Intelligent And Mobile Robotics
Development of an agent for a robot that follows a line

Diogo Silva (dbtds@ua.pt)
DETI, University of Aveiro
Aveiro, Portugal

***Abstract* – This paper describes a robot developed for Intelligent and Mobile Robotics at the University of Aveiro. The main objective of the robot is to follow a line (track), which might have some obstacles and track interruptions with the help of some sensors and actuators. Moreover, this project was organized in a sequence of challenges that the robot has to fulfil. This paper covers the description of the robot, the respective challenges and the software that was developed in order to overcome the challenges. Its focus will be on the software developed and not on the robot's hardware.**

***Keywords* – robotics, agent, ua, rmi, robot**

I. INTRODUCTION

This project was created for the course unit Intelligent and Mobile Robotics at the University of Aveiro. It consists in developing an autonomous agent, which is a system that tries to understand the environment using sensors and acts according with it using actuators (described in the section III).
This agent (robot) will have to follow a line and pass several challenges that are described in the following sections. Besides the description of the problem, the strategies adopted to fulfil every challenge are described, e.g. strategy adopted for the robot deal with the tight curves. Moreover, the focus of this assignment was on the source code.

II. SCENARIO

The scenario is where the robot plays its role to fulfil every challenge that was proposed.

A. Description

The scenario is composed by several different details.
It is a white line track which the agent should follow in order to fulfil its objective, like it is shown on image 1. The line has to have the opposite effect from the background to the sensors that detect it, e.g. if the line is white, it will reflect to the ground sensors and be detected, so the background must be a color that absorbs.
On the image 1 there is a perpendicular line to the white line, which is the starter area of the robot. It will stay on it until the challenge begins.
The track is also composed by intersections of the line where the robot must go in front. There will also be tight curves where the robot will have to adjust completely its direction and track interruptions where the robot will have to stay in the same direction that had when the track was interrupted, because there are only interrupts in a straight direction and with a maximum length of 25cm.

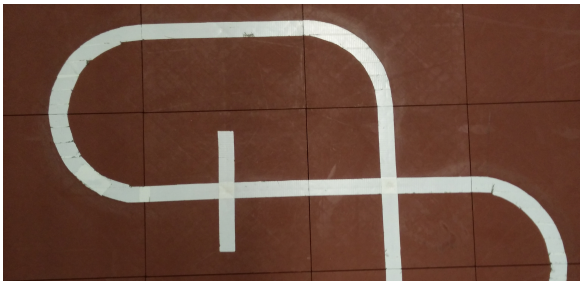

Figure 1: White line with the start zone (perpendicular line)

Besides this description of the track, there might be some obstacles in the track where the robot will have to detect and circumvent them in order to find the line again.
On the document with the problem description there is no restrictions related with the characteristics of the obstacles, so the robot should be able to handle any obstacle that is viable to find on the scenario.

B. Challenges

This section is exactly the same content that describes the challenges in the document provided in the course unit Intelligent and Mobile Robotics at the University of Aveiro.

"The list of challenges will be:
1. Simple navigation in scenario number 1. The robot is positioned just after the start line and, once started, it has to perform 2 complete laps, turn around, complete one more lap in the opposite direction and finally stop at the start line.
2. Navigation in scenario number 1 with an obstacle. An obstacle will be placed over the track line, in a straight segment. The robot should perform as in the previous challenge but overtaking the obstacle. So, the obstacle should be detected and contoured in order to regain the line, avoiding any collision.
3. Navigation in scenario number 2. The robot is positioned just after the start line and, once started, has to perform 2 complete laps and stop at the start line. The track has several intermediate crosses and some tight curves.
4. Navigation in scenario number 2 with track interruptions. The robot has to handle track interruptions (at least

one with a minimum of 25 cm). It has also to deal with sharp turns in some parts of the track."
The scenarios are completely unknown, only the characteristics of it are provided.

III. BRIEF DESCRIPTION OF THE ROBOT'S HARDWARE

The following image represents the sensors and actuators of the robot used.

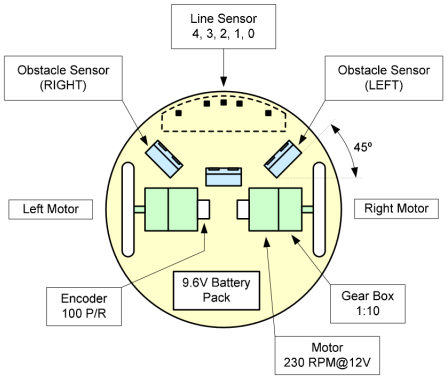

Figure 2: Robot hardware.
Source: Project description documentation

It is composed of sensors that are used to perceive the world. The line sensors on the top of the image are used to detect the white line on the ground, that the robot uses to guide itself.
There are three obstacle sensors in the front of the robot, one pointing to the left, one to the right and a last one pointing in front. Those sensors are going to be used to detect and overcome any obstacle that might be on the track.
Also, there are encoders that allow to check the relative position of the robot since it has started.
Finally there are two motors that control the wheels. Those are the actuators that allow the robot to move in the environment.

IV. ROBOT ARCHITECTURE

The robot is a reactive agent with some internal states. Those states are FOLLOW_LINE and OBSTACLE. There are only those two states because the problem was divided in such way that if there are no obstacles, it just has to follow the line. Otherwise, it will have to circumvent the obstacle. When there is special cases like tight curves, the curve is still there. So it just needs to be detected and then follow it like the normal line.

In spite of existing only those two states, there are some checks that are always done independently of the current state of the robot. Those checks are described in the flow chart of the system in the following section.

Robot has a small buffer (100 entries) implemented that allow to store the last entries of the ground sensors (which are the line sensors mentioned before). It does help to detect some tricky situation to make sure that it is really happening. For example, when the robot is crossing the perpendicular line or a cross, it checks for the 30 positions in the buffer to make sure that it is a cross or a perpendicular line. The way that the robot differentiates the perpendicular line from the cross is discussed later.

A. Main flow of the system

The following image 3 represents the current main flow of the system, it does not include every single detail about the system, e.g. what happens when an obstacle is detected by the front sensor but it is on the other side of the line and the robot is in the state FOLLOW_LINE. Those aspects are not covered in this flow diagram.

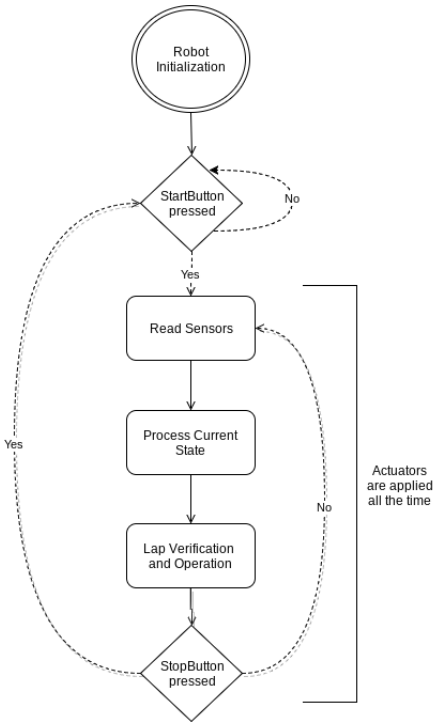

Figure 3: Flow of the system

After the robot initialization, it will wait for the start button. When the start button is pressed, it will enable all sensors and start reading them when possible.
After reading the sensors, it will process the current state and apply new rules to the actuators if needed. The initial state is clearly FOLLOW_LINE since the robot starts on the perpendicular line which was shown previously on the figure 1. Every time that an object appears close enough (Section VII), it will change the state to OBSTACLE. After it finish to circumvent the obstacle, it will return to the initial state.

After processing the state there is a condition that is always checked independently of the state, which is the lap verification.

V. LAP VERIFICATION

A new lap starts every time that the robot crosses the perpendicular line on the start and there is even a challenge

that it must rotate on the start of the third lap.

A. Detection of the lap ending

One of the detections that have been done is to find when there is a perpendicular line. That was done by looking at the ground buffer where the values are stored and check if there is a combination of situations where all ground sensors have detected line, if it is found, it means that there is a perpendicular line.

Although there is a problem. Perpendicular lines and crosses look exactly the same to the robot doing only that verification.

To solve the problem, it was a must to use odometry which consists of knowing the relative position of the robot since odometry has been working. The robot takes the current position on the start and when it detects a perpendicular line or a cross it checks if it is the start position that he has stored or not, with odometry.

B. Odometry Error

Odometry always has an error associated with it. It has several situations that generate more error on the odometry, one of them is slippage from the robot or agressive movements that the robots make on the ground not allowing the wheel to rotate perfectly. The other situation is the approximation that is done when odometry is applied, it assumes that the robot always make straight movements, which is not true, the robot rotates. Alhough it considers tried to minimize that error as possible by doing small movements calculations.

When the robot compares with the initial position, it must take this into account. So it takes 12.5cm (which is half of the distance to the next possible cross) into consideration when comparing to the initial position. **If it is lap, it also resets the odometry position to RESET the error.**

VI. STATE FOLLOW_LINE

This state corresponds to the function follow_line(). It presents the solution for several problems, such as:

1. Following the line correctly without leaving it and efficiently
2. Following the line even during tight curves
3. Track interruptions
4. Obstacle detected in other zone that is not close enough
5. Ground sensor gain

A. Following the line correctly - Controller

The strategy used to follow the line was to create a controller, in this case a proportional. Which was enough to fulfil the expectations, just like the image 4 shows.

To be more precise, the E(s) is calculated using an average of the ground sensors weights have caught line.

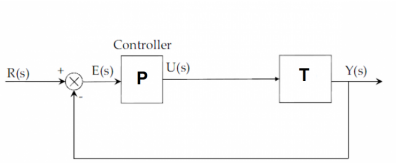

Figure 4: P Controller. **Source:** *http://cs4hs.cs.pub.ro/wiki/roboticsisfun/chapter5/ch5_1_optimizations*

For example, if the weights are -3, -1, 0, 1 and 3 where -3 corresponds to the most left sensor and 3 to the most right one. If the all the three sensors on the left detect line, the error will be the following:

$$((-3) + (-1) + 0)/3 = -1.33$$

Which means that the robot should turn to the right a little. The robot has a KP of 19 and **BASE_SPEED of 60**, which is fast. It means that for the previously case the speeds set on the motors will be $(60 - 19 * (-1.33), 60 + 19 * (-1.33)) = (85.333, 34, 667)$.
It will turn to the right as mentioned before.

B. Following the line even during tight curves

Tight curves were solved by increasing drastically error when those curves are detected. Every time that there is no line detected (it includes track interruptions, it will be discussed in section VI-C) it will enter in a routine which calculates the weight of all buffer of the ground history for the most left ground sensor and most right.
Weight decreases according to the entry's age on the buffer (older entries are less relevant).

```
for (i = 0; i < GROUND_HISTORY; i++) {
   if (ground_buffer[i][G_MR])
     right += (GROUND_HISTORY - i);
   if (ground_buffer[i][G_ML])
     left += (GROUND_HISTORY - i);
 }
```

After that weight has been calculated, the robot will be able to decide if the weight is high enough in on the sensors to start turning or to discard it.

If the weight is enough in one of the sensors, it means the last side that the robot has detected was the one with more weight calculated. So it will increase the error on the P controller by 9 times, which will force the robot to turn into that side until it finds a line.

C. Track interruptions

Track interruption was solved by considering that the default situation is when the robot **does not find any line** and **there is no recent detections on the most left and most right ground sensors**. So the robot will just **stay** on the same direction.

D. Ground sensor gain

During the first tests when the speed was around 40 to follow the line, the robot was moving slow enough giving enough time for the ground sensor to absorb value.

Although when the base speed of the robot was increased to 60, the ground sensors did not have enough time to read valid values from it. So we had to reduce the gain for them to work with such a speed. The final gain value was set 30 (default is 50).

E. Obstacle detected in other zone that is not close enough

There is a situation that should not allow the robot to change the state to the OBSTACLE, which is when the robot does see an obstacle, but the obstacle is not in the same line that its is following.

Take into consideration the image 5.

Figure 5: Obstacle on the other line

If the robot is coming from the bottom left and he will look forward when doing the last curve, he might see the object in the other line with the front obstacle sensor.

Although it cannot enter in the OBSTACLE state. So what the robot does when he sees an obstacle is to reduce the speed by half and waits to keep close to the obstacle till he confirms that he can change the state, or it was a false positive and the robot will keep following the line.

VII. STATE OBSTACLE

The robot will only enter in this state after getting really closer to the obstacle. This state corresponds to the function dodge_obstacle()

This state consists of several parts until circumvent the obstacle. It is internal organized in state.

A. CENTER and ROTATE_CENTER state

The center state consists in making sure that the robot gets close enough to the wall to follow it, after getting close enough it will change state to ROTATE_CENTER where it will rotate to be parallel to the wall and switch again to FOLLOWING_WALL. After this, it is impossible to return to these states until the obstacle gets circumvented successfully.

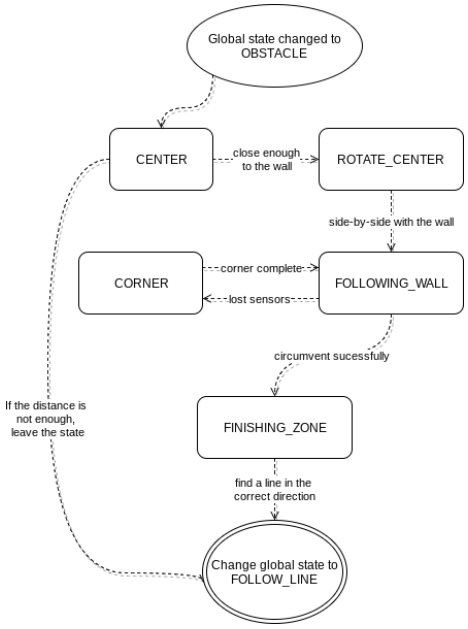

Figure 6: Flow chart of the state OBSTACLE

If the robots gets inside the CENTER state and it is still too far from the wall, it will return to the global state FOLLOW_LINE until get close to the obstacle.

B. FOLLOWING_WALL state

This state consists in following the wall, as the name says, by using the wall on his left and with the help of the obstacle sensors that detects first that wall. It is implemented with a P controller where the distances between the center the robot and the left sensor accepted without any error are 20 cm to 22 cm, otherwise it will try to compensate until reach those values.

C. CORNER state

This state is like an emergency state when the robots fails to detect the wall and the obstacle sensors says that there is no walls on the left to follow.
Most of the cases it means that the wall has finished in there and it is a corner, so it must go forward a bit and turn again to the left until finds a wall on the left again. Then the state will change again to FOLLOWING_WALL.

D. FINISHING_ZONE STATE

The finishing zone is when the robot has already found the line that he is supposed to follow, so it turns to the side that he is supposed to start to follow.
The strategy was to follow the wall on the left, so on the end of the wall it will have to turn to the right to start following the line again.
It will turn to the right until finds the center ground sensor in the middle of the robot, then it changes the global state to FOLLOW_LINE.

VIII. PROBLEMS FACED

A. Blind corners

Take into consideration the obstacle size and the position of the obstacle sensor that points to the left to find wall as shown on the figure 7.

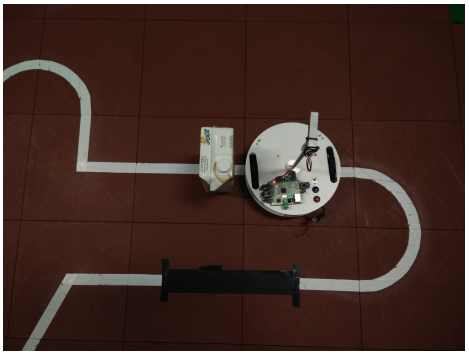

Figure 7: Blind corners when the robot first turns

After the moment that the robot rotates from center, he does not see any wall because the wall has not length enough to see it when turning. So it will be hard to guess the size of the wall, to solve this problem the robot has to detect it first.

To detect this situation, the first measures in the state FOLLOWING_WALL are crucial. Those measures from the sensor that checks for walls on the left will see if there is any walls after rotating. If there is not, it will move forward a small amount, less than usual, because it means that the robot is already in a middle of the process of the corner, so it does not need to move forward that much like he usually does. That is how it is solved.

B. Obstacles over two lines

For this project we decided to take an obstacle over two lines like shown on the image 8.

Figure 8: Obstacle over two lines

The problem with this situation is not related with the fact that the robot has follow the wall and turn when there is a corner, but to find the correct line to follow after circumvent the obstacle.
To solve this problem the robot saves the first position when facing the obstacle for the first time using odometry and after circumvent the obstacle starts comparing with the initial position stored until it matches one of the coordinates, it will mean that he is on the correct line.

IX. CONCLUSION

The robot is considered robust enough to solve all challenges provided in the documentation. It is considered to do even more challenging ones that the ones mentioned, like the two obstacles over two lines.

The velocity that the robots does the tight curves and moves over the scenario is really high, which is good to see that the robot manages to solve the scenario at such speed.

REFERENCES

[1] GS4HS, *Various algorithm optimizations*

[2] Microrato hardware specifications, *http://microrato.ua.pt/*

[3] Society of Robots, *An advanced line following robot with PID control*