robot in our team because it will also be the case for the real contest. This is why the calibration procedure does not work for a case in which two robots are in our base camp.

4 Opponents detection: LiDAR

To be able to detect the opponent during the game, we will use the lidar as well as the beacon placed on top of the opponent's robot. This system gives us two arrays of data containing, for every detected point, the angle at which the point was detected and the distance between the point and the lidar. Consequently, the information given by the lidar is in the relative frame (i.e. the frame attached to the robot and moving with it). That is why we had to perform a transformation of coordinates as explained in this section in order to be able to identify the beacons and the opponents.

4.1 Compensation of the movements of the robot

As said before, in order to obtain the location of the points detected by the lidar, we have to make a frame transformation to change the location of the different points which are expressed in the relative frame into the absolute frame. This is necessary for the path planning and the opponent avoidance. At first, we can consider a static solution and after the application of trigonometric equations, we can obtain the formulas shown in Equation 3 that give the absolute location of the points according to the data given by the lidar.

$$\begin{cases} x_i = r_{lidar,i} * \cos \alpha_{lidar,i} + \theta_{bot} + x_{bot} \\ y_i = r_{lidar,i} * \sin \alpha_{lidar,i} + \theta_{bot} + y_{bot} \end{cases} \tag{3}$$

where x_i and y_i are the absolute coordinates of the point i detected by the lidar, $r_{lidar,i}$ and $\alpha_{lidar,i}$ are respectively the distance and the angle at which the point i was detected by the lidar and x_{bot}, y_{bot} and θ_{bot} are the current position and orientation of the robot.

This static assumption gives good results when the robot is at rest but when it starts moving and especially rotating, the data computed with the previous equations are completely distorted due to the fact that location and orientation considered during the transformation are not the same as when the data was taken by the lidar. Indeed, some delays will occur between the instant the sample is taken and the transformation is performed. There are two kinds of delays that are introduced here. The first delay is the same for every points given by the lidar and is due to the fact that the transformation is performed a certain time after the end of the take of the data. This delay will be denoted as Δt_{lidar}. The second delay is due to the fact that the lidar takes some time to perform one cycle of measurement and is consequently different for each of the samples of one array. To compute this delay, we will have to consider the total amount of data taken during one sampling cycle nb_{data} as well as the frequency of the lidar f_{lidar} which indicates the time taken to perform one cycle of sampling. Considering these two delays, we will add a correction of angle and location considering the distance and angle travelled by the robot during this duration. All this is done under the assumption that the translational and rotational speeds v_{bot} and ω_{bot} of the robot are constant. The new equations are given in Equation 4

$$\begin{cases} x_i = r_{lidar,i} * \cos(\alpha_{lidar,i} + \theta_{bot} - \omega_{bot}(\Delta t_{lidar} + \frac{prop}{f_{lidar}})) + x_{bot} - dv_x(\Delta t_{lidar} + \frac{prop}{f_{lidar}}) \\ y_i = r_{lidar,i} * \sin(\alpha_{lidar,i} + \theta_{bot} - \omega_{bot}(\Delta t_{lidar} + \frac{prop}{f_{lidar}})) + y_{bot} - dv_y(\Delta t_{lidar} + \frac{prop}{f_{lidar}}) \end{cases} \tag{4}$$

Where $dv_x = v_{bot} \cos(\theta_{bot} - \omega_{bot}(\Delta t_{lidar} + \frac{prop}{f_{lidar}})$, $dv_y = v_{bot} \sin(\theta_{bot} - \omega_{bot}(\Delta t_{lidar} + \frac{prop}{f_{lidar}})$ and $prop = \frac{nb_{data} - i}{nb_{data}}$.

The Figure 5 shows the resulting point obtained with such a transformation with a translational speed varying from 0 to 0.3 m/s and a rotational speed varying from -0.5 and 3 rad/s. We can see in this figure that the detection and the computation of the absolute location of the points are always the same no matter the speed at which the robot is moving. This indicates that the transformation is working properly. However, this transformation still has some limitations. Indeed, we did not take the fact that the robot is not moving with a constant speed into account, but is actually accelerating. We also considered that the robot is first performing a translation motion followed by a rotation, which is not exactly the case in real life since these two motions are conducted continuously. The effect of these two approximations is significant when the robot is going through high acceleration phases, leading to a distortion of the computed data. But these effects are at least not observable if the robot remains at a low level of acceleration, which is sufficient for our application.

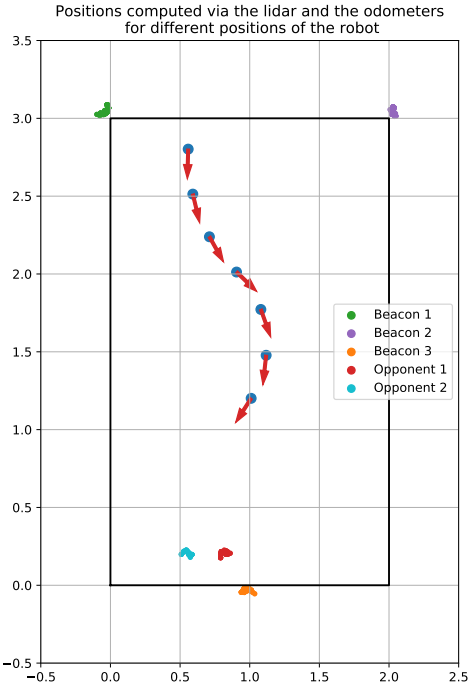


Figure 5: Results obtained with the lidar for different speeds and location.

4.2 Identification of opponents and beacons

As we can notice in Figure 5, we are also able to detect whether a point corresponds to a beacon or to one of the opponent. We are also able to differentiate the beacon and the two opponents. We will explain in this section how it has been performed.

Once the lidar has made its sample, the first thing that is done, even before the conversion into absolute coordinates presented earlier, is to filter out the samples corresponding to the fact that the lidar did not see anything in this direction (i.e. with a radius way bigger than the size of the map). This reduces a lot the amount of data on which we will perform calculation. Afterward, if a point has indeed been detected, we can perform the transformation presented previously.

Once we know that the point has well been detected and that we have its absolute location, we can start a small analyze based on its computed location to see whether it is located at the place at which there should be a beacon. If it is well the case, the point is inserted in the set of the point forming this beacon. If it is not at the location of a beacon, it must be an opponent. But before adding it to the set of the opponent, we must consider that there could be two opponents on the map. This is why we perform another small analysis. If no points have been added in the set of the point forming the first opponent, we simply add it in this set. On the other hand, if some points are already present inside this set, we first check whether this new point is far from the last point that has been added or not. If it is the case, the point is added into this set of points. If not, it must be one of the point forming the second opponent and is added to the corresponding set of points.

Once we have the set of all the points that are forming our two opponents, in order to obtain the position of the opponents, we simply perform an average providing one unique location for each of the opponents. We can notice that the beacon are not used for the localization of the robot as we said earlier in this report because we only use odometry, however we could use them with triangulation since we are able to identify them. This implementation could be part of the possible improvements of the robot which, in addition to the odometry and using a Kalman filter, would give us a better approximation of the location of our robot. This has not been made here but could be done for our real robot if the odometry is not sufficient.

5 Path planning and obstacle avoidance

As stated in the intermediate report, the path planning and obstacle avoidance are accomplished by the *potential field* method. At each instant, the robot calculates the forces that are applied to it. It is repulsed by repulsive forces from obstacles and attracted by an attractive force from a given goal. The robot will gradually be attracted by the goal. To see a video illustrating the final result of the path planning algorithm click here.

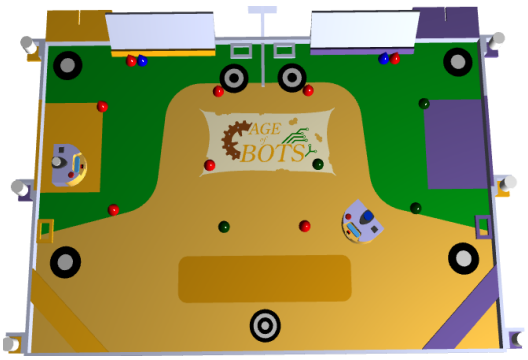
5.1 Simple case : no opponent between the robot and the goal

The working principle of the method is visible in Figure 6. In theory, the total attractive force should reach 0 on the goal. In our application, the goal is defined by a radius and the position of its center and the position of the robot is defined at the middle of the line between the two motorized wheels. The target is detected before the two positions are the same and the robot stops to pick up the goal. As the attractive force is given by :

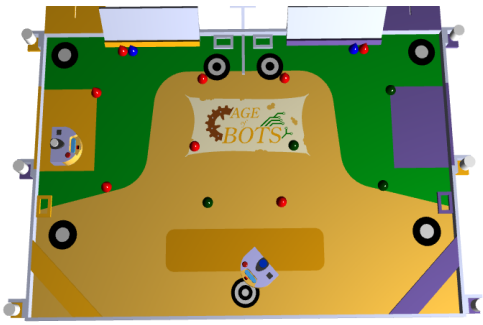
$$F_{att} = -k_{att}(q - q_{goal}) \tag{5}$$

with q the position of the robot and q_{goal} the goal's defined position, this gives a non zero attractive force. The working of the FSM used by the robot to navigate autonomously will be explained briefly in section 6.

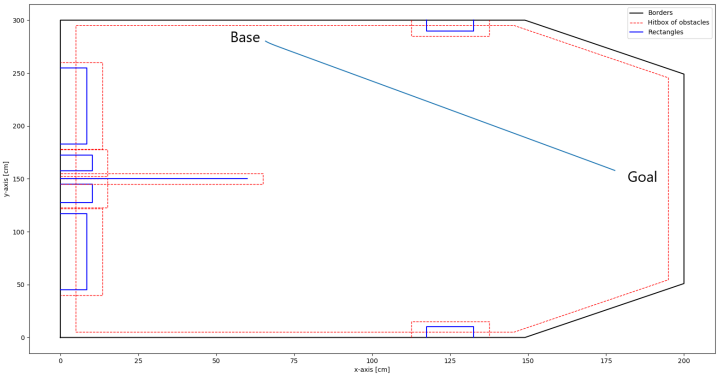
(a) The robot starts at the base. It is attracted by the sample and will start moving towards it.



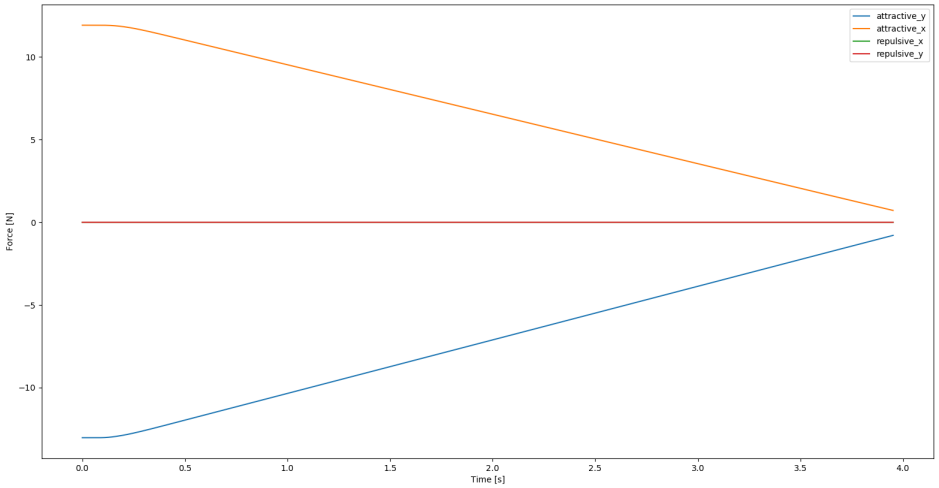
(b) Progressively, it reaches the goal.



(c) Eventually, it gets to the goal.



(d) Plot of the trajectory followed by the robot to reach the goal.

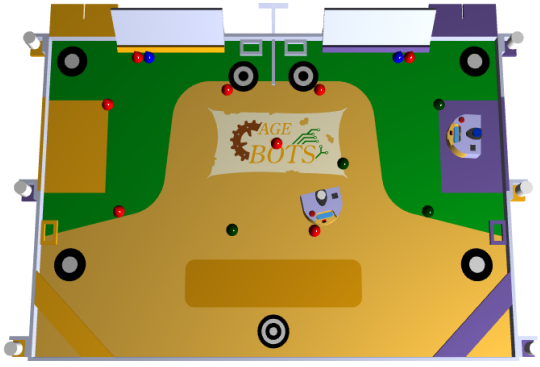


(e) Graph illustrating the evolution of the forces. The attractive forces converge towards 0 as the robot gets closer to the goal, as expected for a *potential field* method. The repulsive forces are barely present: this is linked to the weight that was given to the obstacles and the fact that there are no real obstacles in the path of the robot.

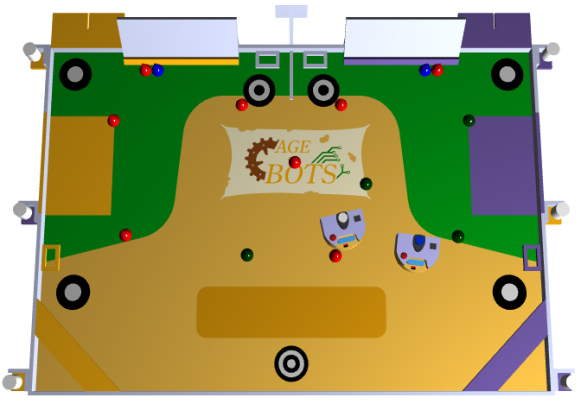
Figure 6: Illustration of the working of the potential field method : simple case with no obstacles between the robot and the goal.

5.2 Case when the opponent is in the way

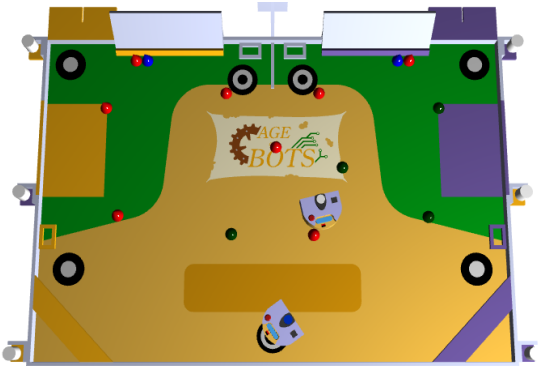
The case when an opponent is getting in the path between the robot and the goal is illustrated in Figure 7. This illustrates the way the robot will avoid obstacles : as soon as it enters the influence zone of an obstacle (namely, the opponent), it is subjected to repulsive forces which repulse it away from the opponent.



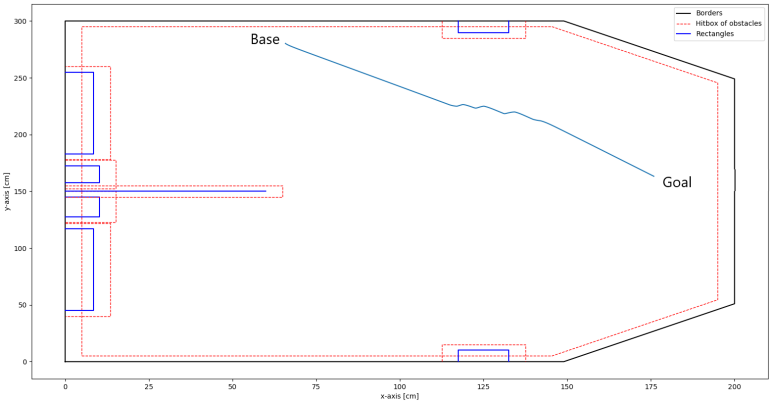
(a) At start : the allied robot is in base and the opponent stands in the way between the robot and its goal.



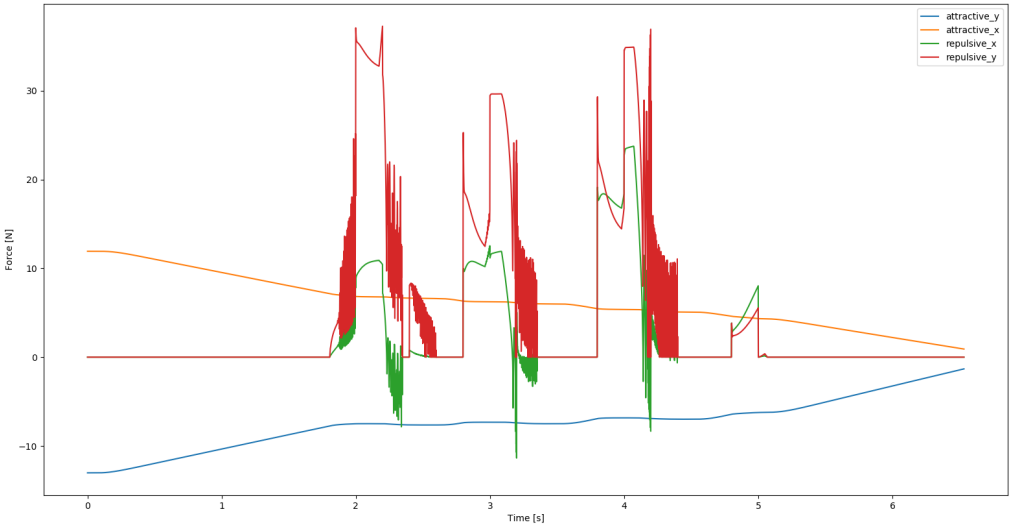
(b) The robot is repulsed by the opponent, as it is detected as an obstacle.



(c) It still manages to reach the goal.



(d) Plot of the trajectory followed by the robot to reach the goal. The effect of opponent's presence is visible : the robot stops moving in a straight line to go around the opponent.



(e) Graph illustrating the evolution of the forces. There are peaks of repulsive forces when the robot enters the opponent's zone of influence, repelling the robot away from it. The attractive forces remains globally similar compared to the easy case.

Figure 7: Illustration of the working of the potential field method : case with an opponent in the way.

5.3 Speed limitations

There is a tradeoff between the angular speed and the forward speed of the robot. There are two extreme cases :

- When both wheels turn in the same direction at maximum speed : we reach the maximum forward speed, which is 0.445 m/s.

- When both wheels turn in opposite directions at maximum speed : we reach the maximum angular speed, which is 4.94 rad/s.

We made the choice to either go at full forward speed or full angular speed. Considering the tradeoff explained before, when one speed peaks, the other is at 0. Figure 8 illustrates this principle.

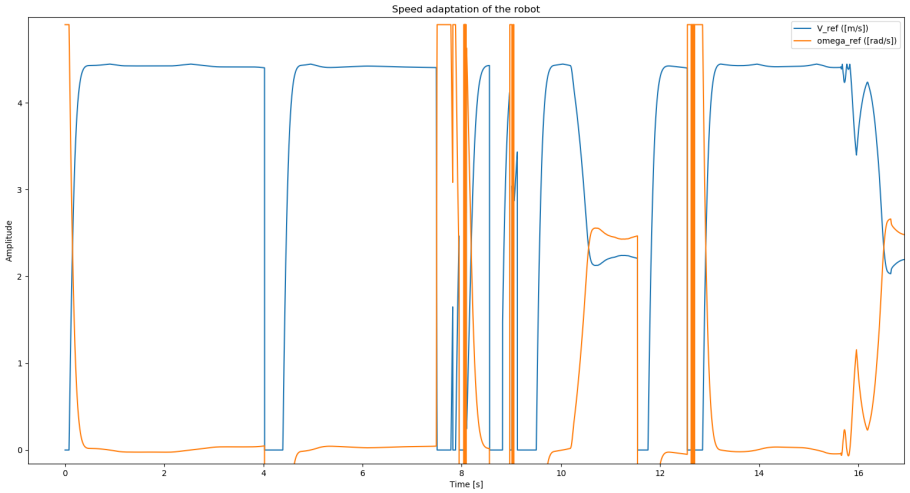


Figure 8: Illustration of speed adaptation : when one type of speed peaks at its maximum, the other one is at 0. For illustration purposes, the forward speed (in blue) has been multiplied by 10.

5.4 Intermediary Goals feature

A struggle of the potential field method is dealing with local minima. In this project, the disposition of some obstacles leads to local minima. To prevent the robot falling into them and to ease the movements of the robot, we created a function that adds new goals between the actual goal and the robot's position. As the obstacles are static and the map is known, these new goals can be hard-coded. To do so, we divided the map into zones (see Figure 9) and defined intermediate goals to move between one zone to another. The function will choose which intermediate goal to set, depending on the current robot position and the next goal position. To choose the coordinates of an intermediate goal, the methodology was to observe the initial trajectory of the robot and then choose a point in the trajectory that maximizes its linearity.

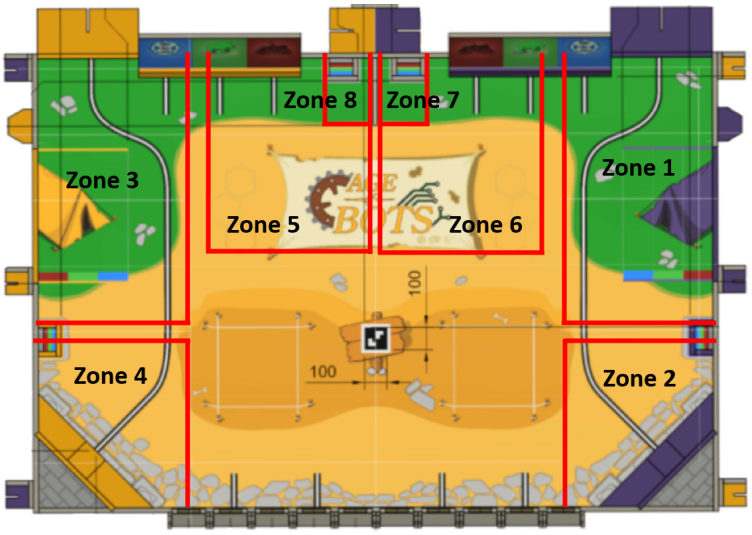


Figure 9: Division of the map in zones

5.5 Destuck feature

As there are inaccuracies on the localization and as there are some obstacles that cannot be detected, it is impossible to remove all probability of being stuck. From experimentation, we know that the robot will be stuck mostly when it is blocked near a goal by an obstacle, either because it is undetectable or because the real position of our robot does not correspond to the actual one.

Two solutions have been tested : swapping the current goal or going to the center of the map. None of them lead to great results in general. The solution to incertitude in positioning should be in regular calibrations of the odometer wheels.

6 Strategy

6.1 Modules control

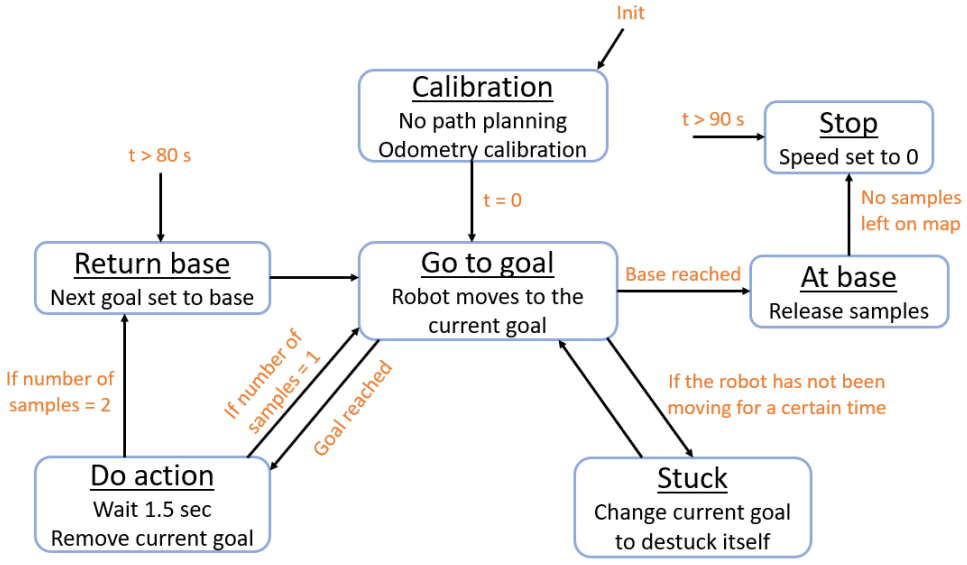


Figure 10: FSM design

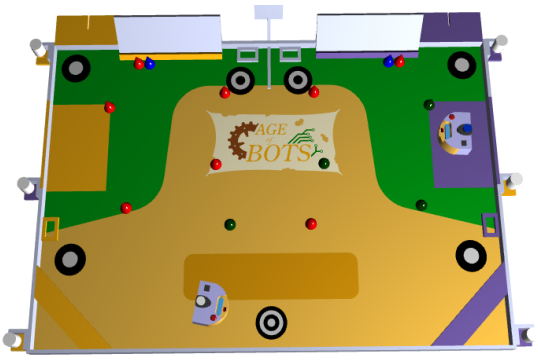
To link all the above modules, we created a FSM. The design shown in figure 10 is a simplified version of the real one.

6.2 Goal strategy

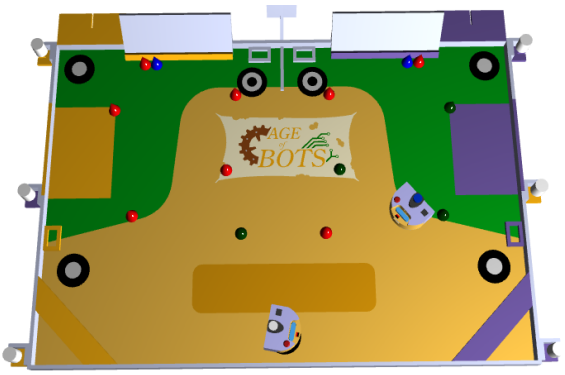
The goals are stocked in a chained list. This choice was done due to the great simplicity of implementation and because there are not a lot of winning goal combinations. Combined with a tracking of the goals stolen by the opponent, we can achieve a kind of adaptive strategy maximizing the utility function only by smartly organizing the list.

6.3 goalStolenByOpponent()

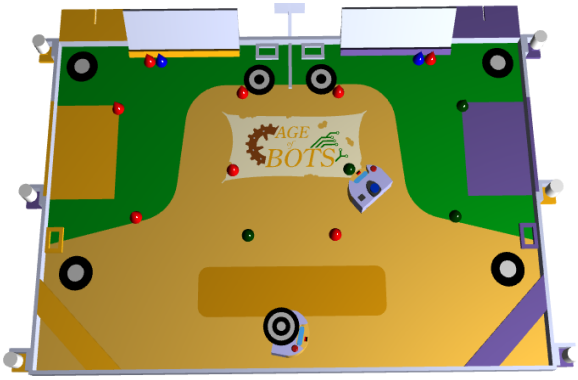
To avoid going to samples that have already been picked up by the opponent and losing time, we implemented a function that tracks the opponent's position and detects if it is on a sample. The working principle of the algorithm is illustrated in Figure 11. The goal order was initially going for the sample of 3 points value with the next sample being the sample of 2 points on the opposite side of the map regarding the robot's starting position. The robot switches from the first target to the second instantly, avoiding losing time and possibly entering in conflict with the opponent.



(a) Initial situation : the robot is in base, the opponent is close to the goal and will soon take it.



(b) The opponent takes the sample as the robot is moving towards it.



(c) It detected that the goal was taken and switches to the next goal.

Figure 11: Illustration of the working of the function goalStolenByOpponent.

7 Conclusion

During this project, we have developed the control of a robot. We first realize the low and mid-level controls. For this part, we implemented a PI controller that controls the motors of the wheels, and we implemented conversion of the global speeds of the robot to the speeds of the wheels. We then implemented the odometry to localize our robot on the map. Because of the odometry, we had to set up a calibration procedure that takes place just before the start of the game. It consists in finding two references, one for x and one for y, with two different walls.

After that, we used the Lidar to be able to spot as well as locate the enemy robots. Its implementation also allows us to use it with the path planning to elaborate a strategy. The positioning of obstacles when the robot is moving is more complex. We had to implement a compensation of the robot movement before sending back the lidar information.

The method used for path planning and obstacle avoidance is the *potential field* method. To avoid suffering from its weaknesses (local minima), we use intermediate goals between samples. Also, to avoid losing time, we implemented a function that detects if the opponent is taking a sample and removes the sample from the list of goals.

There are several things that could be improved. Instead of a potential field, we could implement an A* method (or something similar) for path planning. By implementing this, we would not have to add intermediate goals to help the robot move smoothly between the targets. Also, a Kalman filter can be implemented to reduce the error on the positioning of the robot.