Final Report: Turtlebot Autorace (as of 28th Nov, 2018)

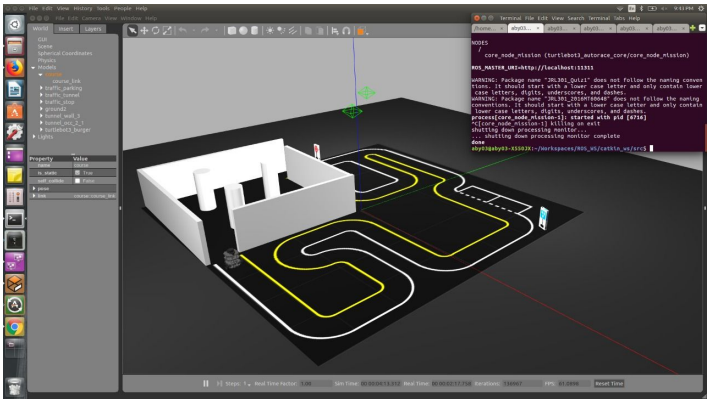
Submitted by: Abhay Saxena, Dhruv Talwar, Diptangshu Sen

TurtleBot3- Autorace
Model- TurtleBot3 Burger

[Gazebo World]
Turtlebot3_autorace

Summary of Work

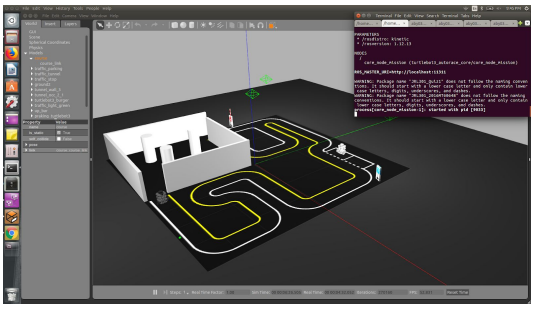
1. Bot detects lanes and follows it
    - Lane Detection done using threshold detection of yellow and white color in HSV coding of image
2. Identifies traffic signals and stops when red
    - Achieved using color threshold in HSV coding
3. Detects parking sign
    - Using point features for matching signboards in arena with already available images of that signboard with SIFT
4. Checks for availability of parking space and does a parking manoeuvre
    - Parking area identified by dashed line detection (Hough Transform)
    - Availability determined based on obstacle scan in that region
5. Reverse manoeuvre to exit parking space and align with the track
6. Detects level crossings
    - Achieved through image filtration and blob identification techniques
    - Orientation of level detected by standard geometry
7. Smoothly navigates inside a tunnel
    - Cost map generation through Laser Scanning to obtain shortest optimal path avoiding obstacles



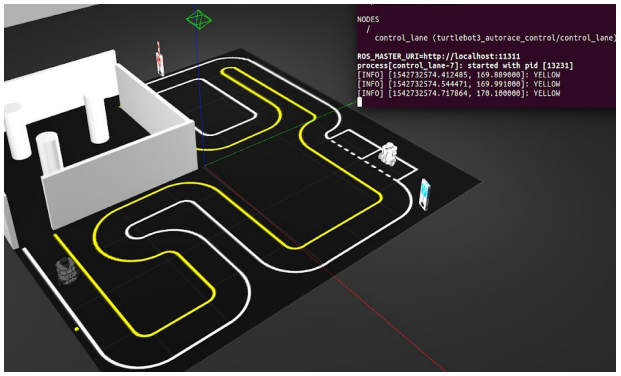
00: ROS Autorace Environment

Flow of Autorace Mission

1. Arena and Mission initialized: Traffic light spawned, static bot model spawned in parking, traffic barrier spawned, that will close when bot reaches near it
2. The bot moves by lane detection, it follows using yellow and white lines and is able to take turns without intersections
3. Bot detects yellow light, stops at red light, then continues on green light
4. Detects parking sign, looks for available space and goes into parking mode
5. Finally, exits parking space and aligns with road for tackling remaining objectives
6. Detects sign for toll gate
7. Identifies toll gate in horizontal position and continuously tracks its orientation to identify safe time of passage
8. Identification of tunnel entry point
9. LaserScan imaging enabled to create a costmap of the surrounding and chalk out the shortest optimal path through the tunnel

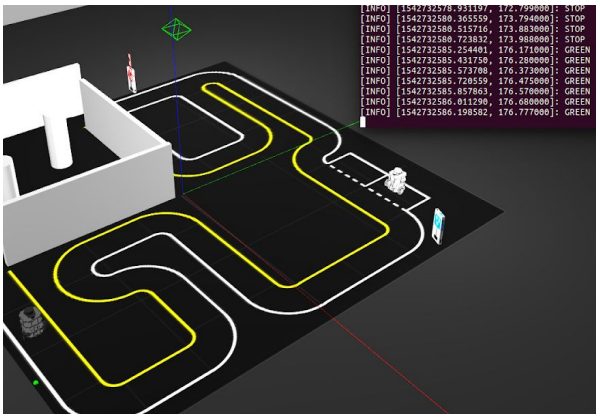


(1.0) Arena initiated, Autorace tasks spawned

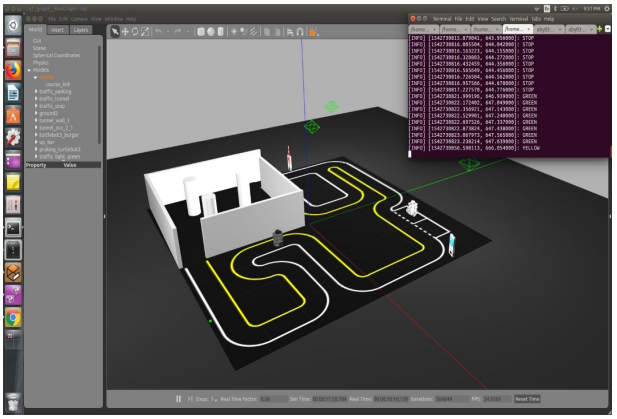


(2.1) Yellow light detection while lane following

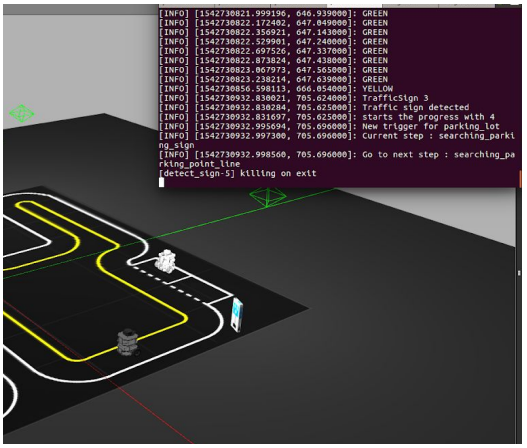
(3.1) Traffic signals identified, temporary stoppage at red light



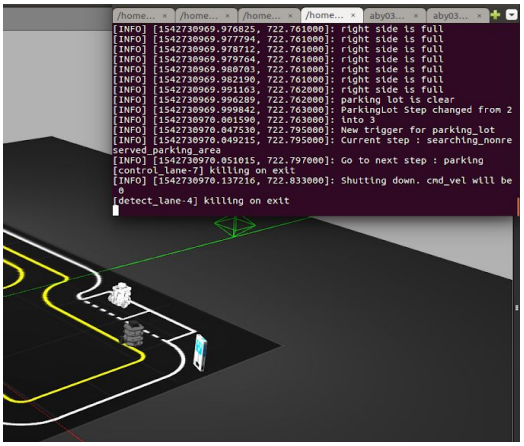
(3.2) Resumption of journey when signal turns green



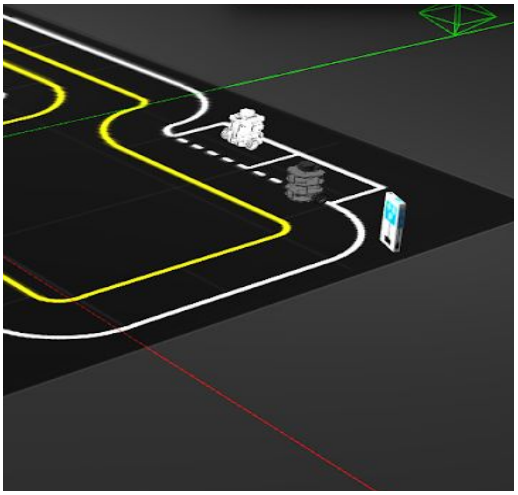
(3.3) Successful negotiation of turns



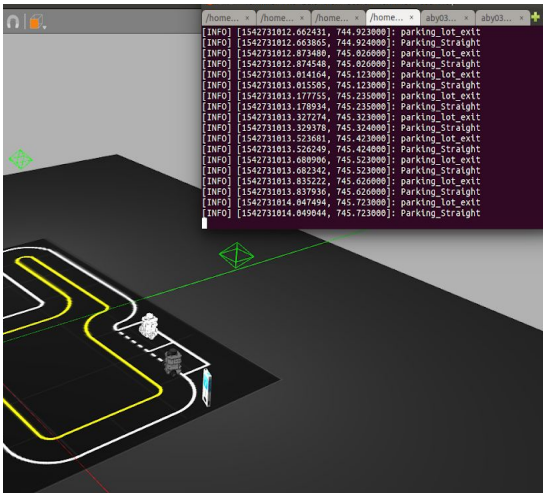
(4.1) Parking sign detected



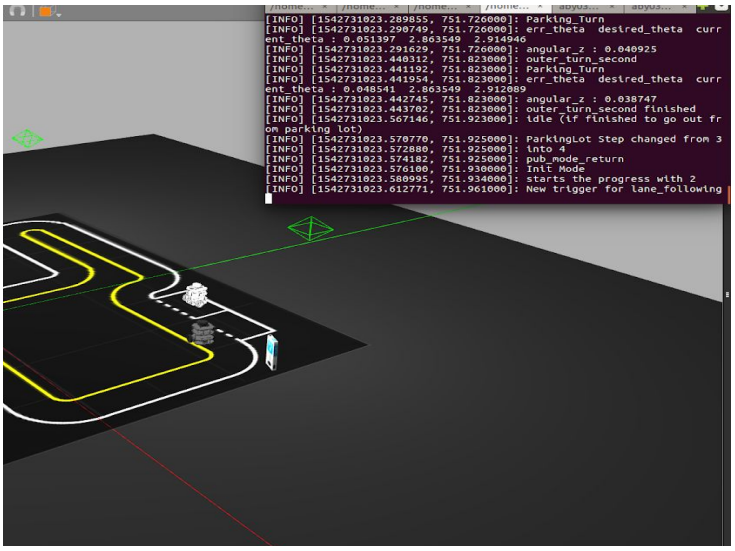
(4.2) Checking for parking availability



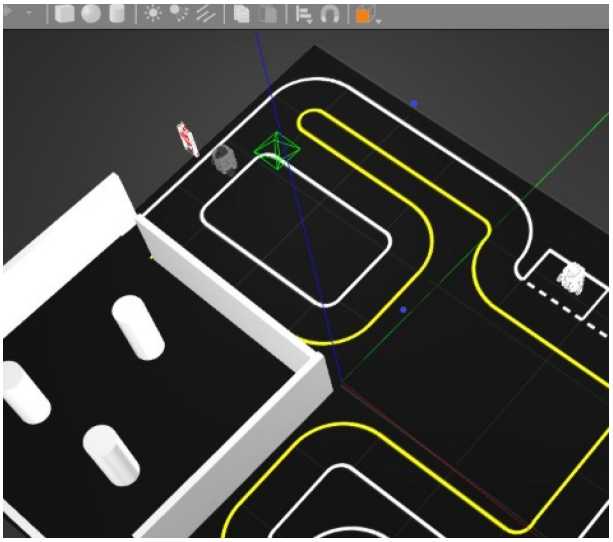
(4.3) Turtlebot manoeuvre to park in empty space



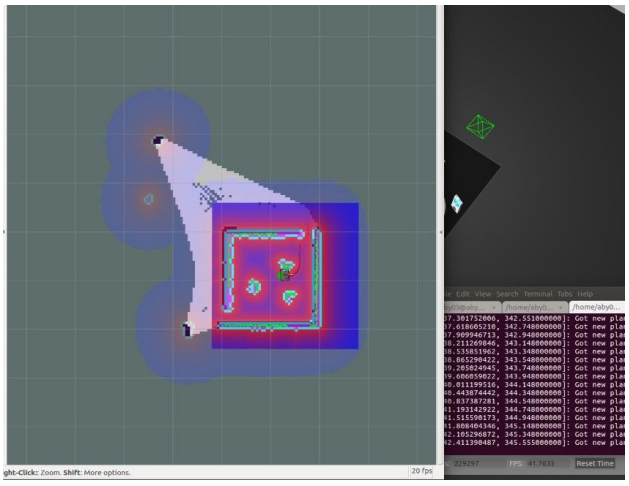
(4.4) Exiting parking lot



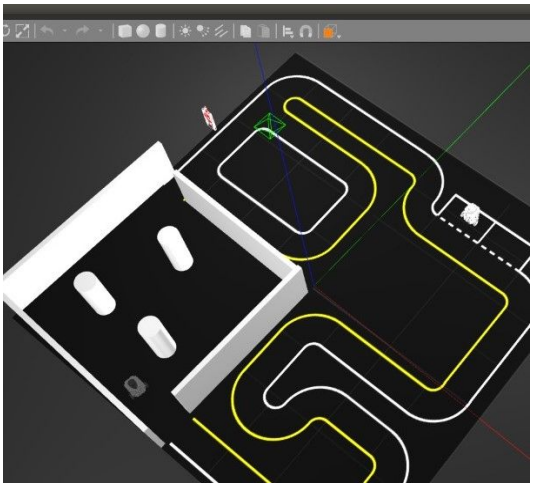
(5) Successful alignment with lane to continue the journey



(6) Detecting toll gate

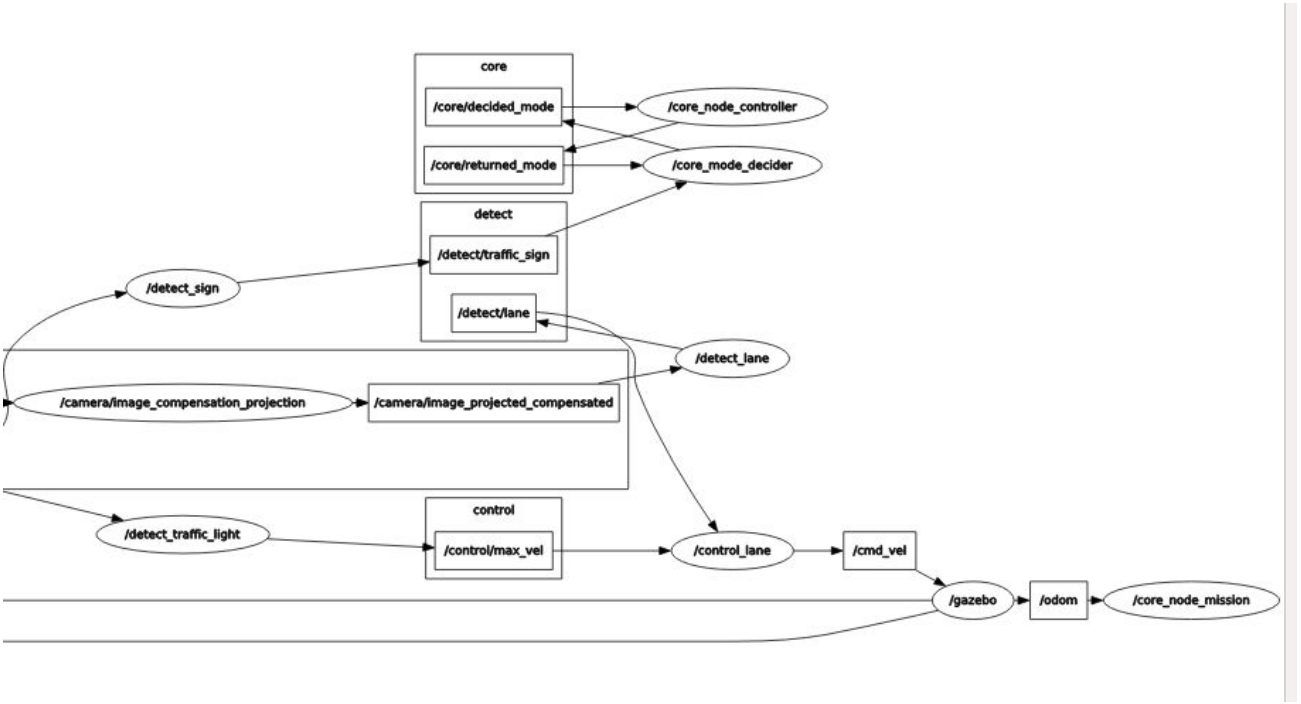


(9.1) Cost map of tunnel showing optimal path

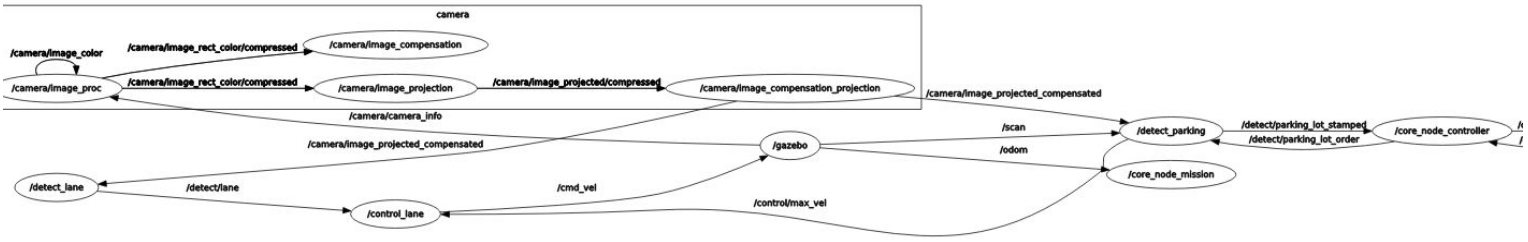


(9.2) Turtlebot navigating through tunnel

Rqt Graphs



6.1 Rqt graph while lane following



(6.2) Rqt graph while detecting parking along with lane following

Challenges

1) The passage of the turtlebot through the circuit is too slow. We tried tinkering with the max velocities to see if we could make it faster. But, even at slighter higher velocities, the bot behaves unpredictably near turns, misses traffic signals etc. We deduced that on exceeding a certain threshold, the sampling rate of the camera becomes so small that the bot moves through a considerable distance before the next image is captured, i.e., the “blindness range” of the bot increases. That is fatal. Our bot performed predictably at max velocities of 0.2 m/s.
2) The shortest path identification inside the tunnel hits a little bit of snag during the exit, when the bot has to iterate back and forth multiple times to find the ideal curvature to

smoothly move out of the tunnel. To overcome this, we tried the following approaches:

a) We tried reducing the threshold of the tunnel boundary, but it was not a very elegant solution.
b) We tried to discard the shortest path algorithm and navigate the bot along the wall to its right. It works for this particular scenario when the entry and the exit are diagonally situated, but does not for other kinds of circuits. The universality of the code fails. Also, the time of passage increases massively, which is undesirable.

3) Due to simulation running slow and requiring a lot of processing, there is a time lag in switching nodes requiring use of the camera, which sometimes leads to unpredictable behaviour or missed feature.