Lane Following with Real-Time Traffic Signal Detection and Obstacle Avoidance

Introduction

As global delivery networks shift towards automation, autonomous package delivery systems are becoming a priority for industrial leaders. Although this might seem convenient as autonomous vehicles deliver your package at your doorstep without any human intervention, implementing it requires consideration of multiple factors such as lane following, real-time traffic signal detection and obstacle avoidance. With the online deliveries soaring every month, it is of paramount importance that autonomous vehicles are incorporated into this industry to keep up with the rising demands. Therefore, in this project, we aim to address the problems of real time traffic signal detection and obstacle avoidance that frequently occurs during lane following and is a crucial part of the autonomous package delivery system.

Related Work

There have been significant advances made in this field. Maji et al. (2024) implemented a ground based package delivery robot by integrating sensing, mechanical design and autonomous navigation to support reliable last-mile delivery. This paper sheds light on the use of ground based cars that can be used to deliver packages independently and how it can be implemented in the real world. Expanding upon this, Yin and Aditjandra (2020) analyze the autonomous delivery vehicles in urban scenarios, emphasizing heavily on their ability to reduce energy consumptions and emissions while improving logistics performance. Together, these papers analyze the technical practicality and societal value of autonomous package delivery systems.

Approach

In this project, we implemented a simple lane following robot which follows traffic signals in real-time while trying to avoid obstacles at the same time. A TurtleBot has been chosen which represents an autonomous delivery robot. The robot is controlled by a ROS2 package with four ROS2 nodes each of which play a major role in lane following, sign detection, obstacle detection and decision making.

A pseudo-code version of the decision maker node would look a bit like this :

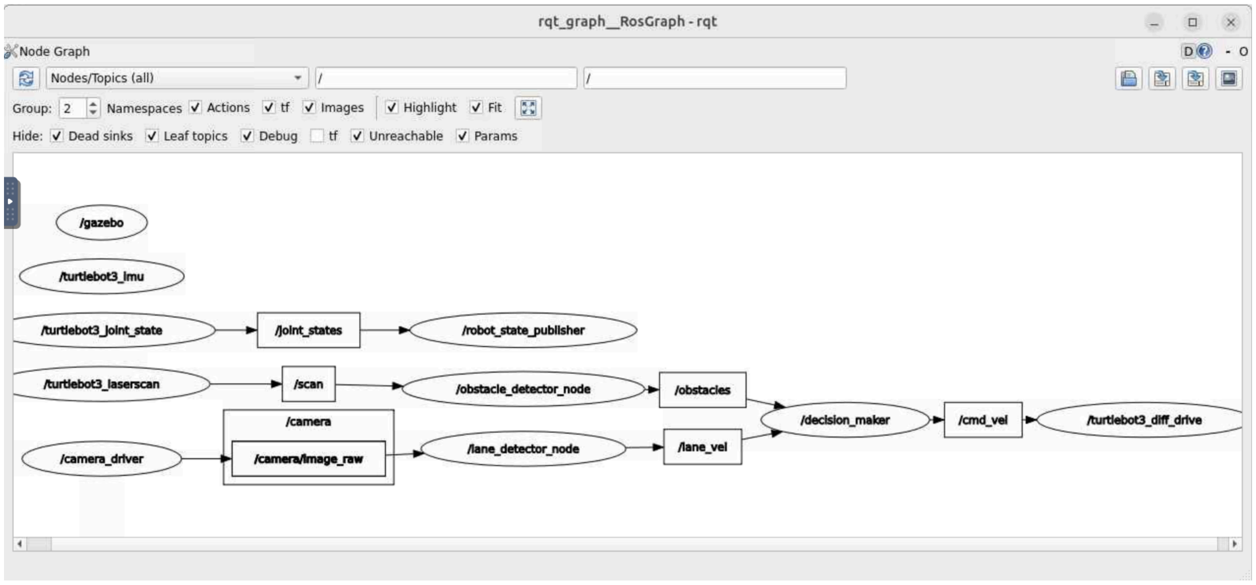
```
While robot_active:
    If obstacle_distance < threshold:
        State = avoiding_obstacle
        If obstacle_side == left:
            Steer_right
        Elif obstacle_side == right:
            Steer left
        Else:
            Stop
    Elif “stop” detected:
        Stop for sign
        Wait(for duration)
        Add to memory
    Elif “turn” detected:
        State = turning
        turn(for duration)
    Elif lane detected:
        State = lane following
    Else:
        State = searching.
```

This is a very barebones version of the decision maker node but this gives a very good idea as to what sort of format we will have. We also utilized a state machine which allowed command over what the robot does to be very simple and efficient.

Use of ROS

The autorace 2020 world has been used for this study. We developed a ROS2 package named autonomous_driving which is composed of four nodes:

1. lane_detector_node:
   - Subscribes: /camera/image_raw
   - Publishes: /lane_vel (Twist)
   - Function: Uses a PID controller to stay in the center of the desired lane
2. sign_detector_node:
   - Subscribes: /camera/image_raw
   - Publishes: /signs/detected (String)
   - Function: Runs YOLO inference and publishes sign labels
3. obstacle_detector_node:
   - Subscribes: /scan
   - Publishes: /obstacles (MarkerArray)
   - Function: Publish the centroids of the nearby objects
4. decision_maker:
   - Subscribes: /lane_vel, /signs/detected, /obstacles
   - Publishes: /cmd_vel (Twist)
   - Function: Uses results from the other nodes to make an informed decision



Nodes/Topics (all) | / | /

Group: 2 Namespaces ✓ Actions ✓ tf ✓ Images ✓ Highlight ✓ Fit

Hide: ✓ Dead sinks ✓ Leaf topics ✓ Debug ☐ tf ✓ Unreachable ✓ Params

ROS 2 System Architecture (sign_detector_node not shown as it needs to be in an isolated terminal due to it needing a YOLO environment)

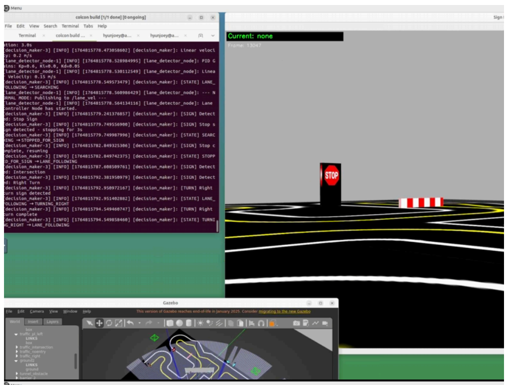
Results

For the quantitative metrics of this project we came up with Lane following accuracy, obstacle detection, sign detection accuracy, and successful lap completion. Going in order respectively the robot did in fact follow the lanes and did not go off track. If the robot were to go off track it would start searching again for the lanes and find its way back. Next, the robot did turn just enough for the robot to avoid the obstacle and for the lane following to keep following the lane allowing the robot to avoid the obstacle. Sign detection accuracy was tracked with a separate screen that is from the sign detection node that allows the user to see what the robot sees and thinks. It creates borders around the signs with what it thinks it is. As long as the sign is not cut off and is not too far away where it does not register it will 100% correctly identify the signs. Finally, for the successful lap completion, the robot did complete a whole lap around the track and hit the speed bump as it was below the sensor for the object detection. We thought this would be a good stopping point as the robot could not actually fit in the tunnel unless it was precise which was incredibly difficult to do.

Some of the challenges we faced with this project was python environment conflicts. This problem arose because the YOLO environment could not run the other nodes which were started in Humble. This problem was a bit tricky at first because we wanted everything to be in a launch file however, we found out the best solution was to run two different terminals. One for the other nodes and one for the sign detection. Another challenge was getting the robot to turn just the right amount for the right and left turn signs as well as for the obstacles. If the robot were to turn too much it would eventually lose the track and if it did not turn enough it would either hit the obstacle or get stuck in a sort of loop until the robot completely lost control. Fine tuning the values for the turning did take a lot of time but was eventually solved. Another challenge was the robot and the sign validation. The robot would either a, see a sign from across the track and register that as a command or b, get stuck in an infinite loop of stop sign stops. If problem a were to occur it would cause the robot to do moves that did not make sense. This was solved by adding distance detection that would allow the robot to only detect signs that were directly in front of it. If problem b were to occur it would just keep stopping and would not move at all. This was solved by adding a checker that would see if the robot had seen a stop sign within the last few seconds and if it did it would ignore that stop sign.

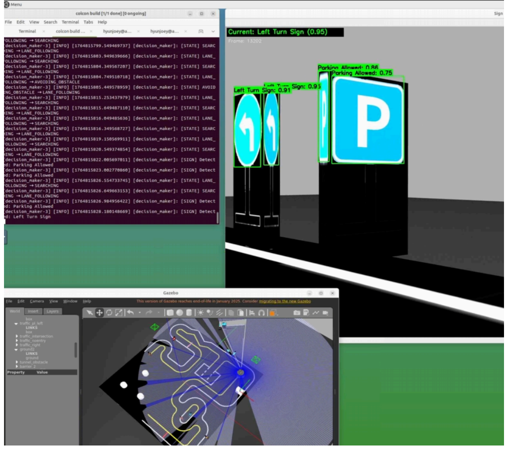
Some of the limitations of our approach is the time based turns would not be precise for real-world scenarios. If a real obstacle were to appear in front of a robot it might get by with a time based turn but eventually it would just hit the obstacle. The only reason why this worked for the TurtleBot was because we knew the obstacle ahead of time. Another limitation would be that this is a very basic version of a robot that could actually move around in the real world. The sign turns would not transition well as the turns could be different curvatures. Obstacles would also cause an issue as they all vary in size.

Some future improvements would be to figure out a way for everything to work in the launch file and get rid of the separate terminals as this would make it a bit easier to run. Also adding adaptive speed control based on the curvature of the road or depending on the obstacle that is in front of the robot. This would allow the robot to function much better in the real world. Another improvement would be to add more signs and add a sort of traffic light support so the robot could read more signs and also react to the traffic lights. This would also be another way to make the robot more ready.



```
tion: 3.0s
[decision_maker-3] [INFO] [1764815778.473058602] [decision_maker]: Linear veloci
ty: 0.2 m/s
[lane_detector_node-1] [INFO] [1764815778.528984995] [lane_detector_node]: PID G
ains: Kp=0.6, Ki=0.0, Kd=0.05
[lane_detector_node-1] [INFO] [1764815778.530112549] [lane_detector_node]: Linea
r Velocity: 0.15 m/s
[decision_maker-3] [INFO] [1764815778.549573479] [decision_maker]: [STATE] LANE_
FOLLOWING → SEARCHING
[lane_detector_node-1] [INFO] [1764815778.560986429] [lane_detector_node]: --- N
ORMAL MODE: Publishing to /lane_vel ---
[lane_detector_node-1] [INFO] [1764815778.564134116] [lane_detector_node]: Lane
Controller Node has started.
[decision_maker-3] [INFO] [1764815779.241376857] [decision_maker]: [SIGN] Detect
ed: Stop Sign
[decision_maker-3] [INFO] [1764815779.749556900] [decision_maker]: [SIGN] Stop s
ign detected - stopping for 3s
[decision_maker-3] [INFO] [1764815779.749987996] [decision_maker]: [STATE] SEARC
HING → STOPPED_FOR_SIGN
[decision_maker-3] [INFO] [1764815782.849325306] [decision_maker]: [SIGN] Stop c
omplete, resuming
[decision_maker-3] [INFO] [1764815782.849742375] [decision_maker]: [STATE] STOPP
ED_FOR_SIGN → LANE_FOLLOWING
[decision_maker-3] [INFO] [1764815787.608509761] [decision_maker]: [SIGN] Detect
ed: Intersection
[decision_maker-3] [INFO] [1764815792.381950979] [decision_maker]: [SIGN] Detect
ed: Right Turn
[decision_maker-3] [INFO] [1764815792.950972167] [decision_maker]: [TURN] Right
turn sign detected
[decision_maker-3] [INFO] [1764815792.951402882] [decision_maker]: [STATE] LANE_
FOLLOWING → TURNING_RIGHT
[decision_maker-3] [INFO] [1764815794.549460747] [decision_maker]: [TURN] Right
turn complete
[decision_maker-3] [INFO] [1764815794.549858460] [decision_maker]: [STATE] TURNI
NG_RIGHT → LANE_FOLLOWING
```

Here is an example of the sign detector not reacting to a sign that is too far away. Instead, it just ignores it and continues following the track.


colcon build [1/1 done] [0 ongoing]
Current: Left Turn Sign (0.95)
Left Turn Sign: 0.91
Parking Allowed: 0.75
Gazebo

```
FOLLOWING → SEARCHING
[decision_maker-3] [INFO] [1764815799.549469737] [decision_maker]: [STATE] SEARC
HING → LANE_FOLLOWING
[decision_maker-3] [INFO] [1764815803.949639666] [decision_maker]: [STATE] LANE_
FOLLOWING → SEARCHING
[decision_maker-3] [INFO] [1764815804.349567207] [decision_maker]: [STATE] SEARC
HING → LANE_FOLLOWING
[decision_maker-3] [INFO] [1764815804.749510718] [decision_maker]: [STATE] LANE_
FOLLOWING → AVOIDING_OBSTACLE
[decision_maker-3] [INFO] [1764815805.449578959] [decision_maker]: [STATE] AVOID
ING_OBSTACLE → LANE_FOLLOWING
[decision_maker-3] [INFO] [1764815815.253437979] [decision_maker]: [STATE] LANE_
FOLLOWING → SEARCHING
[decision_maker-3] [INFO] [1764815815.649487110] [decision_maker]: [STATE] SEARC
HING → LANE_FOLLOWING
[decision_maker-3] [INFO] [1764815816.049485636] [decision_maker]: [STATE] LANE_
FOLLOWING → SEARCHING
[decision_maker-3] [INFO] [1764815816.349568727] [decision_maker]: [STATE] SEARC
HING → LANE_FOLLOWING
[decision_maker-3] [INFO] [1764815819.150569911] [decision_maker]: [STATE] LANE_
FOLLOWING → SEARCHING
[decision_maker-3] [INFO] [1764815820.549374854] [decision_maker]: [STATE] SEARC
HING → LANE_FOLLOWING
[decision_maker-3] [INFO] [1764815822.085697011] [decision_maker]: [SIGN] Detect
ed: Parking Allowed
[decision_maker-3] [INFO] [1764815823.082770860] [decision_maker]: [SIGN] Detect
ed: Parking Allowed
[decision_maker-3] [INFO] [1764815826.554733745] [decision_maker]: [STATE] LANE_
FOLLOWING → SEARCHING
[decision_maker-3] [INFO] [1764815826.649663153] [decision_maker]: [STATE] SEARC
HING → LANE_FOLLOWING
[decision_maker-3] [INFO] [1764815826.984956422] [decision_maker]: [SIGN] Detect
ed: Parking Allowed
[decision_maker-3] [INFO] [1764815828.180148669] [decision_maker]: [SIGN] Detect
ed: Left Turn Sign
```

Here is an example of the sign detector reading the signs and how accurate the robot thinks it is. As you can see from the picture the robot is correct in both signs and in the terminal you can see it begin to start reacting to

Conclusion

This project has successfully demonstrated an understanding of the class material by creating a novice version of an autonomous driving system that is capable of lane following, traffic sign recognition, and obstacle avoidance inside of the simulated environment. Our system worked well as it allowed the robot to stay in the lines, avoid obstacles, and follow street signs. This was the original goal and we succeeded in creating the nodes to make that happen. While the system has its limitations, such as not being able to work properly in a real world environment and being limited to the course, it provides a foundation for an autonomous vehicle in the future and combines the things that we learned in class while also ensuring that we also did work that was not based on the labs such as the decision making node. Some of the lessons that we learned from this project was just how difficult it is to make a true autonomous vehicle. Although we had a beta version of an autonomous vehicle it would definitely not work in the real world as there are too many variables that we did not account for. Going off of this, fine tuning parameters was also incredibly time consuming and difficult to get right. Another thing that we learned from this project was how to make a simple autonomous vehicle package. Combining everything we learned from this class and making it all come together with the decision making node was incredibly satisfying and seeing everything we learned in class being used at the same time made us realize the importance of learning about the different nodes and how to create and utilize them.

References

1. Maji, S., Rath, T., Goswami, B., & Das, P. (2024). *Design and development of a package delivery robot*. International Journal of Research Publications. Retrieved from https://www.researchgate.net/publication/381280010
2. Yin, Y., & Aditjandra, P. T. (2020). *Autonomous delivery robots and their potential impacts on urban freight energy consumption and emissions*. Transportation Research Part D: Transport and Environment. Retrieved from https://www.researchgate.net/publication/340803013