Aarhus University MSc Course Project Control of Mobile Robots (AY 2019-20)

S. L. Skovgaard
dept. of Engineering (of Aff.)
Aarhus Unitversity (of Aff.)
Aarhus, Denmark
201401682@post.au.dk

Abstract—The article will document how to develop control software for ground robots and simulate the controller using the popular Gazebo software.

I. Introduction

The main focus of this paper will be to explain and document the control software used to control a Husky robot. The software makes use of the robot operation system ROS. The paper will explain theory, code, simulation and results with accompanying figures and plots. The main objective of this project is to get a ground robot to autonomously navigate through a maze.

II. Theory

A. The Robot Operating System

In order to develop software for robots in a structured and easy way, the robot operating system(ROS) [1] has been used for this project. ROS is based upon the subscriber-publisher pattern where nodes is able to publish data as topics and subscribe to specific topics. In this project a node has been developed that is both a subscriber and publisher which enables the node to: gather data, make calculations using this data and the publish new data.

B. Control Theory

In order for the robot to be able navigate the maze a closed-loop control algorithm has to be developed. For a ground robot the linear velocity is defined by the variable v and angular velocity is defined by the variable ω. These variables will be mentioned later in the theory section. Three different variables is needed to get a stable control algorithm: ρ, $\alpha$and β. These can be derived in a geometric manor by looking at figure 1 [2]. The following equations can be then derived [2]:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \tag{1}$$

$$\alpha = -\Theta + \text{atan2}(\Delta y, \Delta x) \tag{2}$$

$$\beta = -\Theta - \alpha \tag{3}$$

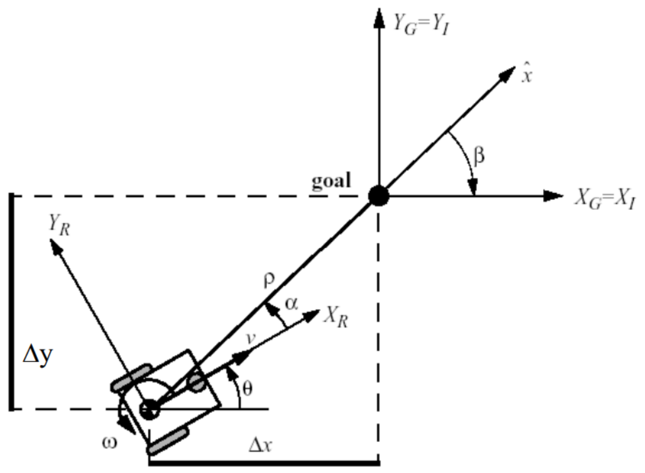

Fig. 1. Overview of ground robot and goal

It can furthermore be shown that with the following equations the control algorithm will drive the robot to the desired location.

$$v = k_\rho \rho \tag{4}$$

$$\omega = k_\alpha + k_\beta \beta \tag{5}$$

By utilizing these equations we can develop a stable closed-loop control algorithm for the ground robot if the controller coefficients also adhere to the following constraints [2]: $k_\rho > 0$, $k_\beta < 0$ and $k_\alpha - k\rho > 0$

C. Gazebo

Gazebo is simulation environment that works together with ROS and is therefore suitable as a simulation tool for this project. The ground robot used for the simulation is Clearpath's HUSKY robot. Luckily Clearpath provides the HUSKY robot as a downloadable package for Gazebo and this package is used for this project. Furthermore the maze the robot i supposed to navigate has been provided. This maze is in the form of a world file.

III. Python code

The code used for this project is built using the rospy package which is a Python library. Roscpp is also available if C++ is preferred.

As mentioned earlier in this article, ROS makes use of the publisher/subscribre pattern. The Python program made makes use of a single node to act as both a publisher and subscriber. This way it's possible to subscribe to location data (x and y coordinates) from either the HUSKY robot or Gazebo, process the data and push new data to the robot. The most accurate coordinates will always be the ones from gazebo but since we want to see how the odometry data from the robots sensors and Gazebo's data differ, odometry is used here. Another benefit of using odometry is that it is possible to implement this to a real world robot. Gazebo's coordinate data is only available when simulating.

Whenever a topic is released with the /odometry/filtered ID, the coordinates are extracted and used for the calculations discussed in II. To get the Theta angle quaternion coordinates are also extracted and a quaternion_to_euler function is used to get yaw (Theta) of the HUSKY robot.

When the calculations are done new velocity and angular velocity is published to the HUSKY robot. The program will then loop and do the calculations again.

To get the robot to properly navigate through the maze, the four corner coordinates are needed, see figure 2. These have been found by manually driving the robot to the green dots and read the coordinates from /gazebo/model_state topic.

When these corner points have been obtained, they have been coded into a list. This list will contain four coordinates in the form [[a1,a2], [b1,b2], [c1,c2], [d1,d2]]. By having this structure it is possible to iterate through the four coordinates. The switching between coordinates happens whenever the distance to the current target is below 0.5.

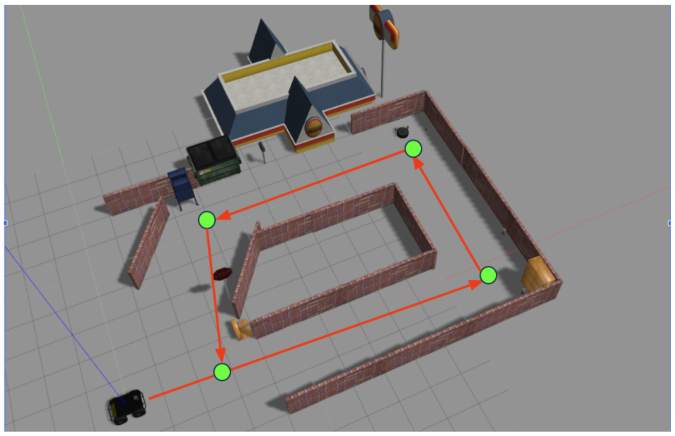

Fig. 2. Overview of the maze

IV. Results and discussion

To confirm that the control software works as intended and the robot isn't crashing into the walls or the burger joint the robot has been observed through the program rviz. This program makes it possible to visualize what the different sensors on the robot sees. On figure 3 the odometry data can be seen in rviz. The red line is the trajectory of the robot according to the odometry data. It can be seen that the robot goes to the four corner coordinates. When going from the third to the fourth coordinate the robot makes weird turn to the right, but quickly doesn't crash into the walls and eventually goes to the last point. The reason for this behaviour is uncertain, but since it successfully completes the maze, it haven't been explore more.

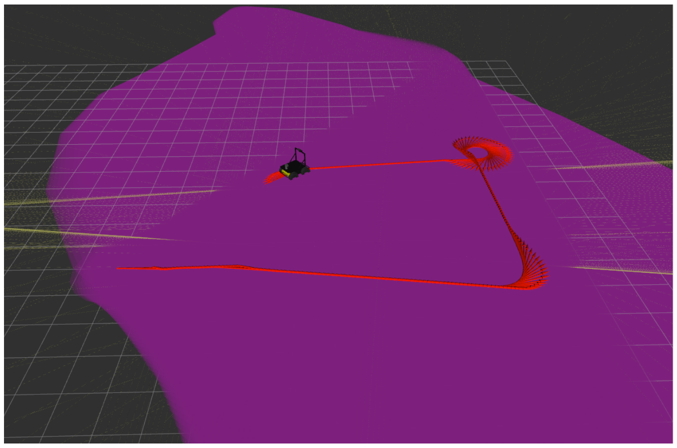

Fig. 3. Odemetry visualized through rviz

Another way of getting data from the robot is through its inertial measurement unit or IMU. By using the program rqt it is possible to subscribe to topics from the IMU and visualize them in a plot. These plots will give a good indication of how the robot behaves. These plots can be seen on figure 4 and 5.

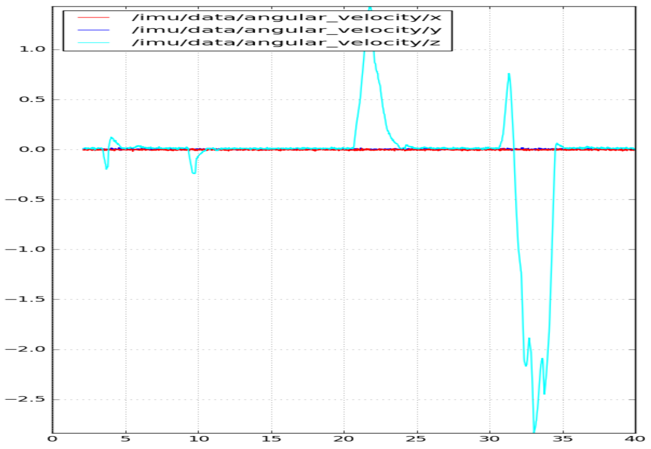

Fig. 4. Plot obtained through rqt showing angular velocity

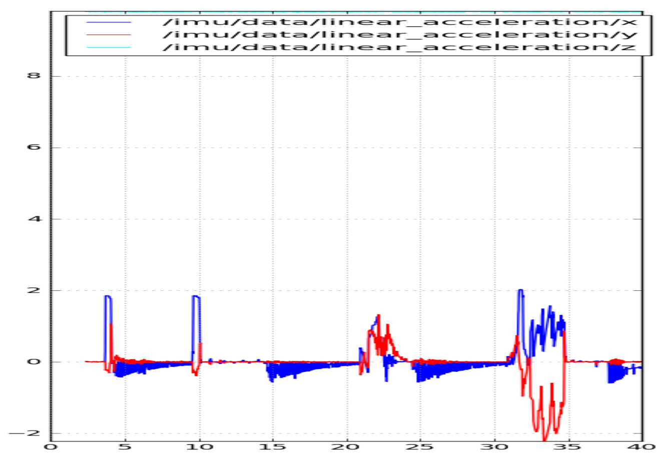

Fig. 5. Plot obtained through rqt showing linear velocity

The angular velocity tells us when the robot turns and if it turns right or left. At t=32 we see the full circle turn with the angular velocity making a significant dip. On figure 5 we see how the robot accelerates whenever it reaches a goal(one of the four coordinates).

In order to see how much the odometry from the robot differs from the real values several plots have been made. One plot that shows the robots estimated travelled distance and the true distance. The true distance is obtained from /gazebo/model_state. The distance is found by calculating the euclidean distance from the last position coordinates to the new position coordinates.

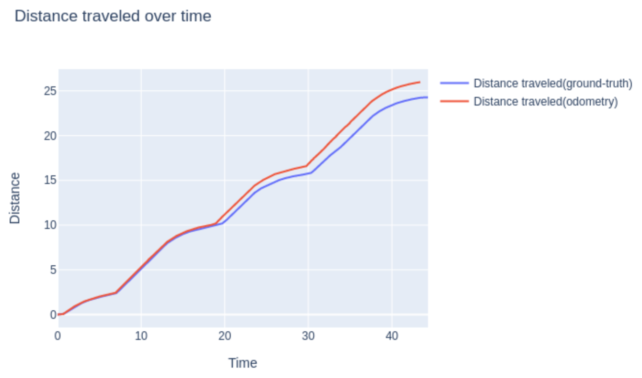

Fig. 6. Plot of distance over time

On figure 6 the two lines follows each other fairly well however the odometry obtained distance diverge over time. This is caused by multiple factors. The main factor might be that there is some wheelspin or that the sensors used simply isn't very accurate.

When plotting the position of the robot the divergence is even more pronounced. On figure 7 it can be seen that the robots odometry data diverge greatly when turning left. This is most likely because of wheelspin which the odometry data can't take into account. If the robot was to loop through the maze again it would certainly crash into the walls because the error between odometry and the correct data woud be too big.

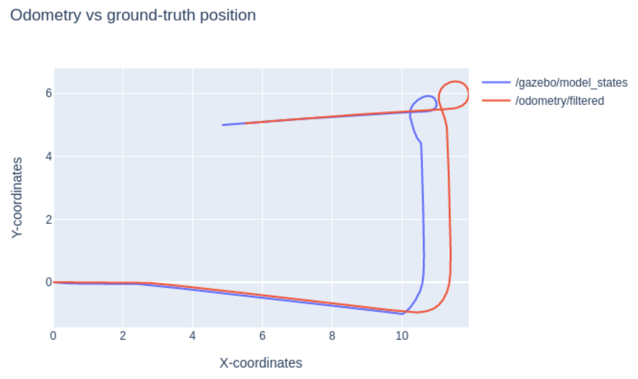

Fig. 7. Plot of coordinates from start to finish

When plotting the yaw or the rotation around Z-axis of the robot with respect to the world frame, it can be seen that the odemetry and ground-truth data is not diverging over time. This might be because the sensor used to calculate yaw is not dependent on the wheels rotation and is therefore more accurate. This can be seen on figure 8.

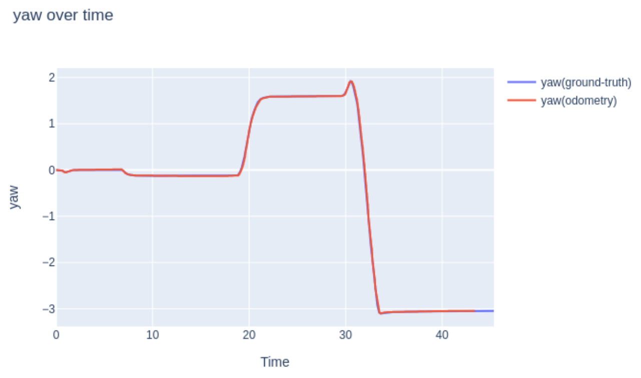

Fig. 8. Plot of yaw over time

V. Conclusion

The main objective for this project has been to develop control software for a ground robot so that it is possible for it to navigate through a maze without crashing into the walls. There has also been a big emphasis on dependability of odemetric data versus ground-truth data. The plots shown during the Results and discussion section have clearly shown how odometry data becomes less accurate over time. Furthermore different tools for analysing the robots sensor data have been used such as rqt and rviz.

Task 5 haven't been completed because of time constraints and would have high priority for future work. All in all it can said that this project has been completed to a acceptable level.

References

[1] http://wiki.ros.org/, date: 3/10/2020

[2] Erdal Kaycan, "Control Of Mobile Robots, week 3: Ground Robot models", 16th of September 2020