University of New South Wales

Sydney NSW 2052
Australia

School of Computer Science and Engineering

COMP3431 - Robotic Software Architecture

Term 3 - 2019

Self-Driving Robots

James Lin

December 1, 2019

1 Problem Statement

The goal of this project is to develop core functionalities that can be utilised in autonomous self-driving vehicles. This includes mapping, navigation and obstacle detection and avoidance.

Mapping represents the vehicle's ability to represent its local surroundings, road and obstacles as it explores unseen areas. An accurate construction and representation of the map allows additional functionalities for the vehicle such as path planning, the ability to navigate from point A to point B. Navigation is the vehicle's general ability to drive. This involves the vehicle's ability to accurately follow along a lane abiding by traffic rules, stopping at appropriate signals (eg, traffic light), turning at intersections, and driving on the correct side of the road. Obstacle detection and avoidance is another core component of the system, enforcing the safety of the vehicle and driver. Common obstacles involve other vehicles, unknown roadblocks and pedestrians.

2 Background

In terms of navigation, the core problem to tackle is the detection of road markers on the ground, as the vehicle must drive in between the lines. This becomes primarily a computer vision problem, as the laser is elevated and oriented in a position where it cannot detect markers on the floor.

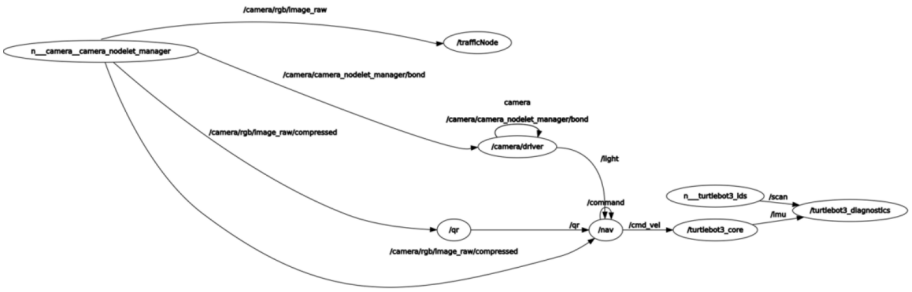
[1] first segments the image into sky, obstacle and road regions utilising K-means segmentation based on dark channels and a regional growth based method. Afterwards, lines are extracted based on a vertical envelope and the vanishing point determining roughly where the vehicle should travel to is calculated as the estimated intersection of two line groups. Similarly, [2] performs segmentation and vanishing point detection. However, segmentation is based on a GrowCut approach applied using image superpixels, and a Multiple Population Genetic Algorithm (MPGA) is used to search for vanishing point candidates based on a voting scheme. Rather than segmentation like the previously mentioned papers, [3] performs the task of road detection utilising a Canny edge detector, then an applied Hough transform to represent the road markers in the form of geometric lines. Additionally, Kalman filters were used to predict future lines and regions of interest to account for blurred images from vehicle movement and vibrations. Similarly, [4] uses the UNSCARF algorithm for road locating, comprising of clustering, edge detection, interpretation generation.

In terms of mapping, the goal is to provide an accurate representation of the road boundaries and markers that can then be used for navigation functionalities.

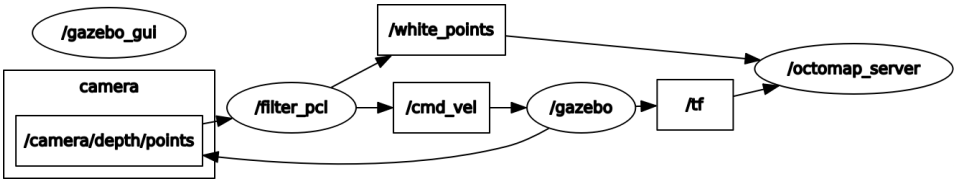
Perspective transform has been utilised by [5, 6] to provide a non-skewed top-down view of the road, which can then be utilised for further analysis. Furthermore, [7] demonstrates that a perspective transformation matrix can be calculated using the Top-View Transformation Model (TVTM), utilising the camera focal length, tilt angle and height from the camera to the ground. This was demonstrated with the application of a vision-based car parking assistant. Furthermore, there are many visual-based Simultaneous Localisation and Mapping (SLAM) methods such as ORB-SLAM [8], ORB-SLAM2 [9] LSD-SLAM [10], L-SLAM [11] and OpenRatSLAM [12] that utilises the RGB-d camera to accurately approximate the vehicle location utilising key features in the image [13].

3 Team Solution

Overall, the project implemented several solutions to mapping, navigation and obstacle detection. The interaction of nodes and topics is visualisated in Figure 1.



(a) RQT Graph for Histogram & Gradient-based Navigation with Traffic Light detection



(b) RQT Graph for Point Cloud Mapping - Octomap

Figure 1: RQT Graphs

3.1 Mapping

In terms of mapping, three methods were proposed: mapping with raw depth camera point cloud data, mapping with point cloud using Octomap package, and mapping with pure RGB image input. Raw point cloud mapping was implemented without testing, while Octomap was tested in the Gazebo simulation environment, and RGB mapping was tested in the real-world environment.

3.1.1 Point Cloud Mapping - Raw

Implementation

One solution to mapping is to convert the inputs from the point cloud into coordinates on an OccupancyGrid which represents the map. Firstly, the point cloud topic was subscribed to by the node, and only green and white point cloud markers were extracted and processed representing the green

road region and white road markers respectively. Further pre-processing to remove noisy points was performed. Transformations using the depth information from the points and the relative position between the camera frame $/camera\_rgb\_optical\_frame$ and global frame $/map$ from SLAM was performed.

3.1.2 Point Cloud Mapping - Octomap

Implementation

An alternative solution that also utilises the point cloud data utilises the Octomap ROS package. Similarly, the point cloud is pre-processed to remove noise and only white points representing road markers were extracted. The Octomap package was used specifying the $/odom$ frame and stitched together with the map produced from SLAM's Gmapping package.

Evaluation

This method was only tested utilising Gazebo simulation environment with a pre-constructed road map identical to the real-world environment. A sample output is seen in Figure 2. Generally, the mapping appears to be accurate with a few inconsistencies with the mapping during rotation and points with relatively large depths. However, this method should be tested in a real-world environment where sensor data and transforms may be inconsistent or out of sync.

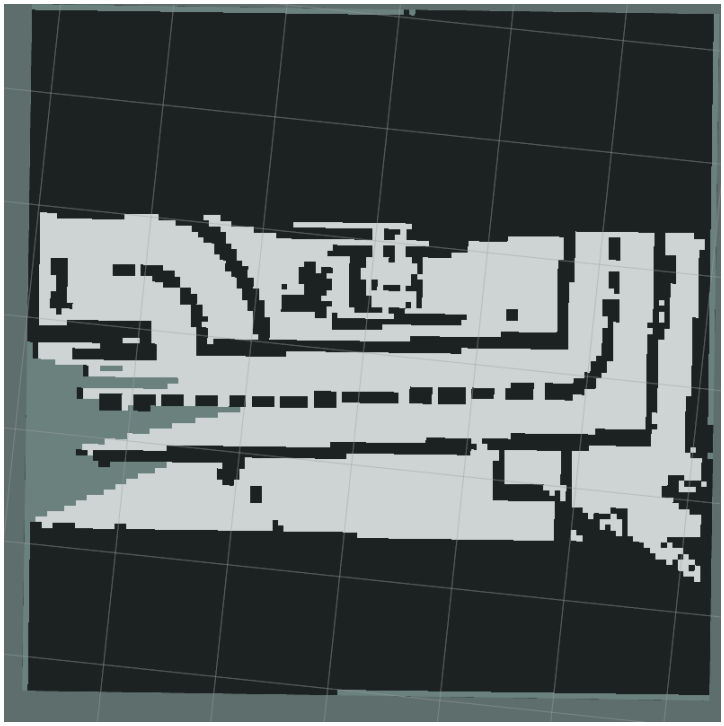


Figure 2: Octomap Output on Gazebo Simulation Environment

3.1.3 RGB Mapping

Implementation

RGB mapping differs from Point Cloud Mapping as it does not make use of the depth camera. It interpolates the depths of individual pixels from the raw RGB image based variables of the RGB camera such as distance from the ground, angle of tilt and focal length. For this method, the goal is to produce a grid-based representation of the map showing road markers relative to the position of the robot. Other obstacles such as buildings and vehicles are ignored. The implementation consists of three main steps: segmentation of road markers, the perspective transformation of the image, and then the pixel-wise transformation to a grid representation of the road. The process is summarised in Figure 3.

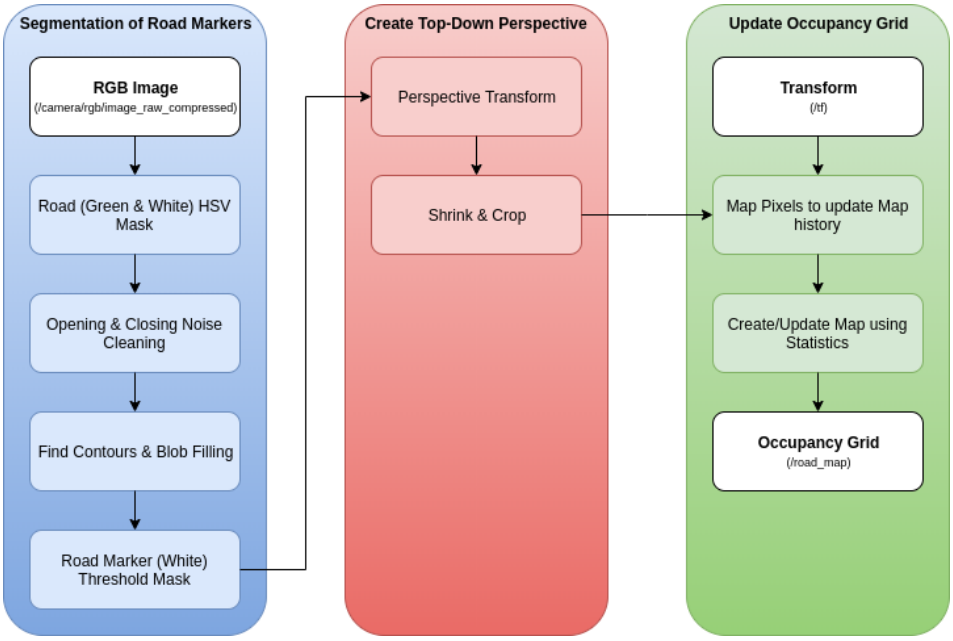
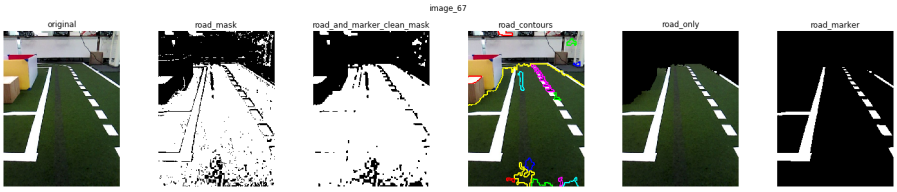


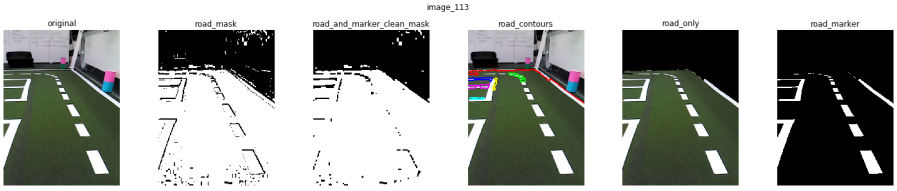
Figure 3: RGB Mapping Flow Chart

Segmentation of Road Markers

As this step suggests, the goal is to only preserve information about white road markers in the raw RGB image, ignoring other obstacles and vehicles. Segmentation was performed by taking the input RGB image and first detecting and masking the road region using HSV colour ranges. This is to prevent the classification of white items outside the road region as road markers in future steps. After the HSV mask, noise removal was performed using opening followed by closing morphological operations with a kernel size of (5x5), and the Find Contours algorithm was used to identify individual segments of the mask. The largest contour in terms of area coverage was taken and assumed to represent the road region, and ultimately used to create a road mask. Thereafter, road markers were identified by a grayscale threshold on the road mask. This whole process is illustrated in Figure 4



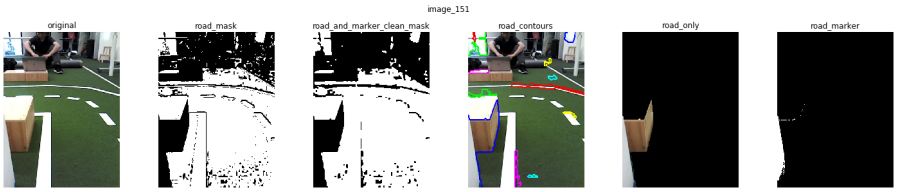
(a) Segmentation Process for Image with obstacles inside road



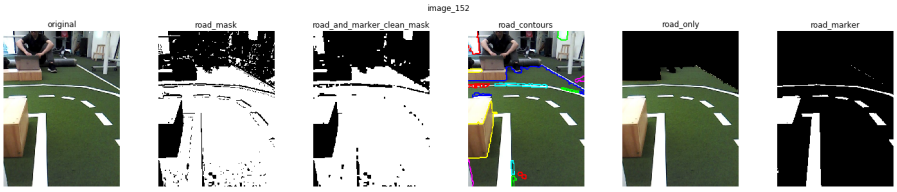
(b) Segmentation Process for Image with white background objects

Figure 4: Segmentation of Road Markers Process (Good outputs). From left to right: Original image; Applied HSV Road Mask; Opening & Closing Noise Removal; Find Contours; New Road Image; Threshold on New Road Image

Figure 4a demonstrates that this process is capable of removing objects inside the road region, and Figure 4b shows that it can ignore white background objects such as the cupboard and table, as they are of similar colour to the white road markers.



(a) Segmentation Process for Image with small road region



(b) Segmentation Process for Image with obstacle inside road

Figure 5: Segmentation of Road Markers Process (Bad outputs). From left to right: Original image; Applied HSV Road Mask; Opening & Closing Noise Removal; Find Contours; New Road Image; Threshold on New Road Image

On the other hand, the algorithm is not always successful as demonstrated in some sample outputs in Figure 5. For example, in scenarios where Find Contours finds that the largest by area contour does not represent the road, then the masking would show a different output and the ultimate result shows essentially a black image, as shown in Figure 5a. However, this output occurs quite infrequently, around 1.2% of the time, and should not cause major issues in mapping as another image taken at a similar location is likely to be properly segmented. Another bad output occurs when white objects are located inside the road region, thus are considered to be white road markers as well. While this output has less of a detrimental effect in terms of mapping as the white objects should still exist outside the road boundaries in the map, it should still be fixed in future considerations.

Creating a Top-Down Perspective

The goal of this step is to convert the newly masked road markers into a top-down perspective. The RGB camera is mounted on the robot with height H from the ground and tilt angle θ to produce a field-of-vision. However, due to this orientation, the image of the road markers do not correctly represent their length and angles and need to be transformed to a top-down view to minimise these perspective errors, as shown in Figure 6.

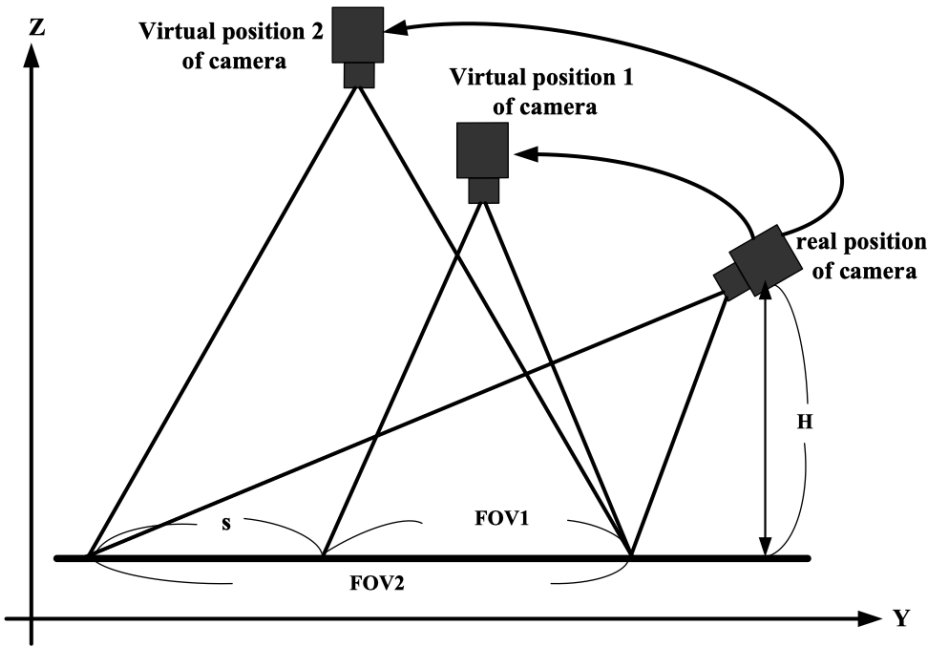


Figure 6: Comparison of camera positions [7]

In order to do this, a homography matrix is created, mapping four coordinates in the original image to four coordinates in the desired warped image. The homography matrix can be approximated using trial-and-error, as the desired warped image should maintain the width of the road to be the same length as perceived in the image. In the original image, the width of the road appears to be decreasing as the road is further away from the camera (or going up the image).

After a homography matrix is created, it is used to transform the original image into its warped perspective counterpart. After the image is shrunk to reduce computational resources and memory, cropped to remove the potential unreliable noisy road markers around the upper areas of the mask, and post-processed for mapping transformation. This process can be visualised in Figure 7.

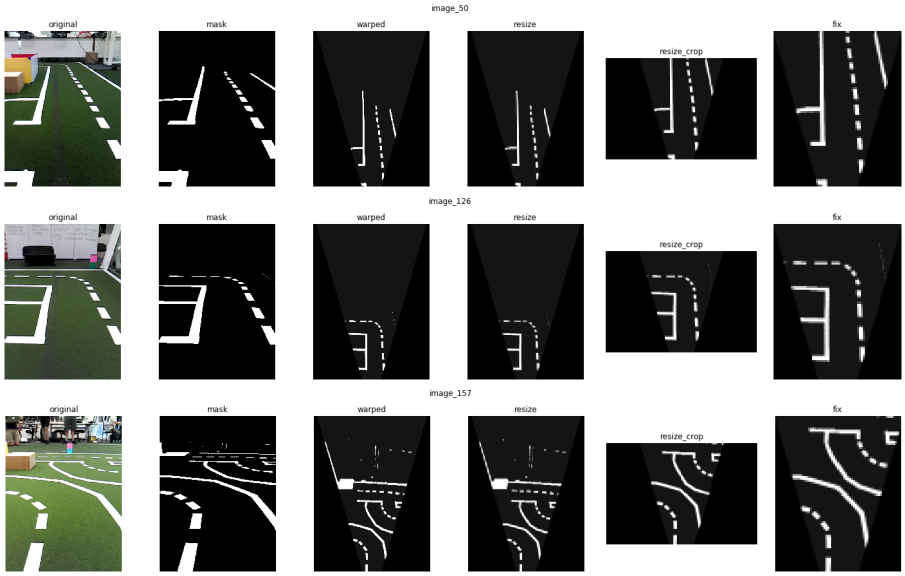


Figure 7: Top-down Transformation of Road Markers Process. From left to right: Original image; Road Mask; Perspective Transform; Shrinked Image; Cropped Image; Post-processed Image

Update Occupancy Grid

After all the image processing steps, it is finally time to convert the image to a map. The goal for this step is to map each individual pixel to an (x,y) coordinate relative to the global frame, which could be either */map* or */odom*. In order to achieve this, the pixels must first be converted to a position relative to the robot represented by the */base_footprint* frame, then a translation to the global frame.

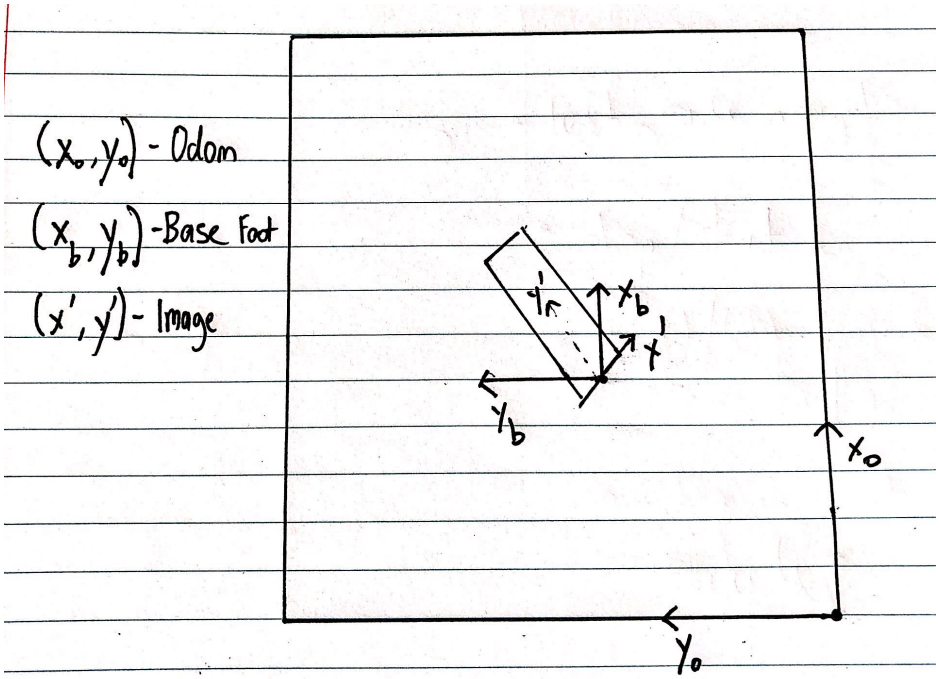


Figure 8: Relative Transform Coordinates of Odometer, Base Footprint and Image from Camera

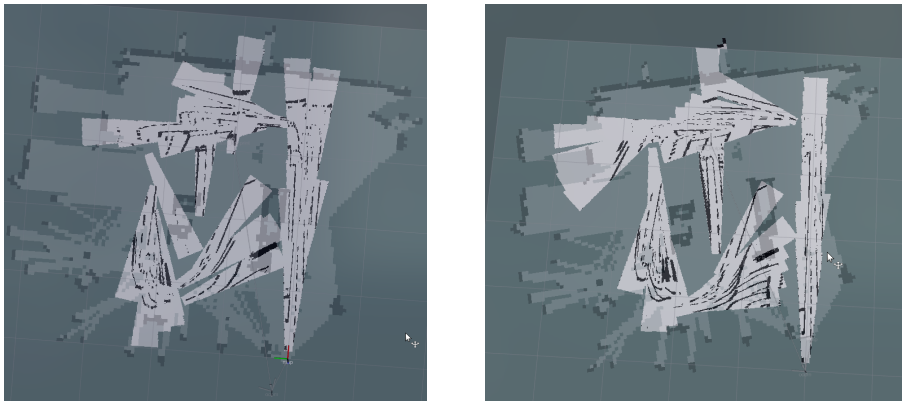
From the representation of transform coordinates shown in Figure 8, the conversion equation is:

$$y = -\acute{x}cos\theta + \acute{y}sin\theta + y_{bo} \tag{1}$$

$$x = \acute{x}sin\theta + \acute{y}cos\theta + x_{bo} \tag{2}$$

where (x, y) represents the position in the global frame, and (x_{bo}, y_{bo}) represents the relative coordinate of the $/base\_footprint$ from the $/odom$ or $/map$. A map grid is then created by iterating this process for every pixel in the image for every timeframe where an image is received.

The actual map is generated by averaging and then thresholding all the historic map representations. To prevent unreliable white road markers due to a mixture of sensor readings and image processing, a white road marker must have at least a 30% confidence with a minimum of 5 sightings. These are hyperparameters that can be tuned by the user. The output is shown in Figure 9.



(a) Output with /*map* as Global Frame (with SLAM)

(b) Output with /*Odom* as Global Frame (no SLAM)

Figure 9: RGB Mapping Output on RVIZ

Evaluation

Ultimately, the map output shown in Figure 9 shows many inconsistencies. Road markers do not align, and there are often unwanted repeated features visible in different areas of the map. Several potential factors are likely to have attributed to the inconsistent output. Firstly, the sensor readings and transform values that are received are likely to be inconsistent. Without using SLAM, readings from /*odom* are likely to experience positional error growth over time. While using the SLAM Gmapping module in ROS, the lack of landmarks during the robot test drive may result in inconsistent positional readings as well. Furthermore, the /*map* frame being constantly adjusted during the test run also results in an output that appears more inconsistent than that produced from /*odom*. Another reason is the depth interpolation in the image. Each pixel in the RGB image has an associated depth based on the width and height of the field of view, and there could be an incorrect conversion. Other potential errors could be attributed to an inaccurate perspective transform or segmentation. An optimisation of algorithms to improve processing time could improve potential synchronisation issues as the image is transformed into a map.

ROS Setup and Packages

This mapping functionality was implemented by creating a node *map_roads*, that subscribes to /*camera*/*rgb*/*image_raw*/*compressed* and /tf while they are synchronised. A map is generated and published as an OccupancyGrid object to topics /*road_map*. Computer Vision tasks were implemented using *numpy*, *OpenCV* and *scikit* − *image* in Python2.

3.2 Navigation

Two main algorithms were developed, implemented and tested for navigation. This comprises of histogram and gradient-based navigation, and navigation using a deep learning framework.

3.2.1 Histogram & Gradient-based Navigation

Implementation

Histogram & Gradient-based Navigation (HGN) interprets an image and makes a decision based on the distribution and positioning of road markers. It assumes various road orientations such as straight road, curved road, sharp-curved road, and tells the robot what velocity and angular velocity to drive at. Overall, the implementation of this algorithm is divided into three main steps: segment and transform image, gather points and data, and choosing an action. This is summarised in the flow chart shown in Figure 10.

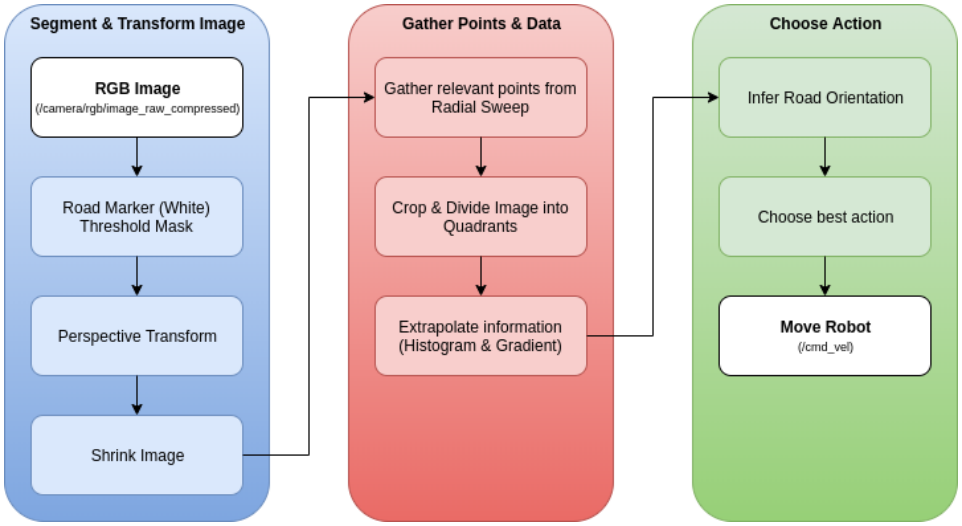


Figure 10: HGN Flow Chart

Segment & Transform Image

The goal of this stage is to process the image into a state where information about roads can be gathered. Firstly, the raw image is thresholded to segment out white regions. The image then goes through a perspective transform and shrink stage to produce a top-down perspective of the roads. These algorithm used in these steps is identical to that explained in the RGB Mapping section. The main difference is that a road mask is not produced.

This is a redundant step due to the radial sweep step performed during the gather points and data stage. These steps are illustrated in Figure 12.

Gather Points & Data

The image is then processed via a radial sweep algorithm from the bottom centre of the image to simulate finding nearest objects around the camera similar to a LiDAR scan. The concept of a radial sweep is shown in Figure 11 where the green point represents the origin (in the project, this is located at the bottom centre), the black lines should represent the white road markers, and the red points represent the intersections from a particular angle extruding from the origin with a black line. After a radial sweep is performed, the image is divided into quadrants for future processing. In the final implementation, the bottom left and right quadrants make up one-eighth of the image, and the upper left and right quadrants make up the rest of the bottom half of the image. The upper half of the image is disregarded as the proportion of noisy points increases drastically.

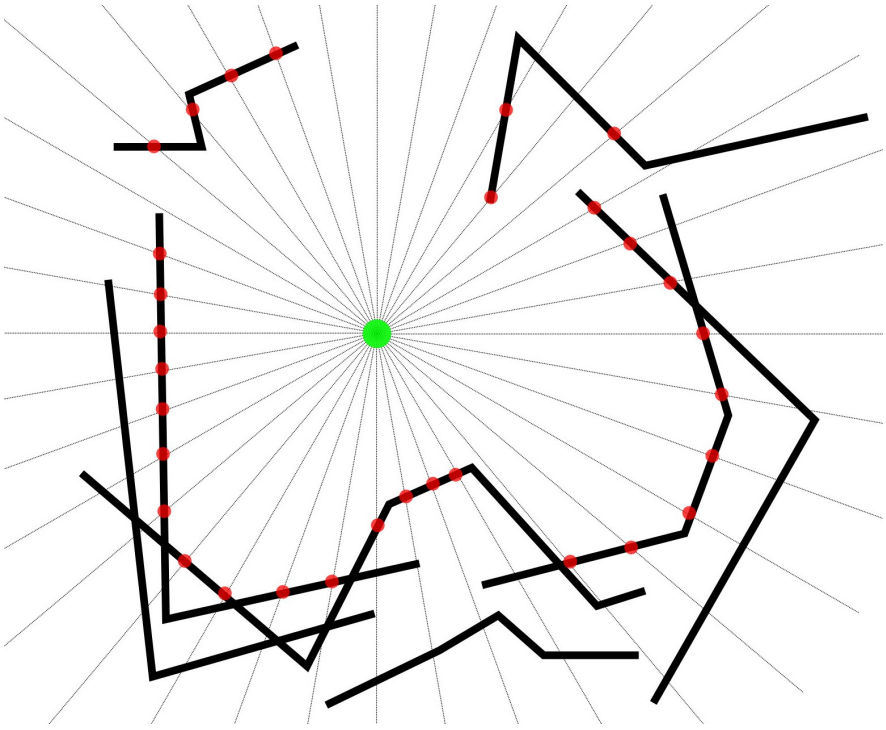
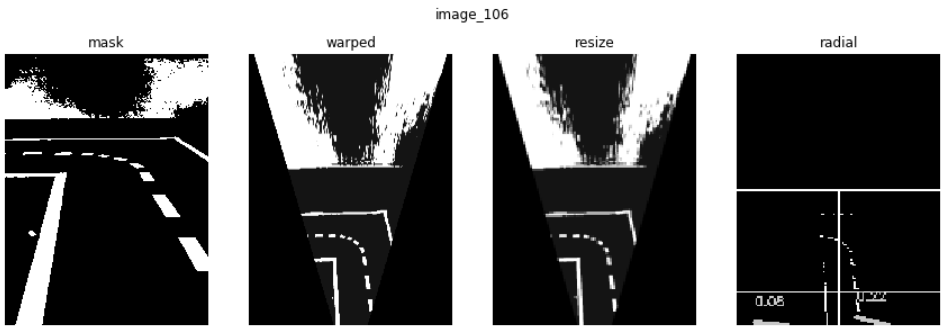


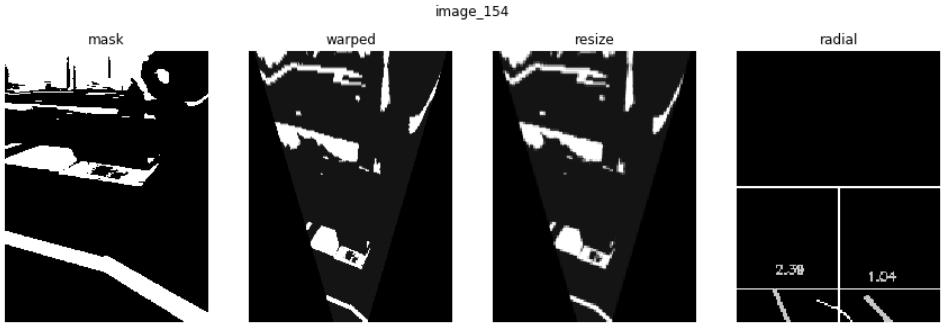
Figure 11: Radial Sweep Diagram

With the collection of points in each quadrant of interest, information can be collected and processed utilising statistical and line-fitting models. In the final implementation, a histogram to determine the frequency and value of the most frequent point in terms of x-value, and a line-fitting model to

determine the overall gradient of the points was used to choose the best action. Other potentially useful information includes standard deviation, range, an average based on x and y histograms could be used in the future for fine-tuning.



(a) Straight road with minimal gradient



(b) Curved Road with higher gradients

Figure 12: Segmentation, Transformation and & Point Gathering Process. From left to right: White mask; Perspective Transform; Shrinked Image; Radial Sweep with four quadrants & gradient information

Choose Action

In this stage, the data is processed to infer information about road orientation, and an action is ultimately decided. Assumptions were made so that roads could be straight, slanted (to the left or right), or sharply curved (to the left or right) as shown in Figure 13. This information can be inferred from the histogram and gradient data collected in the previous stage using Table 1.

(a) Straight Road (b) Slant left-curve Road (c) Sharp right-curve Road

Figure 13: Examples of Road Orientations

Table 1: Histogram & Gradient to Road Orientations Mapping

X Frequency	Gradient	Road Orientation	Action
High	High	Sharply Curved	Very Low velocity and High angular velocity
High	Low	Straight	High velocity and Small angular velocity for alignment
Low	High	Slanted	Medium velocity and Medium angular velocity
Low	Low	Unknown	High velocity and Zero angular velocity

In the case of a straight road, the vehicle tries to align itself so that it is positioned in the centre of the road. This was achieved generating an angular velocity that is proportional to the difference of displacement of the most frequent x-value from the desired position.

As the actions each apply to a separate quadrant in the image, the final action taken is the maximum angular velocity, and the minimum speed out of all the quadrants.

Evaluation

Overall, the robot was able to navigate curved-roads, sharply-curved roads, and straight roads with a few inconsistencies. One of the main issues with the proposed algorithm is that there is a road marker blob at the bottom centre of the transformed image, then the radial sweep would only identify points clustered in that area, losing lots of valuable information it could've gathered. Furthermore, with this algorithm, the robot turns curved roads too early and crosses the solid line to turn instead of staying between the lanes. On the other hand, some situations in sharp turns, the robot turns too late and drives out of the road boundaries. While the main focus of the algorithm is on lane following, it can be adapted to ensure that the vehicle drives on the correct side of the road. Furthermore, the additional fine-tuning of global hyperparameters such as gradient thresholds, frequency thresholds, turning rates and speeds can improve the performance of the robot in navigation.

ROS Setup and Packages

This navigation functionality was implemented by creating a node *nav*, that subscribes to */camera/rgb/image_raw/compressed* that runs a call-back running the implementation described. An action in the form of speed and angular velocity is generated and published as a Twist object to the topic */cmd_vel* that is interpreted by the robot motors to drive the robot. Computer Vision tasks were implemented using *numpy*, *OpenCV* in Python2.

3.2.2 Deep Learning

Implementation

An alternative implementation to navigation was a deep learning approach utilising convolutional neural networks. The problem domain was represented as a classification problem, where the input to the deep learning model is the RGB image from the mounted camera on the robot, and the task is to predict whether the robot should turn left, right or go straight. Firstly, the image was processed to improve the performance of the model. This involved rotation to the correct orientation, cropping and resizing the image to decrease the computational complexity of training the CNN. The model comprises of a series of convolutional layers, max-pooling functions, ReLU activation functions, and a final fully-connected layer and softmax function to output probabilities. The maximum probability correlating a left, right, or straight road is then mapped to a fixed linear and angular velocity. Rotations were fixed with a linear velocity of 0.08, and an angular velocity of 0.2 to provide smooth navigation experience. However, this makes the solution less generalisable to similar problems.

Evaluation

During the demonstration, the robot was able to demonstrate smooth navigation on straight roads and curved turns, with a few inconsistencies. The overall performance was better than that of the HGN navigation algorithm. However, the CNN may have overfitted as the training and testing environment is the same, and thus the robot should be tested on unexplored roads.

3.3 Additional Features

Outside of mapping and navigation, several small-scale functionalities were implemented to improve the overall driving performance. This includes traffic light detection in the form of coloured beacons, QR codes, as well as obstacle detection using the LiDAR sensor.

3.3.1 Traffic Light Detection

Implementation

Traffic light detection was implemented utilising two-coloured beacons to mimic traffic lights in the testing environment. This functionality commands the robot to stop when a red traffic light is seen as the top colour of the beacon and to go otherwise. Beacon object detection was performed using a combination of HSV thresholding and blob detection with additional processing to remove blobs with too large or small areas, and incorrect aspect ratios. In ROS implementation, a node $trafficNode$ was created that listens to RGB images coming from the topic $/camera/rgb/image\_raw$, performs the algorithm and sends a boolean message to topic $/lights$ for the navigation module to listen to. In terms of navigation, a *true* message forces the robot to halt completely, and a *false* message enables it to continue moving utilising its navigation algorithm.

An alternative version to traffic light detection utilises QR codes rather than beacons. When a robot detects a QR code, the robot stops for a few seconds, then turns into an intersection instructed by the QR code. In the timeframe of this project, only QR detection was implemented and tested. This was achieved on a high level using Python wrapper package *pyzbar* that implements *zbar* aimed to read various bar codes. A thread was generated to countdown 3 seconds before resuming the navigation, and a timing cooldown of 30 seconds was implemented so the robot does not immediately stop after seeing the same QR code right after resuming navigation.

Evaluation

This module was tested with both navigation algorithms in the real-world environment. Traffic light detection generally worked with a few uncommon inconsistencies. There was one moment where it consistently incorrectly detected a red and yellow coloured building as a traffic light. With some fine-tuning of parameters and shapes, then the results of traffic light detection can improve.

On the other hand, QR detection was tested and ran without much issue due to the simplicity of the algorithm. One potential factor to consider is the long time it takes to run the algorithm that may potentially cause loss of sync in other nodes or delays in the robot's actions.

3.3.2 Obstacle Detection & Avoidance

Implementation

A simple implementation of obstacle detection prevents vehicles from driving when there exists a large obstacle in front of the robot. This can be achieved by utilising the LiDAR sensor. A *laser_stop* node is created that listens to the */scan* topic retrieving a *LaserScan* object. The algorithm checks if the average depth of scans between two angles, -35 degrees and 35 degrees representing the front of the robot is less than a defined global threshold. If it is, then a large object is detected and a *stop* message is published to the navigation module to override the motors and command the robot to completely stop. Otherwise, the navigation module is free to explore if a large object is not detected.

4 Contribution

In terms of core functionality, I was primarily involved in developing RGB mapping and HGN functionalities. This involved researching and developing a computer vision pipeline for segmentation and perspective transformation of images and determining the transformation mathematical equations to convert pixels to coordinates in an OccupancyGrid. For HGN, I wrote the implementation of the algorithm as a ROS node to interface with the robot and generated ideas about road orientations and gradients. Furthermore, I implemented small-scale additional functionalities such as QR detection, obstacle detection and avoidance, and interfaced the TrafficNode module with the robot command module. I discussed viabilities of solutions of other algorithms such as deep learning and provided skeleton code and utility functions for other team members to work with.

5 Evaluation

An evaluation of each functionality is discussed in their respective section in the Team Solution section. Generally, the algorithms were developed and evaluated using rosbags, Gazebo simulation environments, and ultimately the real-world testing environment using the Turtlebot3 Waffle robot model.

In terms of mapping, Point Cloud Octomap appeared to produce the most accurate output despite only being tested in the Gazebo environment. Point Cloud Raw mapping was implemented but not tested, and RGB mapping was primarily tested using rosbags representative of the real environment. RGB mapping suffered from too many inconsistencies due to transform values using either SLAM or only the odometer.

In terms of navigation, the robot was generally able to navigate the whole course using the HGN algorithm, with a few uncommon inconsistencies. The robot turns too early during slightly curved roads and crosses the solid line which is not desired. Furthermore, the robot can handle straight road navigation with alignment fixes and sharply curved roads. The deep learning solution appeared to work much more smoothly during the demonstration, especially during curves. However, it would be desired to test the robot on a different unexplored course as the CNN may have overfitted from being tested and trained using the same environment.

In terms of additional features, traffic light detection performed decently during the demonstration. Beacon detection did not account for building obstacles with a similar colour scheme to the coloured beacons, while QR code sometimes appeared to detect non-QR code objects and force the robot to stop undesirably. The obstacle detection functionality using the LiDAR sensor was not able to work during the demonstration due to conflicting messages with other modules. This can be easily fixed for future iterations of the project.

6 Conclusion & Future Work

In terms of future implementations, the performance of the robot can be greatly improved with a few small bug fixes and hyperparameter tuning. It would be desirable to be able to test the robot while it simultaneously runs the navigation and mapping implementation to see if there are any syncing issues. RGB Mapping could be tested with a properly configured Gazebo environment to identify if the unreliable transformations would still occur. Furthermore, the perspective transform homography matrix could be calculated instead of relying on trial-and-error and also be calibrated with a grid. For HGN navigation, a workaround can be found for cases where the radial sweep gets blocked by a white region on the bottom centre of the image. Deep learning can be tested in different environments, and alternative navigation using a wallfollow approach could be experimented with. Additionally, an accurate map could be utilised for point-to-point navigation. In terms of future additional functionality for navigation, the robot should be able to drive on the correct side of the road, and handle intersections instead of just lane following. Generally, working on this project was a great learning experience that provided plenty of insight into the difficulty and challenges of robotics.

The project itself can be improved by providing properly documented specifications of expected functionalities and tasks with examples. Additionally, the freedom to play and tinker around with robots other than the Turtlebot3 Waffle that was already used for assignment one would be a nice change of pace. Furthermore, including more road courses, various obstacles can increase the challenge for motivated students.

References

[1] Yong Li et al. "Road detection algorithm for autonomous navigation systems based on dark channel prior and vanishing point in complex road scenes". In: *Robotics and Autonomous Systems* 85 (2016), pp. 1–11.

[2] Keyu Lu et al. "Vision sensor-based road detection for field robot navigation". In: *Sensors* 15.11 (2015), pp. 29594–29617.

[3] Farid Bounini et al. "Autonomous vehicle and real time road lanes detection and tracking". In: *2015 IEEE Vehicle Power and Propulsion Conference (VPPC)*. IEEE. 2015, pp. 1–6.

[4] Jill D Crisman and Charles E Thorpe. "Unscarf-a color vision system for the detection of unstructured roads". In: *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. IEEE. 1991, pp. 2496–2501.

[5] Ziqiong Liu, Shengjin Wang, and Xiaoqing Ding. "Roi perspective transform based road marking detection and recognition". In: *2012 International Conference on Audio, Language and Image Processing*. IEEE. 2012, pp. 841–846.

[6] Youfu Wu and Zusheng Chen. "A detection method of road traffic sign based on inverse perspective transform". In: *2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS)*. IEEE. 2016, pp. 293–296.

[7] Chien-Chuan Lin and Ming-Shi Wang. "A vision based top-view transformation model for a vehicle parking assistant". In: *Sensors* 12.4 (2012), pp. 4431–4446.

[8] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system". In: *IEEE transactions on robotics* 31.5 (2015), pp. 1147–1163.

[9] Raul Mur-Artal and Juan D Tardós. "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras". In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1255–1262.

[10] Jakob Engel, Thomas Schöps, and Daniel Cremers. "LSD-SLAM: Large-scale direct monocular SLAM". In: *European conference on computer vision*. Springer. 2014, pp. 834–849.

[11] Pyojin Kim, Brian Coltin, and H Jin Kim. "Linear RGB-D SLAM for planar environments". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 333–348.

[12] David Ball et al. "OpenRatSLAM: an open source brain-based SLAM system". In: *Autonomous Robots* 34.3 (2013), pp. 149–176.

[13] Arthur Huletski, Dmitriy Kartashov, and Kirill Krinkin. “Evaluation of the modern visual slam methods”. In: *2015 Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT)*. IEEE. 2015, pp. 19–25.