Robot Environment Interaction Using TurtleBot

Andreia Rodrigues
Department of Informatics Engineering
Faculty of Engineering of the University of Porto
Porto, Portugal
up201404691@fe.up.pt

Eduardo Leite
Department of Informatics Engineering
Faculty of Engineering of the University of Porto
Porto, Portugal
gei12068@fe.up.pt

Francisco Queirós
Department of Informatics Engineering
Faculty of Engineering of the University of Porto
Porto, Portugal
up201404326@fe.up.pt

Vincent De Clercq
Department of Informatics Engineering
Faculty of Engineering of the University of Porto
Porto, Portugal
up201804291@fe.up.pt

***Abstract*—Humanity is always searching for new ways to enhance technology's capability of making our lives better. It is for that reason that the pursue of knowledge and its availability should be a priority. Nowadays, one of the ways technology is making our daily lives better is through the field of robotics. Robots can help humans perform physical and logical tasks and so it is important to improve their capabilities. In this paper, we will talk about robot autonomy, in particularly, techniques that help the robot navigate around an environment without needing human interaction. As we will show, this can be achieved by combining robotic sensors and actuators with mathematical algorithms developed for this effects.**

I. Introduction

With the ever increasing role of robots in our daily lives and due to their ability to make human life easier, we can see a continuous interest in making these robots as intelligent and autonomous as possible. Machines have evolved from simple task performers such as tighten a screw to robots capable of being able to make informed decisions based on the world that surrounds them. An important factor in measuring how intelligent is a robot is their level of autonomy, by making robots able to function without human interaction we are adding a great value to them and enhancing their potential to help humanity [1].

In most cases, mobility plays an important role on a robot's ability to behave autonomously. For a robot to be truly independent of human action, he must be able to move himself in the environment relying only on the inputs he gets from it. Because the characteristics of the environment cannot always be known in advance and due to the possible noise that can affect a sensor's performance, the task of making the robot able to navigate and interact safely and efficiently with its surroundings is not trivial. For this reason it is crucial to study new ways to improve these features and to promote the literature on the subject. [1], [2].

In this paper the group will explain how we approached some of the most common problems of robot navigation, such as wall following, line following, object following and object avoidance, using a ROS standard platform robot called TurtleBot3. In section II, we will give a brief overview of TurtleBot3 and the ROS platform, followed by a description of how we structured our own C++ implementation. The following sections each explain in more detail each of the modes the robot can operate in, providing a deeper view of the techniques used to overcome that specific challenge, always accompanied by the results. At the end of the paper, we will present some conclusions as well as possible future work.

II. Architecture

A. Introduction

For this project we used TurtleBot3 Waffle, which is a "small, affordable, programmable, ROS-based mobile robot for use in education, research, hobby, and product prototyping". ROS stands for Robot Operating System and it is a framework for building robots which provides many tools and libraries for most common operations [3], [4].

For the programming part we used C++ for its speed and ROS compatibility, along with the OpenCV library which "is an open source computer vision and machine learning software library" [5].

B. Objective

Our goal was to develop a foundation in which we could develop each operating mode and run it independently while maximizing code reuse between features. We also wanted to make sure the safety of the robot itself was prioritized and that we were leveraging all the capabilities of TurtleBot's sensors in an efficient way.

C. Solution

In order to achieve this, after initializing ROS and subscribing to the necessary topics, we used an update loop. In this loop there were two phases:

- Sensor/Callback Phase, in this phase we call every sensor callback by using Spin Once (single-threaded spinning) and we modify the robots internal state. For this particular project we are listening to callbacks from four sensors, the LIDAR for

the wall following and object avoidance behaviours, as well as for our safety routine common to every operating mode, the odometer for the avoid object and follow line modes, the color camera for the follow line and follow object modes and the depth camera which is currently not being used.

- Update/Phase, in this phase we take into consideration the current state of the environment around the robot as perceived by the sensors and the current operating mode (received as an argument at startup) to make the correct calculations that will result in a decision on the best velocity (linear and angular) for the robot to achieve its intended purpose. After calling the specific mode behaviours, we always call a safety function that uses the LIDAR distance to evaluate whether or not the robot is in danger of damage, this routine can override the previous calculations and make the robot stop.

Our solution follows a hierarchical architecture, where the robot maintains an internal state and has a planning phase between sensing and acting as seen in the following image.

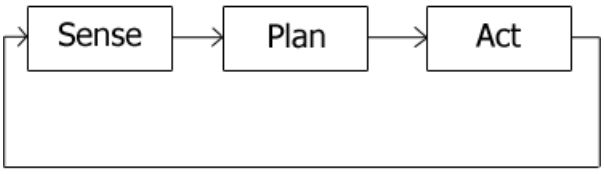

Fig. 1. Hierarchical Architecture (Source: Wikipedia)

III. Wall following mode

A. Introduction

A wall following robot is a robot that tries to maintain a constant distance to a wall, either left or right. This can be useful in a mapping context, where the robot is used to create a detailed map of an unknown area. Following a wall can be easily extrapolated to simply following stationary objects. This is particularly useful in combination with other operational modes, to avoid running into obstacles.

To start following a wall, the robot should first detect it. This will be done by comparing sensor data, namely the distance to the wall as perceived by the LIDAR with a previously fixed value that represents the minimum distance to a wall to start following it.

B. Objective

The objective is to code the robot in such a way that is keeps a constant distance to a wall. This will be done using the LIDAR module of the TurtleBot. Our goal is not only to create software that will follow a wall, but ideally we will also reuse this code to go around obstacles on the line in section IV.

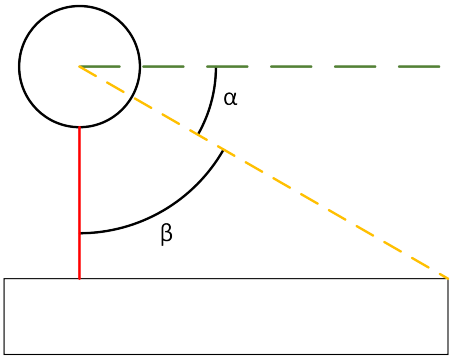

Fig. 2. Problem visualization.

C. Solution

In order to develop a solution to this problem, it is first necessary to evaluate the useful information retrieved from the sensor for the task. In this case, we can retrieve information from the surrounding space using the LIDAR. From this, we took the information about the closest object to the robot. We get distance and angle of the robot in relation to the robot.

One important aspect we considered was the need to choose a side of the robot to follow the wall. This is important because in the event the robot finds an environment where the walls are equally spaced from both sides, the robot could get confused about which side to follow and stay in a loop. To avoid this, when the robot starts, in the first scan of the surroundings, we pick the side closest to the robot as the side to use to follow the wall for the rest of the run.

The wall following problem can be modeled as trying to stay on an imaginary line parallel to the wall the robot is following as shown by figure 2.

To this effect we used an equation present in [1] which says:

$$\omega = -vk(\sin q_3 - (q_2 - T) + K) \tag{1}$$

The meaning of the symbols are expressed in table I.

TABLE I
Expression Symbols

Symbol	Meaning
v	Linear velocity
k	Control parameter
q_3	Angle between the optimal line and the front of the robot
q_2	Distance from wall
T	Target distance from wall
K	Curvature factor

In order to better utilize the representation given by figure 2, we can rename some symbols:

$$(\alpha, d) = (q_3, q_2) \tag{2}$$

This changes the equation to:

$$\omega = -vk(\sin \alpha - (d - T) + K) \tag{3}$$

While the equation mentions a value K, this value serves to characterize the shape of the wall. This is a complex process which requires a lot of processing at run time. Processing time is often the limiting factor in robotics. This added with the lack of need for this process resulted in removing this factor from the equation resulting in a new equation:

$$\omega = -vk(\sin\alpha - (d - T)) \tag{4}$$

This solution only works when the wall is on the left side of the robot due to how the part of the expression that takes into account the distance from the wall and that target distance from the wall, $(d - T)$, will always give a value that its sign does not change depending on the side the robot is following, affecting the resulting angular velocity. To solve this, the right side expression of the equation was changed to $(\sin\beta * ((d - T)/T))$ where β represents the angle between the front of the robot and the laser from the sensor corresponding to the closest object. Besides fixing the aforementioned issue by using the sine of the angle, it is also not sensitive to the absolute values used, due to the use of a ratio. This change results in the following equation:

$$\omega = -vk(\sin\alpha - (\sin\beta * ((d - T)/T))) \tag{5}$$

The angle obtained from the LIDAR sensor corresponds to the β angle, corresponding to the angle between the front of the robot and the laser pointing to the closest object. In order to obtain the α angle, the angle between the front of the robot and the line parallel to the wall, we can use β, using the following equation:

$$\alpha = \begin{cases} \beta - \frac{\pi}{2} & \text{if } \beta >= 0 \\ -\beta - \frac{\pi}{2} & \text{if } \beta < 0 \end{cases} \tag{6}$$

Additionally, to give more control, when the robot is faced with an approaching wall directly in front, the robot slows down to more easily perform turns around corners. When this condition is true, the linear velocity of the robot is reduced.

D. Results

To test the algorithm we ran tests in simulated and real environments. The result was satisfactory. The robot indeed followed walls and even followed along small crevices like doorways. However, we only collected data from the simulated environment. As the data shows, the robot fluctuates relatively close to the target wall distance, getting farther from the wall due to turns around corners and getting closer to the wall when approaching a wall from the front. This indicates the need to be more aggressive in turns and more cautious when detecting a wall in front.

IV. Line following mode

A. Introduction

A robot with line following ability can follow straight or curved lines usually based on their color. Robots of this kind, have a great importance in industrial contexts, where a robot is often required to travel in a specific path to change between workstation or to carry resources from a source to a destination. And so, line following is a crucial part in the challenge of increasing robotic autonomy and value. [6].

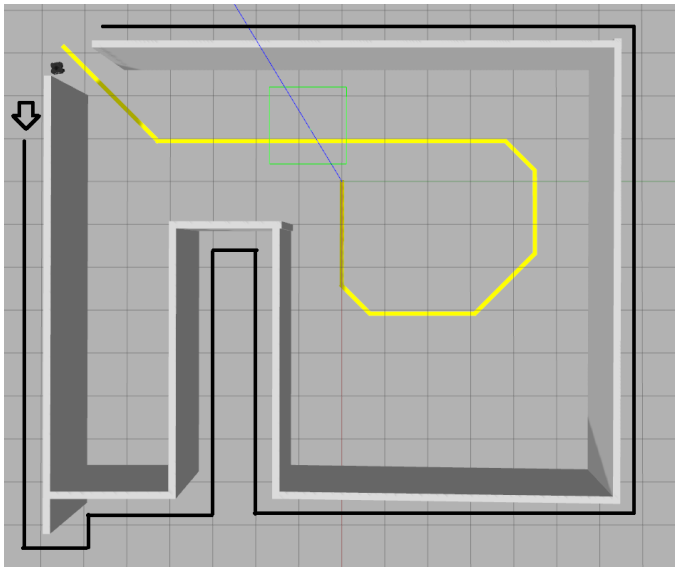

Fig. 3. Test path for a wall following session.

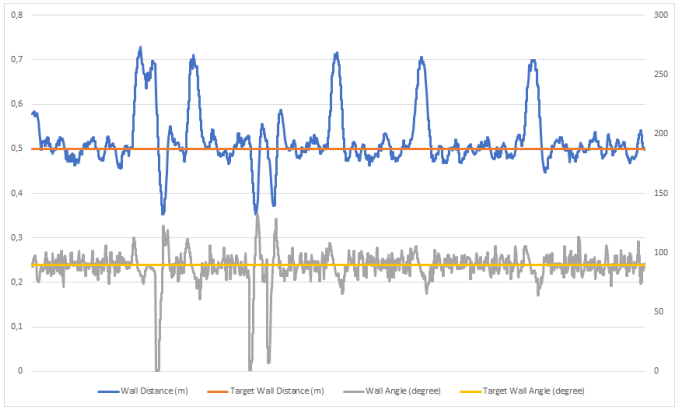

Fig. 4. Results from a wall following session.

In order to detect the presence of the line, the robot must be able to correctly identify its presence and characteristics, and so, it is necessary that the robot can extract and interpret this information from the image it receives from the camera sensor. It is necessary to ignore noise or other lines in the image as well as know the relative position of the line so that the robot can move at the right speed towards the right direction. Something that can further enhance this behaviour, is the ability to detect and go around obstacles wherever possible, this can be done by applying the same techniques described in section III.

B. Objective

Our goal was to program the robot to be able to follow a colored, straight or curved line on the ground without deviating too much from it while being able to go around possible

objects in its path and ignoring noise and other possible lines in the surroundings.

C. Solution

Our solution to this problem starts at the image received from the color camera sensor in the camera callback, and it includes multiple steps:

1) Application of a Gaussian filter, this is used to reduce the noise in the image that comes from the sensor and it works by recalculating the value of each pixel as a result of a function that takes into consideration the value of neighbour pixels multiplied by a coefficient that decreases as the distance to the considered pixel increases, along a Gaussian function (hence, the name of the method).
2) Sample the image down to reduce the search space and, in turn, reduce processing time. This was done using a pyramid representation technique.

Fig. 5. Gaussian Blur (Source: Wikipedia)

3) Convert the image to HSV (hue, saturation, value) color space. This is regarded as a better color space for detecting objects of a certain color.
4) Defining the HSV intervals of the target color. It is important to define an interval and not an exact value due to the nature of camera having noise and also lighting issues in the environment making the object not having an exact same color all the time.
5) Using the HSV image, convert the image to an array of true and false values that represent if the corresponding pixel was inside the intervals specified or not.
6) Mask the array to cover three sections:
   a) Top part of the array
   b) Bottom left part of the array
   c) Bottom right part of the array

   This is done to avoid being confused with parts of the line that are not yet relevant to react to.
7) Calculate the image's pixels' moment. This is done using the `cv::moments` function from the OpenCV library.
8) Using the image's pixels' moment we can calculate the centroid of the pixels to generate a general idea of the direction of the line. The `cv::moments` function provides many values, but we used the `m00` and `m10` values where:

$$m_{ij} = \sum_{x,y} (array(x,y) x^i y^j) \tag{7}$$

   $array$ means the array mentioned before and x and y the x and y positions of each element of the array. Using the expression m_{10}/m_{00} we get the x coordinate of the centroid.
9) If the centroid is sufficiently to the left or right of the picture, then the line is turning left or right, accordingly. Otherwise the line is going forward.

Additionally, there is the task of going around objects in the path of the line. In order to do this, we employed the technique mentioned in section III. Once the robot finds the line again, the robot should follow the line in the same direction as before. However, the robot must know what is the direction it was following. To do this, we utilized the odometer installed in the robot to measure the orientation of the robot when it first finds the object then when it finds the line again, we compare the current orientation with the stored orientation and determine if the robot should give priority to either the left or right side. To execute this priority, when we mask the bottom left and bottom right corners, we apply a bias to apply less or more masking to the prioritized and not prioritized sides respectively. This creates a bias towards the robot assuming the correct orientation.

D. Results

To test the algorithm we ran tests in simulated environment. The robot was able to go from the start of the line to the end of the line without much deviation from the line. The only main issue was that the robot would not align perfectly straight with the line, fluctuating between both sides, but was still able to follow the line successfully. The data shows strong dips in the centroid x coordinate on left turns and spikes on right turns, as one would expect. Again, the fluctuating values are due to the robot not being aligned with the line when going forward. This could indicate the need to have a more continuous evaluation, allowing the robot to align itself better when going forward, rather than a discrete left/right/forward evaluation.

V. Object following mode

A. Introduction

The ability for a robot to be able to follow an object or a person is of great importance to enable cooperation between robots or between a man and a robot. For instance, if a robot can follow an handicapped man and bring his groceries, it has

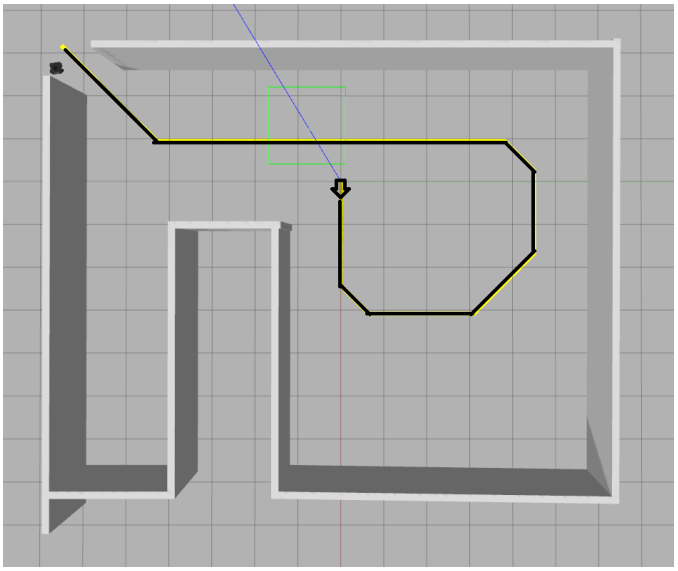

Fig. 6. Test path for a line following session.

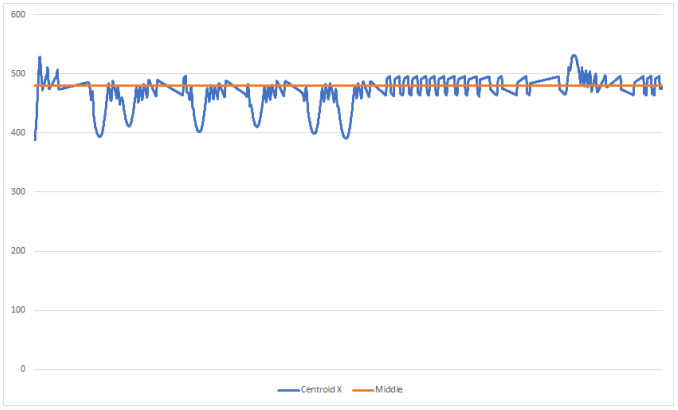

Fig. 7. Results from a line following session.

the potential to greatly improve the his quality of life. In order to do this, it is necessary that the robot can identify the object by using information made available to him by the sensors, such as distance to the object, using for example a LIDAR or a depth camera that and shape recognition of the object using the color camera. [8].

B. Objective

The objective for this mode is to have the robot detect objects of a certain color, determine where they are relative to itself and follow them in space. For this, we'll use the color camera that the robot is equipped with.

C. Solution

To solve this problem we again resorted using the color camera and applied a Gaussian filter, sample the image down, converting the image to the HSV color space, defining the HSV intervals for the target color and converting the image to an array to positive and nil values, depending if it belongs to the interval for each pixel as described by the intervals. For the masking we only resorted to masking a portion of the top of the image, only looking for close objects. We again calculate the moment to determine the centroid of the pixels corresponding to our HSV parameters. This also is used to determine if the robot should conclude if the object is on the left, right or in front.

D. Results

To test the algorithm we ran tests in real environment. Testing in this environment had the issue of the robot detecting objects from the target color even when the target object was not in view. This is most likely due to a combination of noise from the camera and environment having similar enough colors to the target color when testing. However, when showing the target object, the robot followed the object in a satisfactory manner, turning towards the object at a responsive rate. Due to the camera's low field of view, the object must not move too quickly, otherwise the robot will not be able to follow it correctly. Unfortunately, no data was collected to observe.

VI. Object avoidance mode

A. Introduction

A robots ability to avoid objects in real time is inserted into the obstacle avoidance category of autonomous robotic navigation and it is an important requirement for a robot to have practical applications in a vast number of fields [7].

B. Objective

For this mode, the objective is for the robot to detect objects around it up to a certain distance and move away from them. In case a object follow the robot, the robot must try to maintain a perpendicular orientation in relation to the object while moving forward. For this we used the LIDAR sensor of the robot.

C. Solution

To avoid an object, we used the LIDAR sensor to find the closest object within a predefined detection distance. If an object is found, the next objective is to turn away from the object. To do this, the LIDAR is used to get the angle of the front of the robot to the object. This angle will called α. To determine how much the robot must turn, we calculate the following:

$$\beta = \begin{cases} \alpha - 180 & \text{if } \alpha >= 0 \\ \alpha + 180 & \text{if } \alpha < 0 \end{cases} \tag{8}$$

Then we used the odometer to get the orientation of the robot upon detection then we added β to the orientation to get the target orientation and then the robot rotates to the target orientation. After this, the robot checks for two conditions: check if the robot has some object in front of it or if the robot still has an object behind it up to some distance. If the robot finds a new object in front of it, the object to avoid becomes that object, going back to the start of the algorithm. If the robot detects the object still behind it, the object might move

in relation to the robot, it will use the LIDAR to get the angle of the object in relation to its front. From this, the robot will try to correct itself to be facing away from the object. If the angle to the object is α, the angle to the desired orientation β can be determined using the previous equation 8. With this angle, the robot will keep moving forward but with an additional angular velocity equal to:

$$\omega = \sin(\beta) \tag{9}$$

This angular velocity will rotate the robot in the attempt of correcting its orientation in relation to the object.

D. Results

To test the algorithm we ran tests in real environment. We observed the behavior we wanted. The robot would successfully turn away from objects, start moving away and if the object followed the robot and changed orientation in relation to the robot, the robot would move in direction opposite to the object as intended. Unfortunately, no data was collected to observe.

VII. Conclusions and Future Work

The group concluded that although seemingly simple, the task of automating a robot's navigation is quite challenging. The noise from the sensors and the computation time are a major factors that canot be overlooked when aiming for an efficient and smooth result. The overall outcome of our experiments was satisfactory and all major goals were attained. That being said, there is always room for improvement, some parameters could be further optimized with more testing, such as danger distance and optimal wall following distance, some more testing could also help identify edge cases, such as narrow turns, especially in the real world (as opposed to the simulator) and of course a lot more could be done regarding the navigation theme, such as finding an object and then return to a shelter or different behaviour for different line colors.

References

[1] K. Bayer, "Wall Following for Autonomous Navigation," SUNFEST, University of Pennsylvania, Summer 2012.

[2] C. Hsu and C. Juang, "Evolutionary Robot Wall-Following Control Using Type-2 Fuzzy Controller With Species-DE-Activated Continuous ACO," in IEEE Transactions on Fuzzy Systems, vol. 21, no. 1, pp. 100-112, Feb. 2013. doi: 10.1109/TFUZZ.2012.2202665

[3] http://www.ros.org/about-ros/

[4] http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[5] https://opencv.org/about.html

[6] C. Hsu, C. Su, W. Kao and B. Lee, "Vision-Based Line-Following Control of a Two-Wheel Self-Balancing Robot," 2018 International Conference on Machine Learning and Cybernetics (ICMLC), Chengdu, 2018, pp. 319-324. doi: 10.1109/ICMLC.2018.8526952

[7] Ioan Susnea, Viorel Minzu, and Grigore Vasiliu. 2009. Simple, real-time obstacle avoidance algorithm for mobile robots. In Proceedings of the 8th WSEAS International Conference on Computational intelligence, man-machine systems and cybernetics (CIMMACS'09), Cornelia A. Bulucea, Valeri Mladenov, Emil Pop, Monica Leba, and Nikos Mastorakis (Eds.). World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 24-29.

[8] Frink, Elizabeth & Flippo, Daniel & Sharda, Ajay. (2016). Invisible Leash: Object-Following Robot. Journal of Automation, Mobile Robotics & Intelligent Systems. 10. 3-7. 10.14313/JAMRIS_1-2016/1.