Lane-following Robot

Umberto Cocca
Università della Svizzera italiana

David Limacher
Università della Svizzera italiana

***Abstract*—Self-driving cars are a huge field of research combining computer vision and robotics. In that report, a simple approach in a simulation with a robot is presented, which can detect some elements in an environment with a camera and navigate along a street and follow some defined rules like navigating based on arrow on the ground. The situation is described in the second chapter after the introduction followed by a chapter describing how those problems have been solved. In the third chapter, a short overview of extensions, that could be implemented is given to get to a more advanced implementation, which would be closer to a real-world scenario.**

I. Introduction

Self-driving cars are a phenomenon that more and more gets into the use of the wide public. Their functions go from the lane-keeping assist, which already is in wide use to fully autonomous cars, which need no interaction of a person. There are already cars that can do that, but mostly only in test scenarios. This is not only due to technical problems but also due to legal limitations. Still, the path is clear and cars, which are basically robots, will get better and they most likely will be used widely in the future. In this project, we try to tackle some problems that have to be solved to get an autonomous robot.

II. Problem

Our problem definition is to have a self-driving robot, which can drive on a street and follow some rules and also change directions based on indications.

A. Robot

For our case, we use the Thymio robot in a Gazebo simulation. This robot has different sensors and actuators to drive. The input mostly used in our case is the front camera, which is installed on the robot and points forward with a certain angle pointing a bit down to be able to see better the ground right in front of the robot. The angle of the camera had to be changed several times during the implementation and was one of the factors where a trade-off had to be considered. If the camera is pointing down too much it can not see a lot in front but with more pointing straight ahead, it can not see what is close. A value in between therefore has been chosen. In addition, it also has different proximity sensors, which are installed around the robot at different angles pointing parallel to the ground. There are five sensors in the front of the robot, where one is centred and the others are sensing in defined angles to the left and right of the centre. Also in the back are two proximity sensors. The robot has two steered wheels, which are actuated both separately and take commands for the linear velocity as well as the angular velocity.

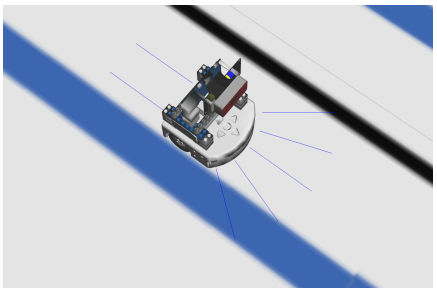

Fig. 1. Robot used for project

B. Map

The next part is the map that will be used in the simulation. For this, a map has been constructed with different elements, that need different handlings. During the project, new elements incrementally have been added to increase the difficulty. This map was generated as an image (in PowerPoint), which can be seen in Figure 7. It then was used as the texture of the ground inside the Gazebo Simulator. The whole ground has a total size of 6 by 6 meters and is completely flat.

C. Lanes

The first element is the street itself with the border of the lane. The outer border is with a blue line and the divider between the left and right lane is drawn with a black line. The goal is, that similar to a real street the robot always drives on the right lane and therefore should always have the black line to the left. A street is also defined by different elements, where the car has to steer along. For that, the most basic section is just a straight road. In addition to that, there are curves, where the robot has to smoothly drive along and hard curves or edges, where the robot has to change direction quite fast. There are also crossroads, where two streets come together and the robot can decide in which direction it goes but still has to get on the right lane of the street again.

D. Markings

As in a regular car situation, there are also markings that indicate something to the robot. The first element is the speed marking, which shows to the robot if it has to go faster or slower. A red line over the street means slower and a green line means faster. This is a simplification of the street signs, where a specific number indicates the maximum speed. Because the

robot does not use any system of navigation the information in which direction to go on the crossroad is indicated by an arrow pointing either left, right or straight ahead. This arrow is in orange colour and has a thin tail.

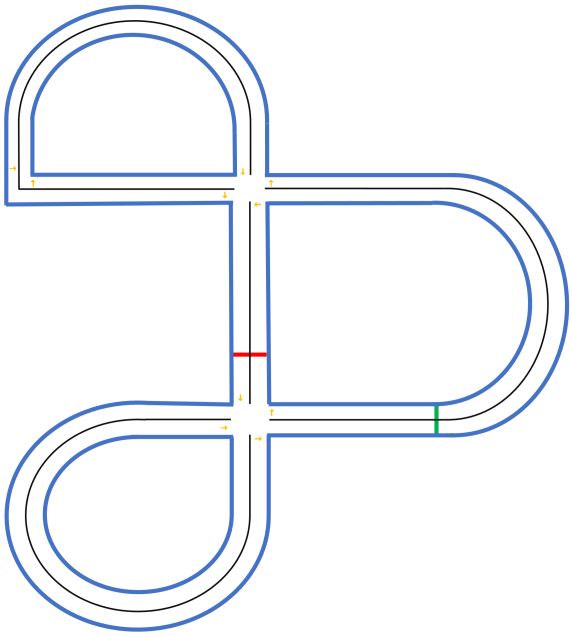

Fig. 2. Street map used for project

III. Solution

A. Lane detection

For the task of detecting lanes on the road the image captured with the camera placed above the Thymio is used as the input for the algorithm. To work with images the computer vision framework OpenCV is used, the so-called CvBridge module has been added, which encodes each frame in BGR colour encoding, the default in OpenCV. Each frame has been cropped in the input to make the processing much faster. Afterwards, morphology was applied to the image to clean the area through erosion. The image segmentation of the lines is done starting with converting each frame into the HSV colour space. After identifying the black and the blue colours in RGB colour encoding, those were converted into the HSV space and a threshold range was created to find them in the image. With this, it is possible to mask the original cropped input and get a black and white image segmenting those colours.

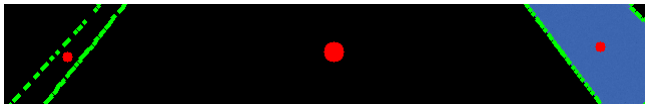

Fig. 3. Final processing image frame from the input. The green points have the purpose of identify the contours of the black and blue lane. The small red points are the centroids of each blob, and the bigger red point in the middle gives an almost precise idea of the robot position between the two lanes in the space.

From that, the connected components are calculated and for each blob, the centroid is detected. This allows to have a clear representation of the position of the lines and based on the centroids of each line, the robot now can be oriented to be in the centre by changing the angular velocity. When the robot reaches a curve it can happen that it is not able to see the right or the left line during the movement. For this reason, the centroid is calculated together with the highest pixel in y coordinates of the blob it is possible to get the orientation of the line, hence don't get out of the street.

Fig. 4. Image frame only the black line is seen. The yellow point has the highest y coordinate of the blob, which is used to navigate also with only one lane.

B. Speed marking detection

The detection of the speed marking on the map is quite trivial. If the robot sees many red or green pixels in front of it increases or decreases a variable that is used as a multiplication factor for the linear velocity. To only increase once, a timer is used that blocks the change (until it most likely crossed the line) for a specific time before it will do this update again.

C. Direction arrow detection

When the robot comes close to a crossroad it should be able to detect the arrow with the camera and see whether it is pointing left, right or straight from the perspective of the robot. For this, it first needs to detect if there is an arrow close enough in view or not. This is done by filtering the lower middle part of the image and apply to the HSV encoded image a mask, where every pixel that is close enough to the orange used for the arrows is put to 1. If there is a sufficient number of orange pixels it finds the biggest contour and calculates the centre of mass. Also, the middle point is calculated, which is achieved by looking at where the border of the orange pixels are and take the centre of the rectangle enclosing them. Now that there are two different midpoints their relation can be considered. Because an arrow has the general form of a thin tail and a larger triangle head at the front the centre of mass is generally more into the direction the arrow is pointing compared to the midpoint calculated with the rectangle. Therefore the angle between those points can be seen as the angle the arrow is pointing and by checking if this angle is sufficiently close to one of the three expected directions it can decide on one.

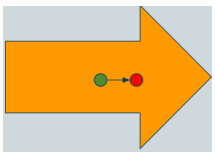

Fig. 5. Detection of arrow with red being the center of mass and green the mid point of the rectangle

Depending on how good the arrow is positioned this could also give some wrong results in some cases. Especially if the robot is not facing the arrow straight this could fail. This could be the case if a crossroad is right after a curve. Therefore voting is taken and an arrow direction needs at least three votes to be considered and the maximum vote is taken and saved in a variable that is considered by the navigation. To reset the information of the direction before the robot gets to the next one all votes are cleared if it does not see some amount of orange pixels for some frames. In general, this works very fine, but due to the limitation of the number of frames that are considered if the robot is too fast it is not able to detect the arrow in the cropped frame that is considered. If that happens the robot will just go straight.

D. Driving

Furthermore, If the robot is not positioned fine and looks at the other lane it needs to be corrected. Hence the result of the image processing is given by two connected components where the left centroid belongs to the blue line and the right centroid belongs to the black line, then the robot receives a slight boost, that sets the angular velocity to -1, trying so to return on its way. This condition is always true because if Thymio drives forward or in the opposite direction it will always have the black line on the left and blue line on the right. A problem is if the robot is totally placed on the other street then currently it follows the other lane.

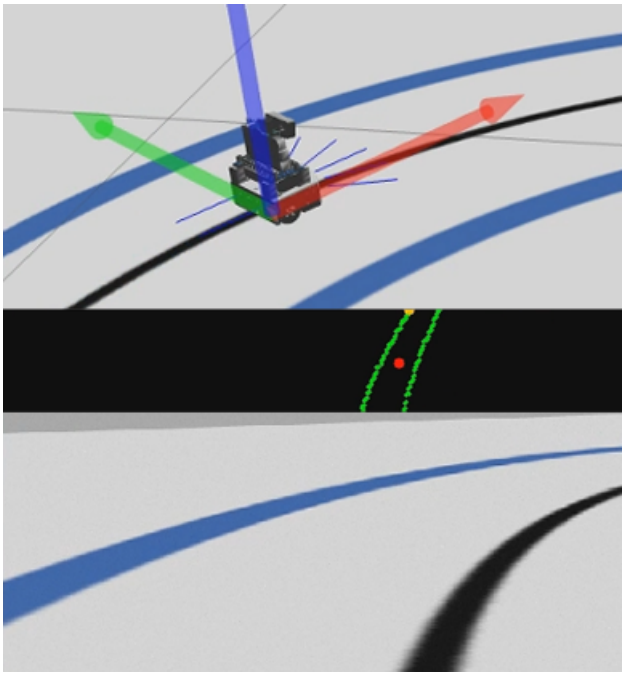

Fig. 6. At the top there is the Thymio that is a bit off the road. Below it is the frame received in input from the camera and in the middle is the image processing of the scene.

Another boost is applied during the movement at the curve. Assuming that the robot is on the correct lane if the amount of pixel that describes the black line is to the right of $width/2$ then to avoid losing lines detection on the road it gets a boost for the angular velocity. Likewise, if the amount of pixel that describes the right blue line is to the left of $width/2$ then another boost is given to the angular velocity, where in both cases it is a factor of 2.

In addition to the general driving, there is also a collision detection so multiple robots can drive with different speeds. If one robot is behind a slower one and the one in front is detected with the proximity sensor the robot stops until the street is free again.

It is important to notice that the robot is not able to drive starting from any position on the map, in fact, if it cannot detect any street there is no command and it simply continues with the previous action, which would be none in case of a start.

E. Crossroad

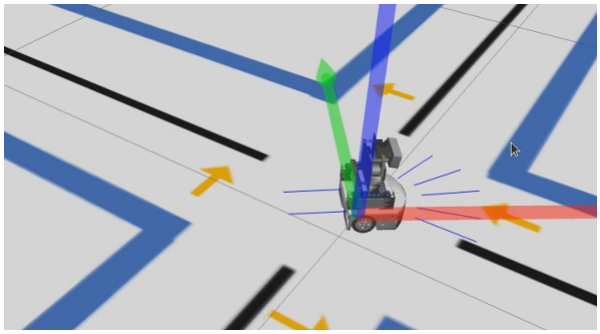

Fig. 7. A road that crosses another road

On the map are two crossroads. As soon as the robot comes close to the crossroad and detects an arrow, it is preparing to turn left, right or go straight depending on the detection. It is not a trivial task because the robot can reach the crossing at different speeds could get off the road if it does a fixed sequence of actions. The radius of the circular motion depends on the current speed of the robot and the distance from the robot to the arrow. The position of the arrow in the 3D space is not known, differently the 2D coordinate of the arrow blob centroid of the frame received in input is known. But this is not always at the same position and it changes according to the speed of the robot and the time the arrow detection decides. Therefore, to solve this problem instead of trying to project the 2D pixel coordinate of the arrow centroid in the 3D space, the radius is predicted through linear regression. The data for training was collected saving the successful attempts. The dataset consists of the values of speed, the coordinate pixel of the arrow centroid and the radius, where the radius value has been assigned manually, by analyzing directly the scene. Thus first the radius value was set that executed the task and finally if the movement was done correctly the data was saved. After applying the linear regression on the data set

the coefficients have been used to calculate the radius with the following formula:

$$R_{right} = 0.637 - 0.093 \cdot speed - 4.049e - 05 \cdot pixel_y$$

When the Thymio points to left, since the left corner is approximately 1.5 times further away compared to the right corner, then the radius was computed in the following way: $R_{left} = 1.5 \cdot R_{right}$.

Problems could occur if the direction arrow is not accurately detected, due to bad positioning of the robot. According to the tests it is possible to state the system is quite robust and rarely misses the mark.

IV. Extensions

Obviously there would be the need to add many extensions to get something close to a self-driving car. Still some are mentioned here to put the project into a context and describe some approaches.

A. Navigation and path finding

So far the robot only navigates based on the arrows indicating the direction it has to go. There could be more advanced concepts mostly based on the global information. It could also have a local information discovering the map and finding a way to a goal. In case of global information the map has to be known, which could be either as the image that is given as the ground with some preprocessing or with an additional graph similar to a classical navigation. With such a graph the shortest distance could be easily calculated with for example A* search algorithm. Still the robot needs to know its location. This could be achieved by just observing the environment and by seeing for example crossings or curves could figure out the current position. It is easier to evaluate the best way if the position is known in advance, which can be done with some external information as it is for regular cars given with a GPS. In Gazebo the Odometry-Information could be subscribed and used to know where in the map the robot is and from that apply the path finding.

B. Kidnapped robot

In our implementation the robot is assumed to start somewhere located on the street and therefore can see the lane and start driving. In case the robot is positioned randomly there would be the need to first find a street. This could be achieved by looking around and see where the closest pixel is that has the color of a street and drive into that direction. As soon as it is close it can figure out the direction of the lane and position itself on the street. From that it can simply navigate the way it was implemented.

C. Priority at crossroad

Not all rules that apply on a regular street have been implemented in our scenario. First of all there are different types of streets normally, which would determine which robot has priority at a crossroad and some alternative determination like the right of way (car at crossroad to the right has priority). So what would need to be done is at the crossroad a detection if there is any other robot also at the crossroad and to which direction it is. If it is on the right or straight ahead and the robot wants to turn left the robot would need to wait. Another approach would be to have stop signs, where one road has priority by this definition.

D. Heightmap

Our environment is completely flat, which is often not the case in reality, because we would expect hills and other height differences. For this a heightmap would have to be added. If the height differences are just small our implementation would most likely still work, but if there is a larger raise or fall the angel of the camera to the objects that have to be detected and handeled differently.

E. More realistic elements

We build our map so the detection of all the elements is easy to implement from a computer vision perspective. They all have distinguishable colors and no additional features especially 3D elements are present. The lanes in a real scenario are way harder to detect and also the signs are not that clear and would need more advanced computer vision and pattern recognition. Also a very important part of self-driving cars is safety in terms of other cars but also bicycles and pedestrians, which are moving objects that can be hard to detect and the handling of such a case is not trivial, because different options would result in different amounts of damage. To achieve this a self-driving car has way more sensors and normally have a view around the whole car and are able to detect the environment in 3D, which is not possible with the Thymio robot used.

V. Conclusion

During the project an iterative way was taken to add more complexity one after the other. A solution has been implemented, where a robot can in most cases follow the correct lane, follow the given rules on the map and navigate by looking at the arrows on the map. Also the speed can be changed by markings on the map, which added some complexity and made the driving harder, because the angles and times especially at the crossroads had to be handled differently and for this also some machine learning has been used to handle that situation. Most orientation is only done with the one camera that is pointing forward on the robot and has some limited vision, because it is not able to see everything necessary with some angles and the robot can lose its orientation in some cases.