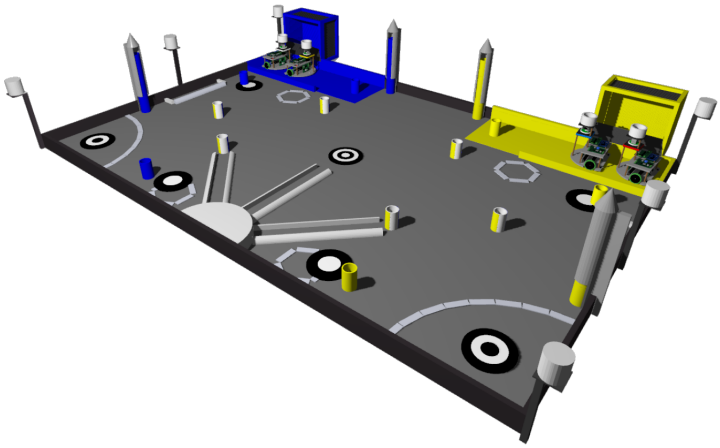
ÉCOLE POLYTECHNIQUE DE LOUVAIN

Université Catholique de Louvain

LMECA2732 - Robotics

APP PBL 2016-2017

Group 5



Team :

Berrada Yassine
Bhole Adwait
El Abdouni Soufiane
Kechtban Louai

Professor :

R.Ronsse

Academic year 2016-2017

Contents

Cooperation between our team and our friends from group 4 was established so that we can exchange some information, tricks and knowledge for this project. Our exchanges were focused on the Kalman part.

1 Introduction

As a part of our robotics course [LMECA 2732], we were asked to develop functionality for the robot, simulate the strategy and analyse the behaviour of robot. Being mechatronics students participating in the eurobot contest, we exploit this opportunity to familiarise our self with the playground as we will uses the same board for the competition. Also, we were motivated to create the most important functionality for our robot and validate the strategy we will use for the eurobot competition.

Let remind the rules for the games. On the ground, targets are place strategically in the playground. The robot need to pass by them to get as many targets he can catch and to get them back to the base. The winner is the robot which score the most point possible.

2 General description of the main controller

The main controller is divided in three functions that are called in different times :

- ***void controller init(CtrlStruct *cvs)*** : This function is called just once at the begining. It allows to initialize all variables needed for the first time step.
- ***void controller loop(CtrlStruct *cvs)*** : At each time step, this function is called to compute the new state of the robot according to the given instructions given at the previous time step.
- ***void controller finish(CtrlStruct *cvs)*** : This function is called once at the end. It is used for example to free all space allocation done during the initialisation function.

The structure *CtrlStruct* is composed of three other structures :

- ***CtrlIn*** : It represents all the inputs of the main controller (rising-falling edges detected by the tower, time, ...).
- ***CtrlState*** : This structure has been done by ourself. It contains all the variables that change states during each time step (current position in (x, y, θ) of the robot, reference speed of the wheels, ...).
- ***CtrlOut*** : It contains the outputs of the main controller (commands calculated for speed regulation).

Here, we describe just a global view of the operations. We will enter into details during the next pages. For the moment, it is important to distinguish the most important steps to accomplish :

- Implementation of a low level controller in speed to be able to move in the map.
- A system of localisation to know where the robot is.
- A system of detection of the opponents.

Once these three systems are implemented, we can work on the strategies to win points and to avoid the opponents.

3 Speed controller

In the speed controller, we use a PI controller to cancel the static error with a back emf (electromotive force) compensation to improve the speed controller by minimizing variable disturbances in the control loop related to the back emf and two limiters, as shown in figure 3.
First of all, let's calculate the load that the motor will have to overcome. By Newton-Euler method, we have :

$$F_r - F_f = \frac{M}{2} \cdot \frac{dv}{dt} \quad (1)$$

$$C_{em} - K_v \cdot \omega_m - C_f = J_m \frac{d\omega_m}{dt} \quad (2)$$

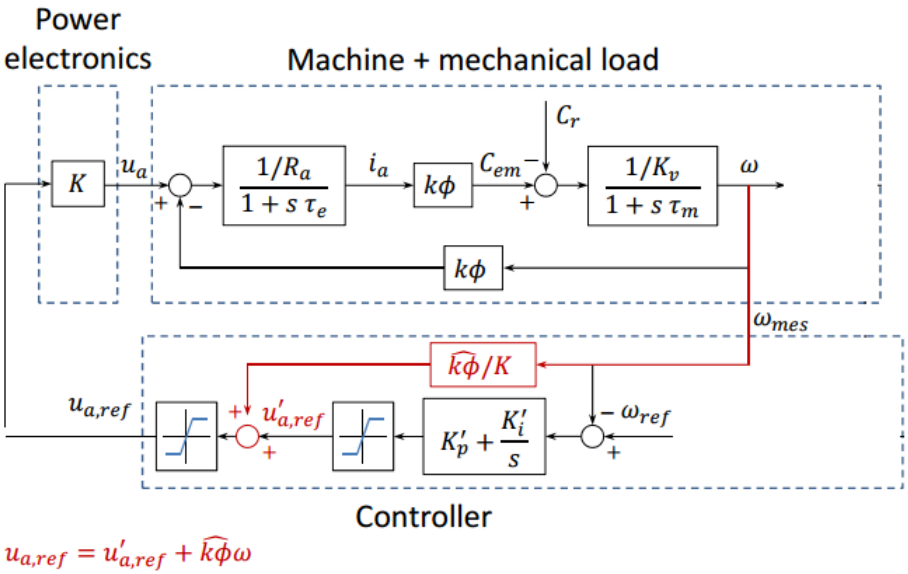


Figure 1: General design of the controller

with F_r that represents the load force given by the motor, M is the total mass of the robot, K_v is the viscous friction coefficient, F_f is the friction force and the subsribe "m" represents a motor variable.
The linear speed is related to the angular speed of the motor by the following equation :

$$v = \frac{\omega_m \cdot r}{\rho} \tag{3}$$

with r the radius of the wheel and ρ the gear ratio. By putting (1), (2) and (3) together, we can write the dynamical equation with the electromechanical torque without forgetting to take into account the efficiency and the reduction ratio of the gear :

$$C_{em} - \frac{C_f}{\eta\rho} - K_v \cdot \omega_m = (J_m + \frac{Mr^2}{2\rho^2\eta})\frac{d\omega_m}{dt}. \tag{4}$$

After the dynamical aspect done, we have to tune the controller. To do so, we used the discretized PID block with the same time step from the project simulation ($\Delta t = 0.001$ $[s]$) from SIMULINK (MATLAB software) connected to the DC motor with limiters and the back emf compensation. We compare the step response of the SIMULINK bloc with the data from the Minibot project and we validate the K_p and K_i found thanks to the software (figure 2 has an angular speed reference of 10 $[rad/s]$).
We then found

$$K_p = 1.535 \tag{5}$$
$$K_i = 1.033 \tag{6}$$

To integrate this implementation in the main controller loop, we created a function called ***void controller iterate(CtrlStruct *cvs)*** that calculate at each time step the commands to give to the motors.

4 Localisation

To be able to know the position of the robot, we implemented two different localisation systems : **odometry** and **triangulation**. The odometry has a small error that increases constantly with the covered distance while the triangulation is precise at rest but has a quite big constant error during the displacement. The idea is to combine the two systems to have a more accurate localisation : the **Kalman filter**.

4.1 Odometry

The odometers provide the angular displacement $\Delta\omega_L(t_n)$ and $\Delta\omega_R(t_n)$ of each of the wheels during period n. On the basis of these two values, the elementary displacements of the robot in rotation $\Delta\theta(t_n)$ and along its trajectory $\Delta S(t_n)$ are given

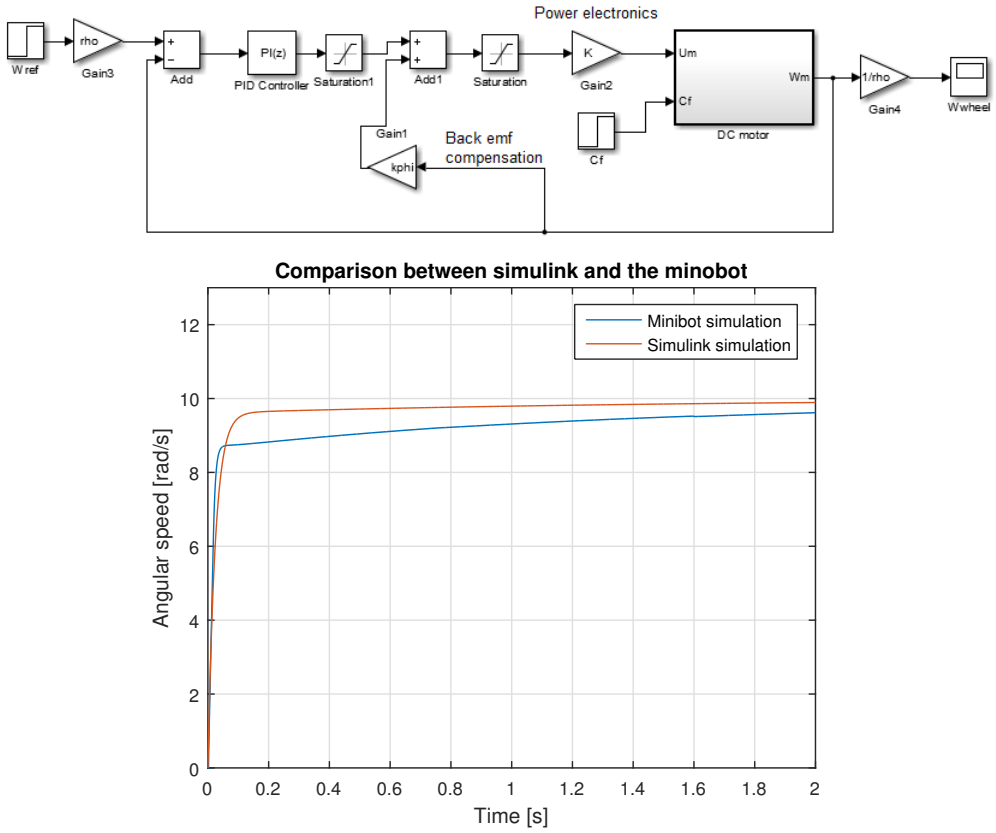


Figure 2: SIMULINK design of the PI controller and the comparison between the Minobot and the SIMULINK simulation

by

$$\Delta S(t_n) = R \cdot \frac{\Delta\omega_L(t_n) + \Delta\omega_R(t_n)}{2} \tag{7}$$

$$\Delta\theta(t_n) = R \cdot \frac{\Delta\omega_R(t_n) + \Delta\omega_L(t_n)}{b} \tag{8}$$

where R is the radius of one wheel and b is the distance bethween the wheels.
After that, we can compute the new position at the next time step according to the position in the previous time step :

$$x(t_{n+1}) = x(t_n) + \Delta S(t_n) \cdot \cos\left(\theta(t_n) + \frac{\Delta\theta(t_n)}{2}\right) \tag{9}$$

$$y(t_{n+1}) = y(t_n) + \Delta S(t_n) \cdot \sin\left(\theta(t_n) + \frac{\Delta\theta(t_n)}{2}\right) \tag{10}$$

$$\theta(t_{n+1}) = \theta(t_n) + \Delta\theta(t_n) \tag{11}$$

The odometry is a precise localisation system as far as we can calibrate the robot and we can avoid any slipping. To calibrate the robot, we have to use two axis that are perpendicular each other. Avoid slipping is not possible to eliminate ⇒ non-deterministic error.

4.2 Triangulation

This type of localisation is hard to implement. The idea is as follows : the tower rotates at a certain angular velocity (we fixed the command of the tower to 15) with a laser that emits light that is reflected on 3 beacons. Thanks to the reflexion, we can collect rising and falling angles. These angles will be used to calculate to estimate the position with the triangulation algorithm[1]. The problem with this localisation is that we can't directly associate the mesured angles with the beacons. For example, the first angle mesured is not necessarily associated to the first beacon. To resolve this problem, we used the following tests :

1. Store the three angles mesured

[1] `http://www.telecom.ulg.ac.be/triangulation/`

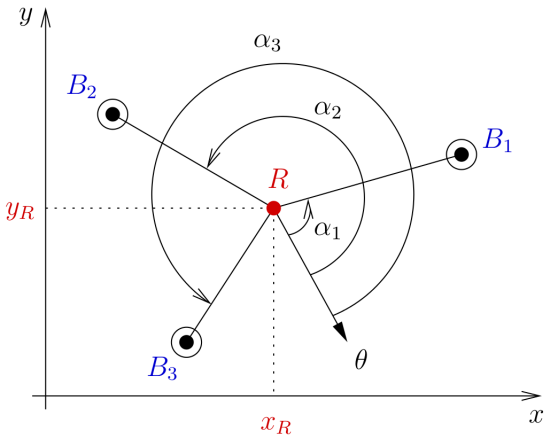


Figure 3: Schematic of the triangulation

2. Compare them with estimate angles with the help of the odometry
3. Map the i index of the mesured angles with the j estimated angles by minimizing the errors

To estimate angles according to the beacons, we will use some trigonometric identities. If we know the position of the robot, we can calculate the angle α_1 as follows[2] :

$$\alpha_1 \quad = \quad \arctan\left(\frac{y_{B1} - y}{x_{B1} - x}\right) - \theta. \tag{12}$$

For our algorithm, we will use (12) to estimate the α_i thanks to the odometry. We thus have :

$$\epsilon_{ij} = \alpha_i - \arctan\left(\frac{y_{Bj} - y}{x_{Bj} - x}\right) - \theta \tag{13}$$

By comparing each line of the matrix ϵ, we find the mapping bethween i and j by minimizing each line. For example, if the minimum of the line one is in the column 2, we have the following mapping : the first mesured angle is associated with the beacon 2.

We have to be carreful about a last problem. Because the angles are wraped bethween $[-\pi, \pi]$, we have to treat the case when the angle mesured and an angle estimated are closed around π (figure 4), that is, one in the second quadrant and one in the third quadrant. Indeed, the discontinuity at this point makes the error around 2π $[rad]$ whereas it should be the minimum error. So, we have to put a condition and to say that all ϵ_{ij} around $[6.2, 6.4]$ is minimum.

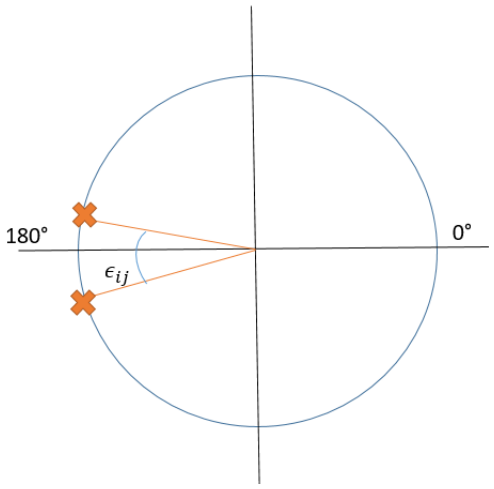


Figure 4: Illustration of the discontinuity

[2] https://orbi.ulg.ac.be/bitstream/2268/89435/1/Pierlot2011ANewThreeObject.pdf

4.3 Kalman filter

The Kalman filter is a real time algorithm that combines series of measurements to produce a precise estimated value which might be more precise than using a single measurement. For this kind of filter we consider the Gaussian probability where it is used to define the probability density of the function by using the mean and the variance. For our purpose we will implement the Extended Kalman Filter which is a linearization of the original non-linear filter because the trajectory of our robot is definitely not a linear system.

4.3.1 Extended Kalman Filter

Because we consider 3 dimensional coordinates, we will use a vector $[3 \cdot 1]$ elements for the positioning of the robot (the mean) and a $[3 \cdot 3]$ vector for the P matrix (the covariance).
In our system, we are interested to fuse 2 types of estimated position from different sources. The odometer where x, y and theta can directly be derive the from it, and the laser tower where we need to calculate the angles $\alpha 1$, $\alpha 2$ and $\alpha 3$ based on the its orientation and position of the robot as respect of the beacons position (see the previous section for more information).

To be able to implement a kalman filter in the system we need to follow some important steps:

1. **Prediction**:
   X represent the position estimated by the odometer (equation [14]). We need also to estimate the covariance error represented by the P matrix (equation [15]). As we can see the covariance is the addition of the elements. It is essential to estimate the uncertainty of the odometer, but for the purpose of this project the uncertainty was given for t=0. Note that, the uncertainty will change at each iteration. The Q represented in the equation[15] represent the motion errors covariance matrix. As we assume that both wheels are independent from each other, the Q is diagonal and can be represented as:
   $$Q = \begin{pmatrix} Kr \cdot \delta Sr & 0 \\ 0 & Kl \cdot \delta Sl \end{pmatrix}$$
   where Kr and Kl are the error constants representing the non deterministic parameters of the motor drive and the wheel-floor interaction, the δSr and δSl represent the distances travelled by each wheel and ∇F, ∇F_u are the two transformation jacobian matrices from the last position space to the new one and from wheels speed space to the position space respectively.

$$X_{k/k-1} = F(x, u) \tag{14}$$

$$P_{k/k-1} = \nabla F \cdot P_{k-1} \cdot \nabla F^T + \nabla F_u \cdot Q \cdot \nabla F_u^T \tag{15}$$

2. **Prediction base on odometer:**
   Because the 2 positioning technique give us different approach for the positioning, we actually need to translate one of them to match the other one. We decide to use the odometer results to match with the triangulation. We use the following matrix to translate the x y and theta to estimated angles.

$$h = \begin{pmatrix} \arctan(\frac{Y1-y}{X1-x}) - \theta \\ \arctan(\frac{Y2-y}{X2-x}) - \theta \\ \arctan(\frac{Y3-y}{X3-x}) - \theta \end{pmatrix} \tag{16}$$

   (X1,Y1);(X2,Y2);(X3,Y3) represent the position of each of the 3 beacons on the table. As we can note the unit of the h matrix is in radian

3. **Measurement:**
   We use the triangulation technique to get the the actual angles Z where Z is the matrix $[3 \cdot 1]$ containing the 3 angles from the triangulation. After that we would be able to compute the error using the equation bellow. The unit of the error is in radian.
   $$Y = Z - h$$
   After calculating the error we need to calculate the projection of the system uncertainty into the measurement space by using the equation bellow equation[17]. As you can note the S matrix has unit of $\frac{rad^2}{m}$
   $$S = H \cdot P_{k/k-1} \cdot H^T + R \tag{17}$$

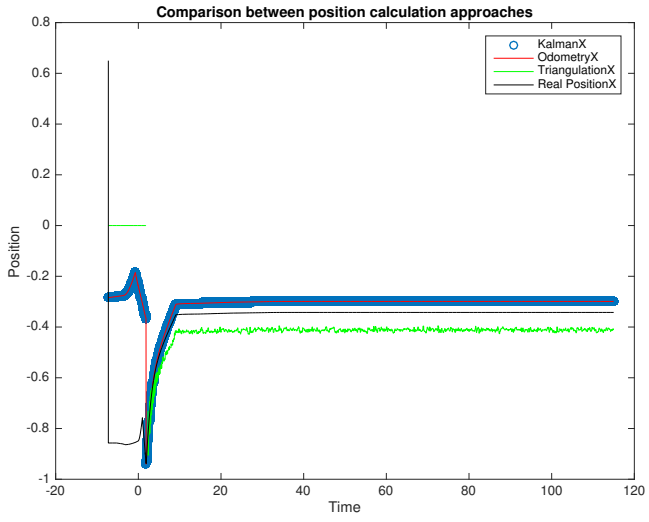


Figure 5: Application of the Kalman's filter

H is the jacobian matrix of h to respect of x y and theta. The equation[18] bellow is the gain K, the most important part of the kalman which will automatically configure it self according to the previous values. The unit of the gain is $\frac{m}{rad}$

$$K = P_{k/k-1} \cdot H^T \cdot S^{-1} \tag{18}$$

4. **Correction:**
   We use the equation[19] to compute the new estimated position of the robot and equation[20] to compute the new uncertainty. Through time, our uncertainty ellipse will vary. We would observe a continuous growth in the ellipsoid which represent the expectation odometer localisation system.

   As we can observe here the gain K is multiplied by Y which result the value unit in meter which mean that $X_{k/k}$ is in meter

$$X_{K/K} = X_{K/K-1} + K \cdot Y \tag{19}$$

   The new uncetainty is calculated using the following equation[20]

$$P_{k/k} = (I - K \cdot H_x) \cdot P_{k/K-1} \tag{20}$$

5. **Results:**

   We ask the robot to go from the base to a certain target(-0.3;0.7). As shown in the figure 5 we observe good correlation between the results of the Kalman Filter the real positioning of the robot. Unfortunately even if the kalman filter developed for this robot look like perfect, we observe most of the time bad results due to the noisy triangulation specially at border of the playground and because of the bad choice of the gains. We work very hard to find a good way to choose best gain for the kalman. We found good gain for the x positioning but not for the y and theta due to the limited time. That is why we decide skip the Kalman filter and base our robot only on the odometery since the odometer show good results.

5 Navigation

The navigation is a part of the middle level controller, it allow to calculate the velocity of each wheel to arrive to the the target.
We know that the velocity dynamically changes to follow the trajectory, and regulate the speed we use the PI controller from the low-level controller.
To do a good navigation, we need to know the current position of the robot(from localization) and the target position

We can see a picture represent the problem statement in the figure 6.

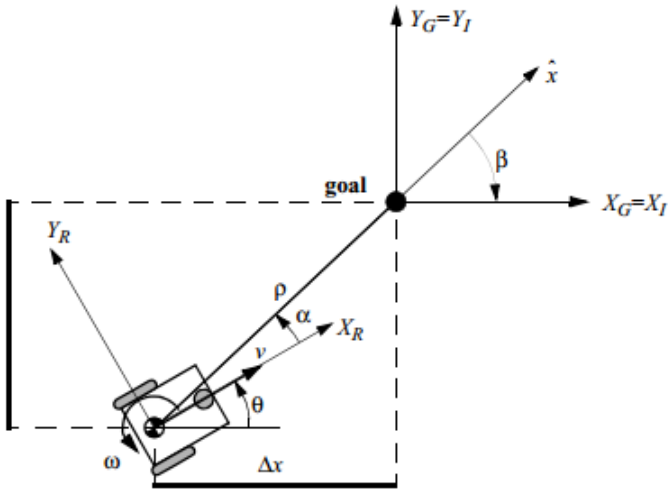


Figure 6: Navigation

When we have (X_{cible}, Y_{cible}) and $(X_{robot}, Y_{robot}, \theta_{robot})$, we can calculate the distance and the angle between the robot and the target thus with this value we can found the speed of each wheel.

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \tag{21}$$

$$\alpha = -\theta + \arctan \frac{\Delta y}{\Delta x} \tag{22}$$

$$\tag{23}$$

Where Δx and Δy are the difference between the robot position and target position. θ is the robot angular position.

The angular velocity for the left wheel and the right wheel is given by:

$$\omega = K_\alpha \cdot \alpha \tag{24}$$

$$v_{left} = v_{ref} - l \cdot \omega \tag{25}$$

$$v_{right} = v_{ref} + l \cdot \omega \tag{26}$$

$$\tag{27}$$

where $K_\alpha = 1$ and V_{ref} is the linear component of the robot while $(l \cdot \omega)$ is the rotation component of the robot orientation $\frac{d\theta}{dt}$ (l is the distance between the two wheels of the minibot)

As we use the A* path planning we follow the paths given by the A* linearly, because the motion might be unpredictable if it moves in curved trajectories.
Thus for to have a good control of the robot way we doing a rotation until the target is on the front direction of the minibot face.Thus we consider that the target is in the face if the angle is between $-15 < \alpha < 15$. For to do this rotation, we give $Vref = 0$.
And after when we are in front of the target we fixed the v_{ref} at a value it that allow to approach the target. Next, we set the condition to stop linear navigation if the target is within 0.06 mm from the current position.

6 Path planning

6.1 Motivation of choice

The path planning is called a *high level controller* because its implementation uses all others low level and middle level controllers (odometry, speed controller, navigation).

Before talking about the algorithm, let's motivate our choice.

Visibility graph: This method appealed as a stupid algorithm since it takes a path longer than necessary. Also there is a risk of collision with the obstacle.
The potential field: The robot has a chance to be trapped into a local minima if a robot is surrounded by obstacles (for example at the base) . We discovered from previous groups that it was hard to tune the controllers for optimal opponent avoidance.That is why this method was rejected.
A* algorithm: A* is a node based algorithm. The program is explained later. The heuristic approach of A* always leads to a path if there is one possible.

The drawbacks of **A* algorithm** are not significant in the application as much as its advantages. Hence, we used the A* algorithm for path planning.

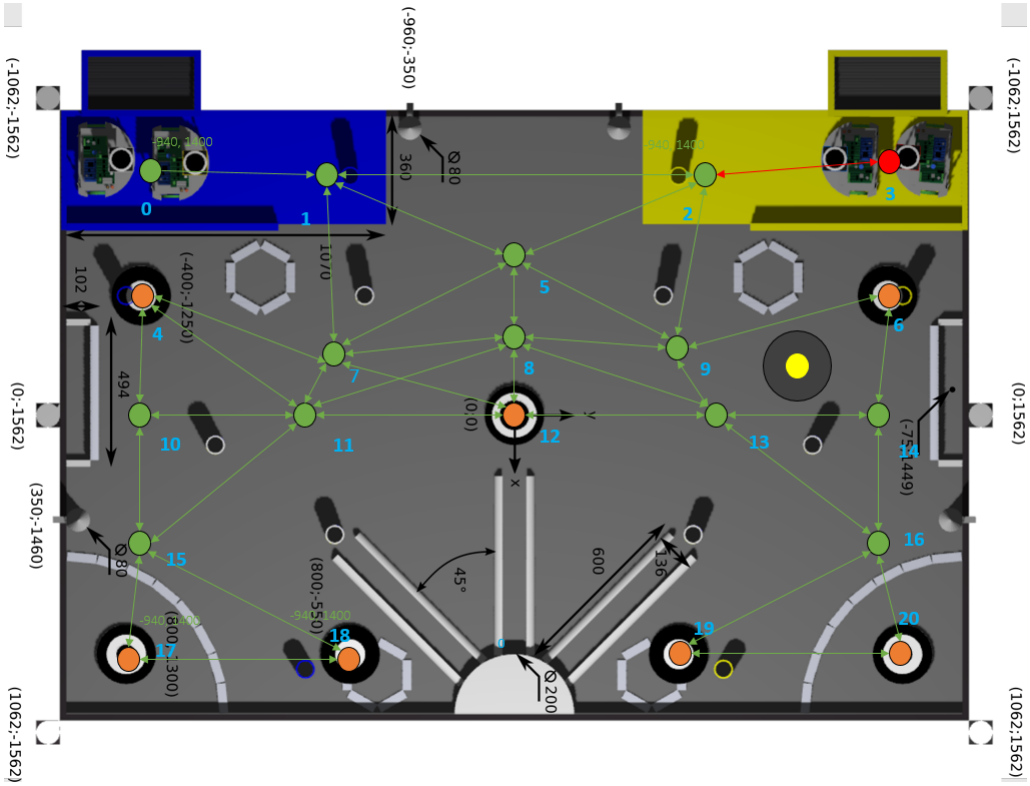


Figure 7: Nodes used for the A-star algorithm

The objective is to move from node to node to finally find a path from the current node towards the target.
The nodes have been placed at strategic points as shown in fig 7 on the map. The green nodes show walkable nodes while the orange nodes are the targets. Every node is set such that they are far enough from the steady obstacles such as the walls. The node connectivity is done only if there is no obstacle in the connecting path (for example there is no path shown between node 4 and 1). The red nodes and red paths are not walkable. The strategy is to use self-defined node based graph rather than a conventional grid.

For the eurobot competition we are going to change the position of the nodes because the targets are at different positions. Also we shall use more number of nodes (to get more resolution) especially in the region of the targets. The reason is that on the simulator, the minibot had to only place itself at the position of the targets, wait for some time and return while in the robot it has to correctly orient itself as it approach to grab the target. This will be based on position and the working of our modules. The size and shape of our robot is significantly different from that of the minibot.

6.2 Small description of our version of A* algorithm

Our implementation of **A* algorithm** uses only two structures : the map structure and the node structure. We have the following for each structure.

- Node :
  - X and Y coordinates
  - f, g and h value
  - walkable boolean
  - open and closed lists
  - pointer to the parent node
  - adjacent nodes
- Map :
  - list of nodes
  - number of nodes
  - list of nodes calculated by A*
  - number of nodes calculated by A*

The search algorithm has two lists (set of nodes), the open list and the closed list. The current node is initialized by the starting node (the minibot node position). The closed list are the set of nodes included in the path towards the target while the open list is the set of nodes that are on search (and might be used later).
The grid based search algorithm uses two cost functions : the path cost $g(n)$ and the heuristic $h(n)$. The total cost $f(n)$ is the deciding factor to include a node in the closed list.

Every node on the open list has its respective f, g and h cost values. The search starts by searching the adjacent nodes of the current node. We have defined the adjacent nodes for every node in the structure node.

g is defined as the distance from the current node to the search node on open list :

$$g = \sqrt{(X_{current} - X_{adjacent})^2 + (Y_{current} - Y_{adjacent})^2} \tag{28}$$

h is the heuristique. We have a lot of choices to estimate this distance. We chose the so called Manhattant distance that is calculated considering the movement first in x-direction and then in y-direction only (not the direct path).

$$h = |X_{target} - X_{adjacent}| + |Y_{target} - Y_{adjacent}|. \tag{29}$$

The total cost is

$$f = g + h \tag{30}$$

The node with lowest f value is taken on the closed list with current node set as the parent of that node. Now we iterate the search process updating the current node being that node. The search process is iterated until the target node is included in the closed list. The set of parent nodes tracing back from the target is finally the path towards the target.

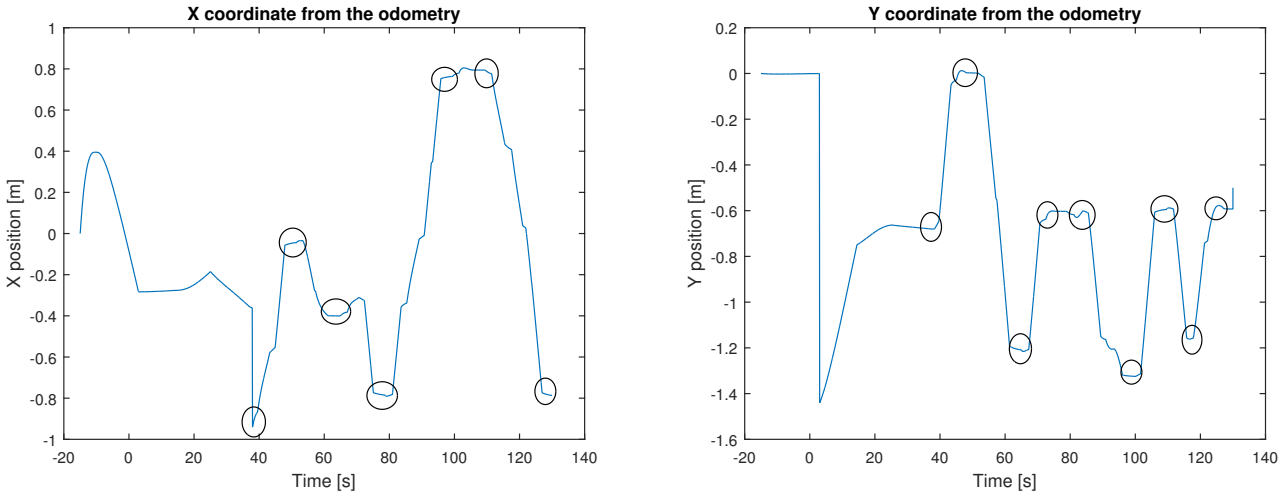


Figure 8: Plots of X and Y coordinates from odometry during strategy simulation

The figure 8 shows the (X, Y) coordinates of the robot during the strategy simulation. As shown in the figure **??**, the robot follows the nodes chose from Astar algorithm. Refer to figure 7 (for the coordinates) to convince yourself with the results.

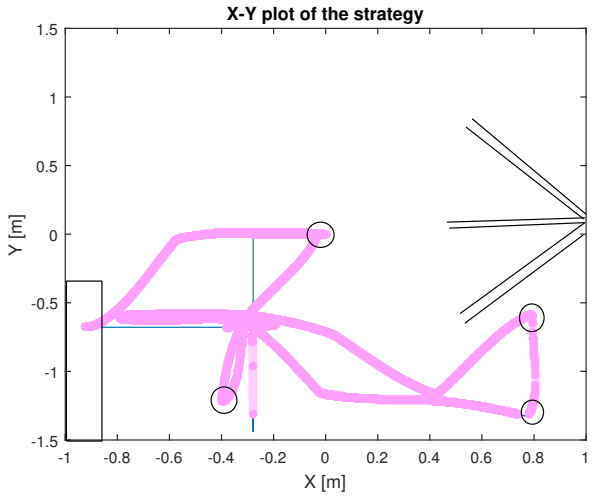


Figure 9: X-Y plot of the strategy

6.3 Opponent Avoidance strategy

Dynamic obstacle avoidance is important, because it is a conservative way to save the points earned (negative points for the robot colliding with higher speed).
In our strategy if the enemy robot is in our path say in the path of target 1, we return to the previous node. Then we reject the target 1 and we set a new target say target 2. Accordingly, a new path is developed using the A* with the source of search being the node preceeding the realization of presence of the opponent. refer fig 12.
As shown in figure10 the robot first aimed to take the target in the center as shown in orange.
It then realized that there is the opponent robot present in its path. Then it goes back to the previous node(shown in blue) and sets the new target (the one on the top left).

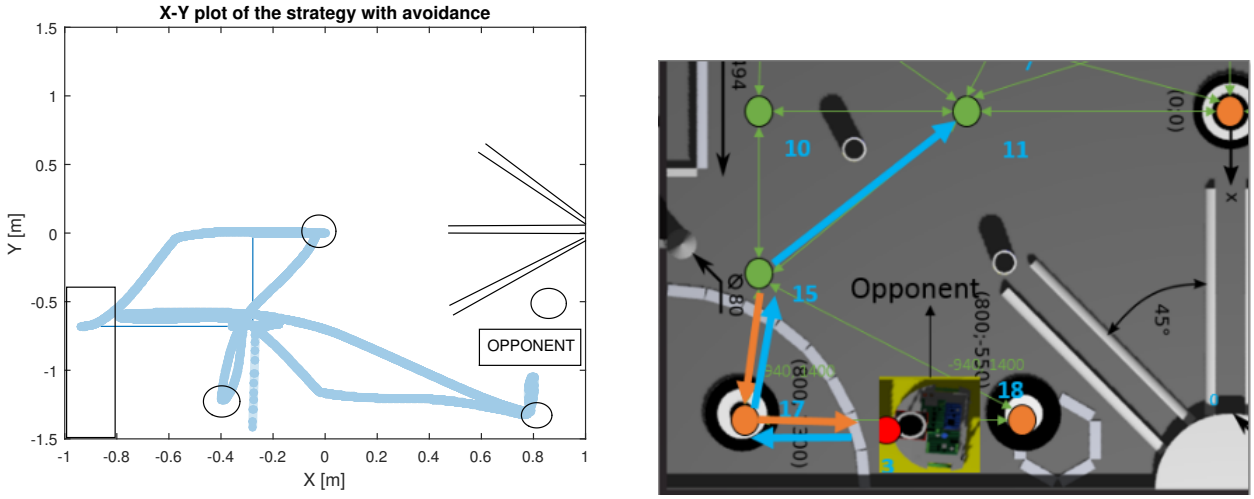


Figure 10: Showing the change of target due to presence of dynamic obstacle and the X-Y plot of the strategy with the avoidance

While implementing the obstacle avoidance method, we found that the output of the opponent position estimation contains lot of noise. To reduce this noise, we have implemented a low pass filter in the output. The output of the filter is shown in the graph.

7 Calibration and the Finite State Machine

7.1 Calibration

At start, after placing on the play area; the minibot is unable to estimate its current position. To get an initialization for the odometry values, calibration is necessary. The algorithm used for calibrating the minibot is described in fig

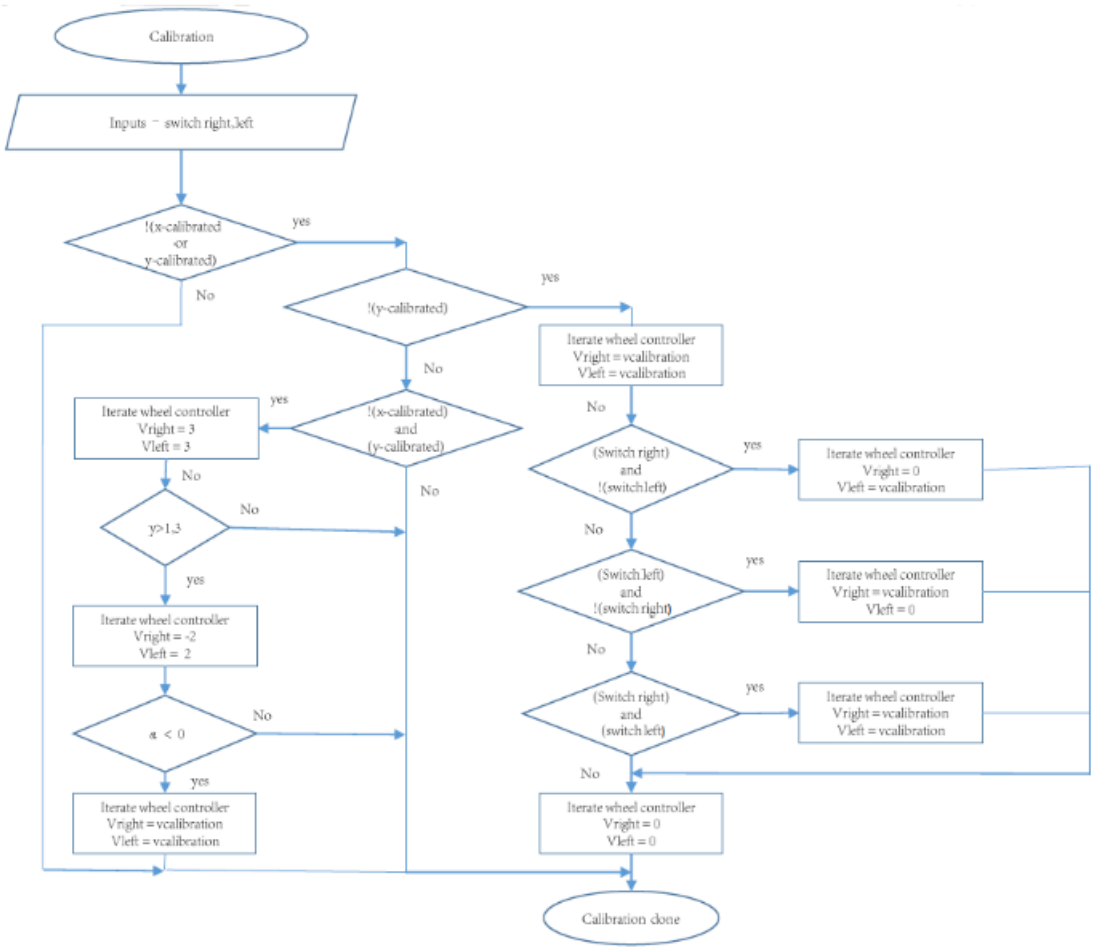


Figure 11: Flowchart showing the position update method

7.2 FSM

The strategy to grab the target, return to base and rejecting to approach targets(if dynamic obstacle is present in path) is done using an FSM. The flowchart shows the algorithm.

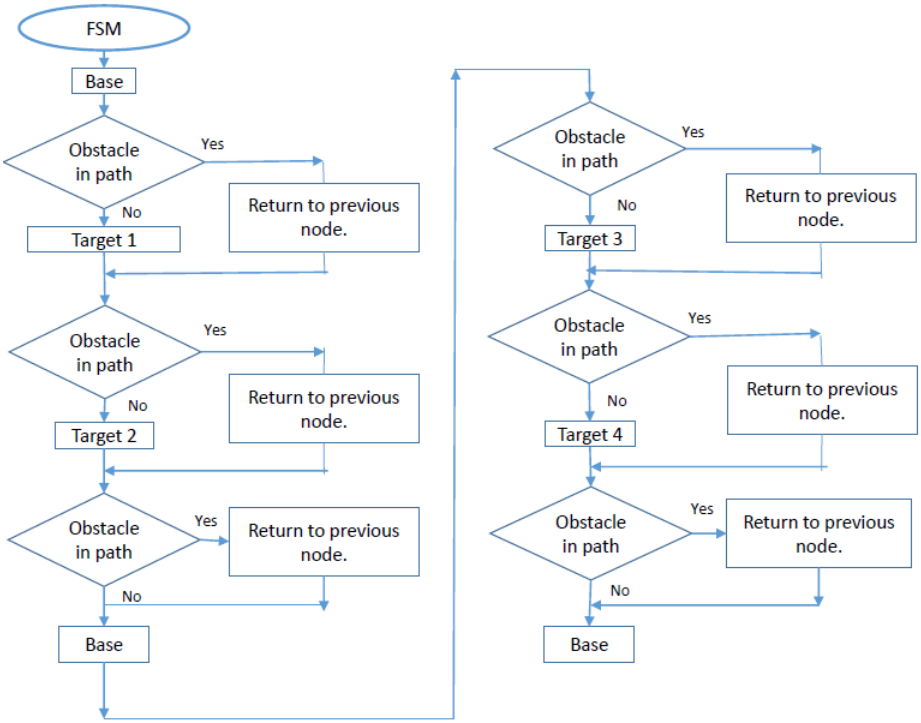


Figure 12: Finite state machine implemented on the controller

As shown in the chart, the robot will start by calibrated it self in the base. After the calibration, the robot will get outside of its base and then will face 2 options. If the robot does not face any obstacle the robot will follow its normal path, for example from the base it will go then to get the target 1 then to the target 2 and comeback to the base to release the target then go back to get the target 3 and 4 and comeback to the base. If on the other hand the robot face an obstacle, the robot will go back to its previous node position and instead of doing its task, the robot will skip the next task.

8 Conclusion

This project was a real benefit for us, as mechatronics student, which participate in the eurobot competition. We had the possibility to implement functionality such as odometery, navigation, and path-planning and following using A* and kalman filter which took sometimes to be perfectionized for the simulation. The simulation work perfectly so it can be used in our real robot. Of course we won't be able to use the kalman in our robot since we don't have 2 different sensors to localize our self on the board.