Autonomous Mobility on Demand: Goto-N Final Report

Alexander Hatteland, Marc-Philippe Frey, Demetris Chrysostomou

1.Introduction:

1.1 Mission:

The Artificial Intelligence Driving Olympics (AIDO) is a competition, hosted on the Duckietown Platform, focused on AI for self-driving cars. Participants submit their solutions to a specific challenge online. The solutions is initially tested on a simulator, before it is tested on real-life Autobots in a Robatorium. In order to do this, individual Autobots start at predetermined positions within the Robatorium and execute the participants solution. Upon finishing this, the Autobot has to be reset and repositioned at the initial starting position (to ensure a consistent testing procedure when evaluating the solutions. Currently, this process is done manually. It requires the operator to enter the Robatorium, pick up the Autobot and move it to the desired position. Not only is this process cumbersome and slow, it is also prone to human error and can cause undesired changes in the acutal town.

The goal of this project is to implement a reset mechanism for the aforementioned problem. It aims to create a node that can localize *n* Autobots in the Robatorium and autonomously navigate them to *n* termination (actually the starting positions for the AIDO submissions) positions. It also hopes to reduce the need for human intervention when evaluating AIDO submissions while increasing the repeatability and the speed with which the AIDO submissions are evaluated. Finally, we hope to create a consistent baseline from which solutions are tested.

1.2 Existing Work & Preliminaries:

The Duckietown Platform does not have a system in place that can reset Autobots to termination positions. Consequently, we had to conceptualize a pipeline that would achieve our desired outcome. We make use of the pre-existing localization system, which we adapted to fit our specific latency requirements, to localize the Autobots. We then feed this information into a global planner that creates waypoint intersection commands for the individual Autobots. Finally, once the last intersection is passed, the Autobot switched to a precision node that uses a closed-loop system to cross-reference the bots current and desired position.

While most of the implementation does not require any additional mathematical understanding, it should be noted that the planner does implement some mathematical theory. The path planning was done using dynamic programming. The map was turned into nodes, each associated with a specific cost, and we use value iteration to try to find the lowest cost paths. As a result, readers who are not familiar with the theory behind dynamic programming and value iteration might be interested in looking at the following resources:

- Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., & Bertsekas, D. P. (1995). *Dynamic programming and optimal control* (Vol. 1, No. 2). Belmont, MA: Athena scientific.
- Puterman, M. L. (1990). Markov decision processes. *Handbooks in operations research and management science*, *2*, 331-434.

2.Problem Definition:

In order to complete the given challenge, we have identified three separate parts that make up our problem definition. This was done to create a split between the functionalities of the project, allowing us to work towards and test not only the entire pipeline, but also individual aspects composing the entire pipeline. We have identified the following definitions:

Localize n Autobots in a Duckietown and autonomously navigate them to n termination states with precision of +/- (see what test results we get and change accordingly).

Develop a multi-agent planning algorithm for up to three Autobots

Develop a robust and repeatable system

2.1 Assumptions

We hope that the system is able to navigate the Autobots to the desired positions autonomously. In order to achieve this, we have established the following assumptions about the operation of the system and the state of the Robatorium.

1. We are given the Map in a YAML format consistent with Duckietown World
2. All Autobots within the Robatorium are controlled only by Goto-N.
3. The orientation and position of the termination state is given. The termination state is within a valid lane, not out of bounds and not in an intersection.
4. The starting position of the Autobot is valid. This means the Autobot is in the lane with the correct orientation. It does not have to be rescued, or is outside of the given map parameters.
5. Localization system is online and can localize the Autobot with ¼ tile accuracy
6. The planned path does not have to be optimal.

2.2 Success Criteria

To measure the success of the project, we have developed success targets and quantitative performance measurements. As the system is composed of two separate parts, the planner and the final precision, we have composed a set of success targets that test the parts individually, and in conjuncture. The purpose of the performance metrics is to test the reliability and repeatability of the planner and the system in order to ensure that it performs the desired functions. The targets defined are:

1. Planner:
   - The algorithm runs to convergence for 500+ iterations for 1, 2 and 3 bots
2. Precision System:
   - Measure the accuracy of the final precision node; desired to be within +/- 10 cm
3. Entire System:
   - Maintain a success rate of 70%+ for 3 Autobots

3.0 Technical Implementation

3.1 Introduction

The completion of the Goto-N task requires the creation of several nodes, both on a server and on the Autobot, in order to localize, path plan and execute commands. On a high level, the pipeline can be explained as follows; the localization system localizes the Autobot on the server. The global path planner node then executes the path planning globally and separate for all the Autobots. Once this is completed, the resulting way-commands are sent to individual Autobots, which then execute them using the Goto-N-duckiebot node. The precision node computes the difference between desired and current pose and sends it to the Goto-N-duckiebot node. The Autobot then stops when the difference is less than the given threshold. The purpose of this section is to further elaborate on the system architecture and structure that we implemented.

3.2 Architectural Setup of Project

The architectural setup of the system can be shown in Figure 1 below.

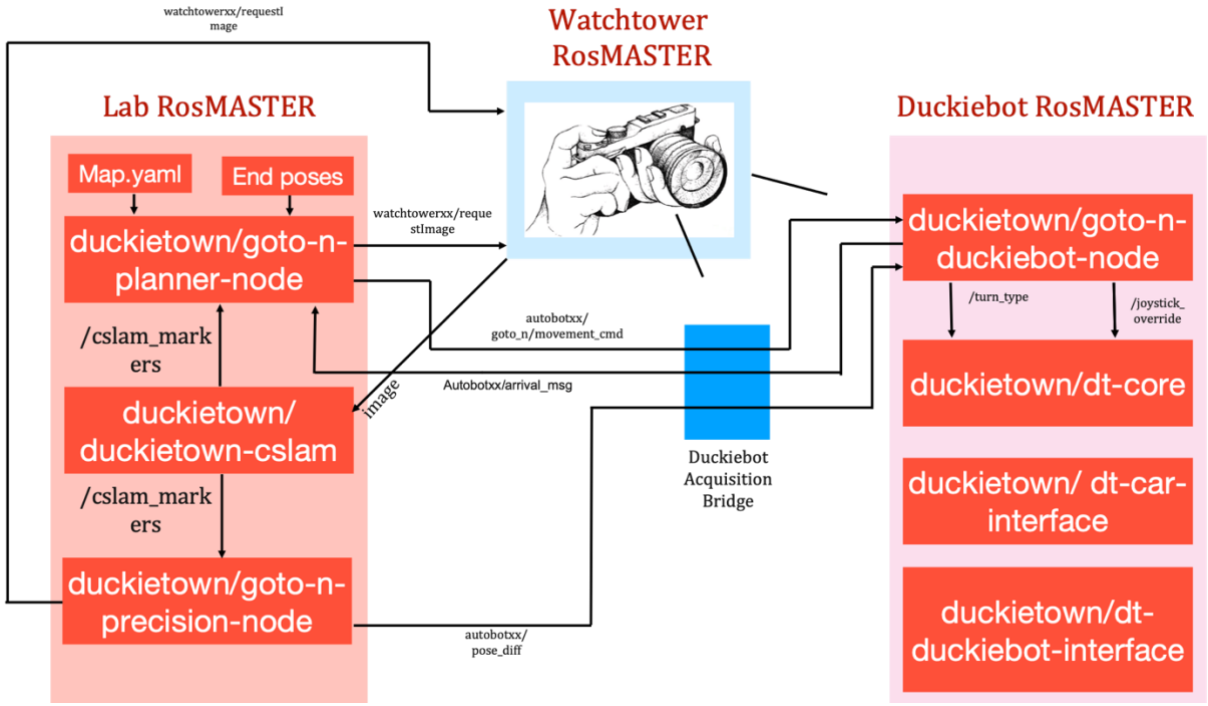

Figure 1: Nodegraph of the Goto-N pipeline

Furthermore, the created packages and functionalities can be summarized below.

Figure 2: Overview of the pythonfiles used

The Goto_N package in figure 2 is equivalent to the Lab RosMaster in Figure 1. The Goto_N package takes in the following parameters:

- Autobots: This yaml file lists all the Autobots present in the map that need to be planned for
- Map: This is a yaml file with all the map data specified according to the conventions set in duckieotwn world
- Termination positions: This is a yaml file that specifies the desired termination positions in global x, y and orientation coordinates
- Watchtowers: This yaml file contains all the watchtower related information for the map

When the Goto-N node is started, it requests watchtower images in order to localize all the Autobots present in the map. The coordinates are given in global x/y coordinates. The Goto-N node then runs the planning algorithm in order to determine the waypoint commands each Autobot needs to undertake to get to the desired termination state. These waypoint commands are then published as a topic, one for each Autobot. The Goto-N-duckiebot node is responsible for subscribing to these topics and executing the waypoint commands.

The waypoint commands are sent to the Autobot in the form of action commands at the respective intersection. The localization and planning algorithms are able to localize the Autobot and determine the next relevant intersection for it. When the waypoint commands are received by the goto-n-duckiebot node, the Autobot begins to run lane-following and intersection navigation. However, we implement a change in the dt-core structure, which allows us to overwrite the actions of the Autobot when it arrives at an intersection. The waypoint override the indefinite navigation commands at the respective intersection.

Once the Autobot has completed its final intersection turn, the goto-n-duckiebot node starts subscribing to the message from the precision node. When the precision node gets executed, the global planner begins to continuously request watchtower images at a rate of 5hz (or at the frequency set in the cslam config). It then calculates the difference between the global position and the desired position of the Autobot and publishes this to a topic subscribed to by the precision node. The precision node continues to execute lane following until the positional difference is within a specified threshold.

Once it has arrived to the desired location, it sets the Boolean arrival_msg to true. If the Autobot has seen a new intersection before finding its final pose, it send arrival_msg to false, and the planner weill replan the route. This is to make the system more robust to uncertainties caused by the bot driving poorly (sometimes takes U-turns or drives a different way in the intersection)

Due to the fact that both the Autobot and the global planner require a RosMaster, we also had to modify the acquisition bridge to be able to communicate between the nodes running on the Autobot and the ones running on the server.

3.2.1 Package Details:

The goto-n node is composed of the files displayed in Figure 2. The functionality is the following:

- **Goto_n.py**: This node runs the global path planner and all associated activities for the goto-n pipeline.
- **Goto_n_compute_accuracy.py:** The final accuracy node is responsible for guiding the Autobot to the desired final termination location.
- **Goto_n_extract_map.py:** This file defines the functions that are necessary for us to extract the given map and decompose it into the node matrices used in the global planner.
- **Goto_n_localize.py:** The functions in this file are used to extract the relevant location coordinates for the Autobot in order to be used in the global planner.
- **Goto_n_planner.py:** The functions in this script execute the global path planner.
- **Goto_n_transition_matrix.py:** These functions define the transition matrix used in the value iteration of the global planner
- **Goto_n_value_iteration.py:** These functions determine the value iteration for the global path planning problem
- **Goto_N_duckiebot.py:** The node running on the Autobot responsible for executing the received waypoint commands at a given intersection.

As mentioned, to setup the communication between the Autobot and lab RosMaster, the acquisition bridge package was altered. The edited files were the *Serversidepublisher.py* and *acquisitionProcess.py* files.

3.3 Global Planner:

The global planner is executed in the Goto-N node and is responsible for the creation of the desired waypoint. The global planner takes in the map information given and creates an alternative node representation of the present tiles by splitting each tile up into four separate nodes and encoding the available potential moves that the Autobot can undertake. Figure 3 below shows the representation of the map:

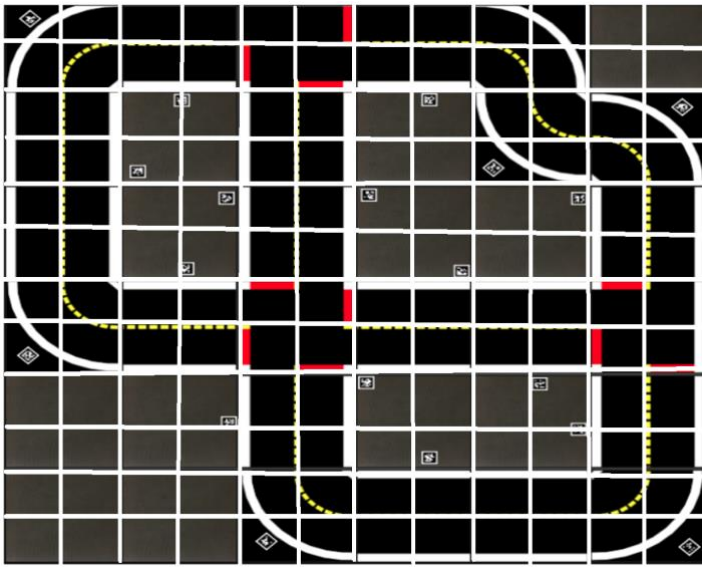

Figure 3: Map divided into 1/4 tile nodes

Ones the tiles are split up into sub-tiles, the planner algorithm encodes the possible actions for each given node. In other words, the planner creates a transition matrix that determines all possible transitions to a new node from a given starting node. This is depicted in figure 4. The different colours define the possible movements; yellow tiles mean the Autobot is only able to move west, green represents movement south, red represents north and finally blue represents east-ward movements. The next step is the creation of a cost matrix that associated a cost with each potential movements. Admissible movements are assigned a cost of 1, while impossible movements have a cost of infinity assigned to it.

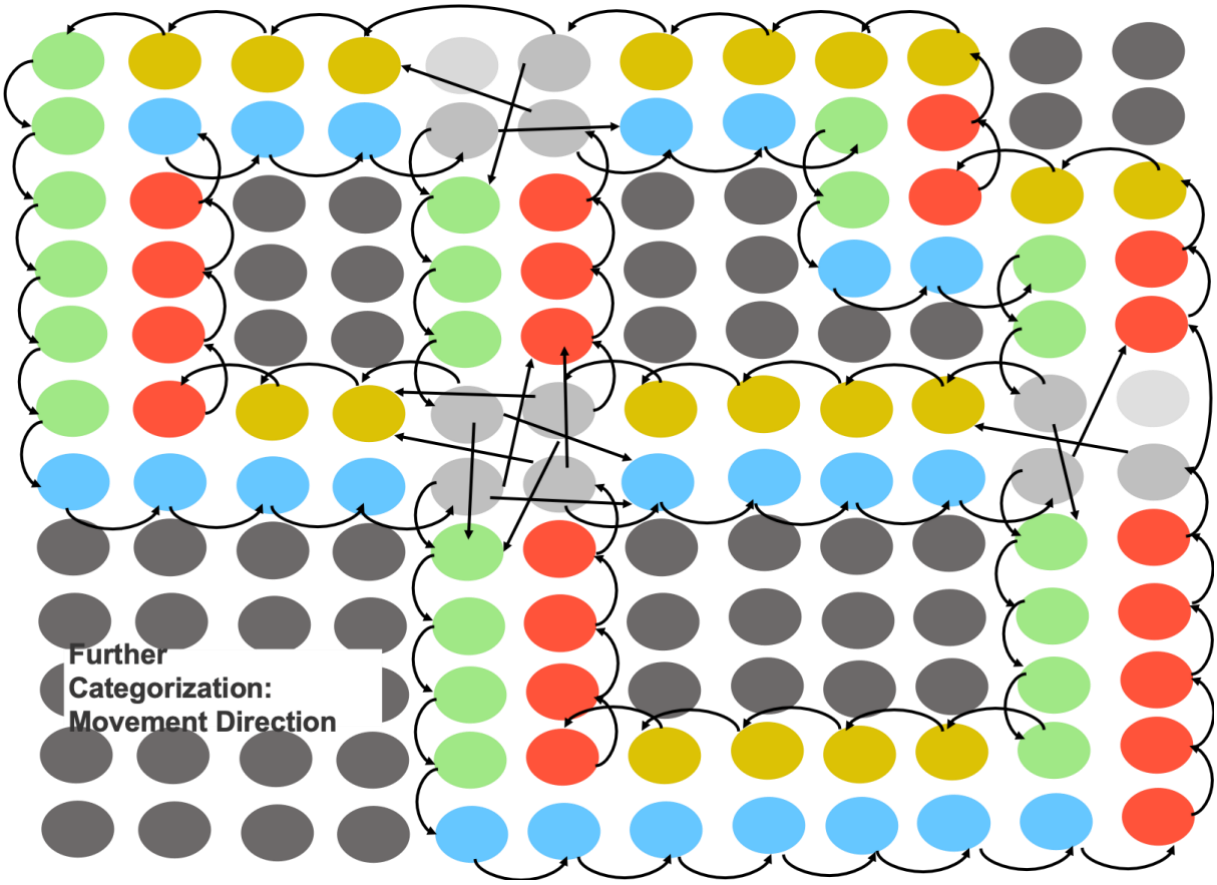

Figure 4: Allowable move from every node in the Autolab at K31

Once the cost and the transition matrix have been created, they are fed into the value iteration process, along with the termination and initialization states. The value iteration finally iterates through all possible combinations and determines the movement commands for the respective Autobots which minimize the total cost, and as such, reduce the total number of necessary movements. Figure 5 gives an indication of the what an optimal movement map looks like for a given termination state (shows the ideal movement to reach the termination from each starting position).

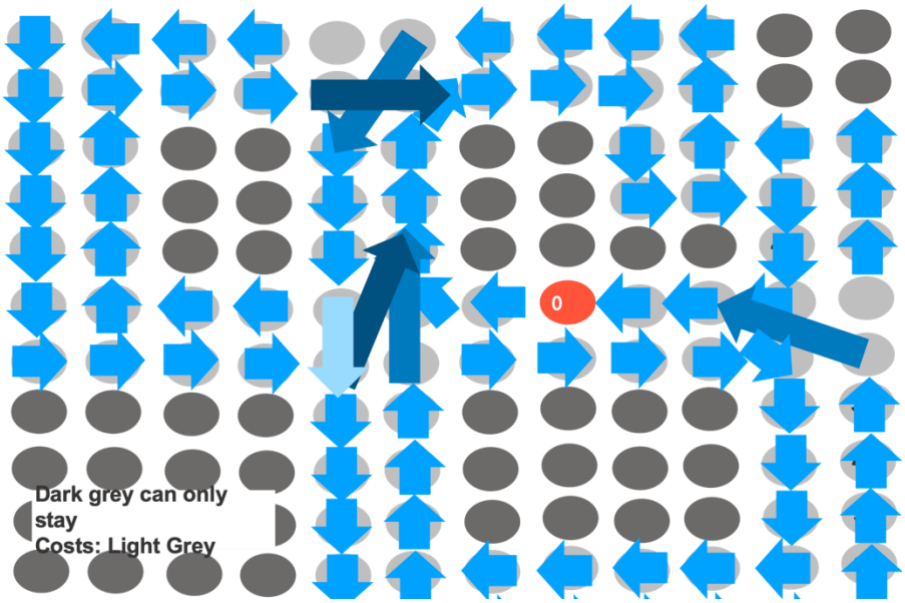

Figure 5: The optimal move for every node, given the termination position (Autolab in k31)

3.4 Precision Node:

The precision node is responsible for guiding the Autobot to the final desired location. Once the Autobot completes the final intersection, the Goto-n-duckiebot node is subscribes to the topic published from the precision node and waits until it is inside a threshold before it stops the indefinite navigation, and sends 0 commands to the wheels and returns an arrival message to the server with true.

3.5 Goto-n-duckiebot Node:

The setup of the system is very easy to follow. The Autobot waits until it receives the first message from the server. It then turns the joystick override command to false, and indefinite navigation starts. It now has waypoint commands stored, and it will perform accordingly. When the Autobot does the last turn it reduces its speed and begins to receive positional differences (delta x and y) between the current and the final location. As long as the positional differences are outside of a specified threshold, the Autobot will continue driving with lane-following. Once the threshold limits are reached, the Autobot receives stopping commands, indicating that the Autobot has reached the final location. The disparity between the final and desired location is then recorded, as in Figure 6, and stored to be analysed in section 4. Data Processing and 5. Data Discussion.

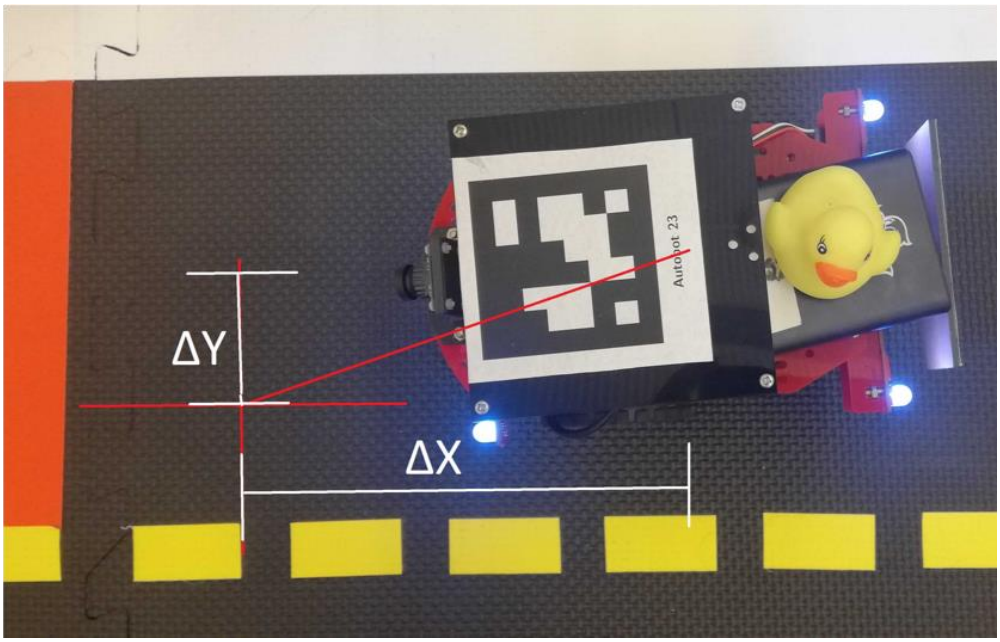

Figure 6: Final precision difference on Autobot

If the Autobot sees a new intersection it means that it has been driving wrong, and the arrival_msg Boolean is turned to false, and then sent to the server. The server then replans for the new position of the Autobot.

4.Data Collection:

Regarding the evaluation of implementations as well as robustness clarification, data were collected in three different approaches. Those approaches simplified the system in the planning component which was only software related, final location and complete pipeline components which were software and hardware related. Accordingly, the software parts of the components were the ones given consideration as hardware issues were mostly unpredictable or temporary.

4.1 Planning Component:

This component is theoretically the only one fully responsible by this project as planning does not rely on other implementations. However, the whole pipeline works if localization provides accurate information to the planning node. Therefore, to remove any reliance on localization before planning and indefinite navigation after planning, this component was only tested with artificial random localizations and neglected what happens after the plan is computed.
To achieve that, the planning algorithm was duplicated on a python file that was disconnected by other components and was receiving random initial and final locations and orientations on the map. Accordingly, this python script was run for three different cases, 1 robot, 2 robots or 3 robots.

4.1.1 Data Structure

The data recorded were:

1. Iteration
2. Sum of number of moves each robot needs in order to reach the termination pose (position and orientation). Those movements are considered to be 1 per tile as presented in chapter 3.3. $N = \sum_{i=1}^{n} N_i$. Where N_i is the number of movements per robot and $n \in \{1,2,3\}$ the number of robots.
3. Starting positions, the initial random position of each robot in (x,y,θ) coordinates
4. Termination positions, the random termination positions required in (x,y,θ) coordinates.
5. Number of Mistakes, a cumulative number, increasing by 1 in every iteration where the number of movements $> 2 \times K_X \times 2 \times K_Y$. Where K_i is the number of tiles the map has in the i direction. The reason why each direction is multiplied by 2, is the fact that the planning algorithm splits every tile in 2. Moreover, that number was chosen since in that case, the robots covered the whole map which means that the plan was wrong. Accordingly, if 1 termination state is unreachable or the robot cannot leave the initial tile, then the cost is initiated to infinity. However, such cases are treaded by the algorithm as exceptions and return messages that the robot is outside of the road.

4.1.2 First Case- 1 Robot:

For this case, 1 random initial pose (position and orientation) and 1 random final pose were given to the planner which was responsible for the computation of an optimal trajectory which the robot should follow. Since the planning for 1 robot is simpler that other cases, it requires less time and thus it was possible to run the planning algorithm great amount of times. Therefore, the data for this case were obtained for 1000 iterations. In those iterations the information obtained is:

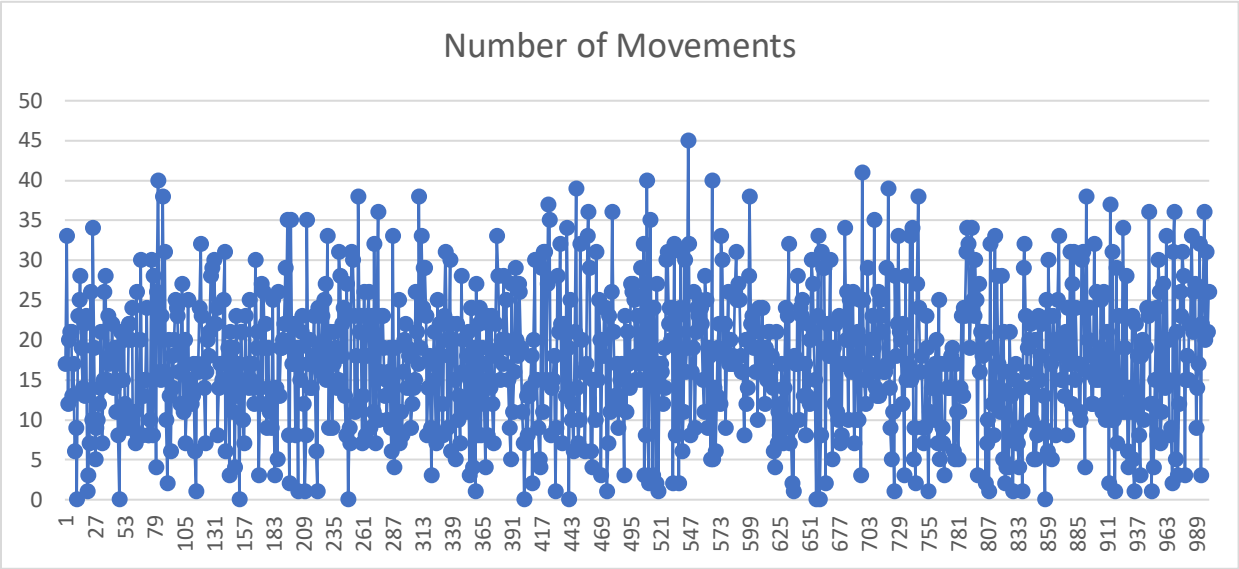

Figure 7: Number of movements for each try of randomly placed termination and start pose.

Average Number of Movements: 17.374
Standard Deviation: 8.6776796
Number of mistakes: 0

4.1.3 Second Case- 2 Robots:

For this case, 2 random initial poses were given to the planner, and 2 termination poses. Each termination pose could be obtained by any of the robots but each robot can obtain 1 termination pose. The algorithm gave:

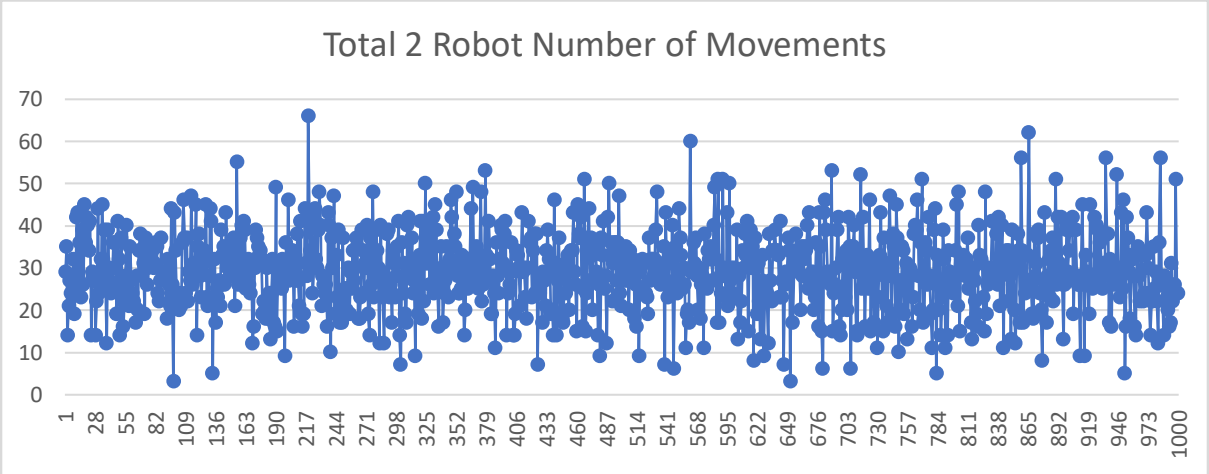

Figure 8: Total number of movements for each try of randomly placed termination and start pose for two robots.

Average Number of Movements for 2 Robots: 28.03656
Standard Deviation: 9.940511
Average number of Movements per robot: 14.01827957
Standard Deviation per robot: 5.073678231
Number of Mistakes: 0

4.1.4 Third Case- 3 Robots:

For this case 3 different initial positions where given as well as 3 different termination positions. The computation time for 3 robots and possible combinations between termination positions takes longer than in the previous cases and therefore the iteration were reduced to 400.

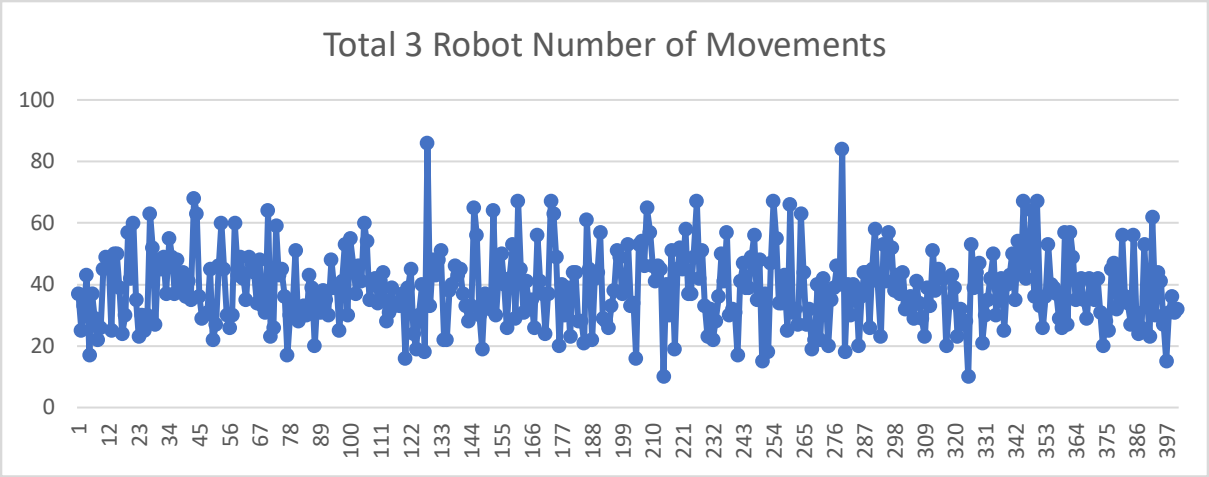

Figure 9: Total number of movements for each try of randomly placed termination and start pose for three robots.

Average Number of Movements for 3 Robots: 38.9225
Standard Deviation: 12.1246416
Average number of Movements per robot: 12.97416667
Standard Deviation per robot: 4.041547201
Number of Mistakes: 0

4.1.5 Comparison Between Number of Robots:

As a comparison between the 3 cases, the average total movements and standard deviation are shown on the first Figure 10. Figure 11 shows the average movements per robot and standard deviation per robot.

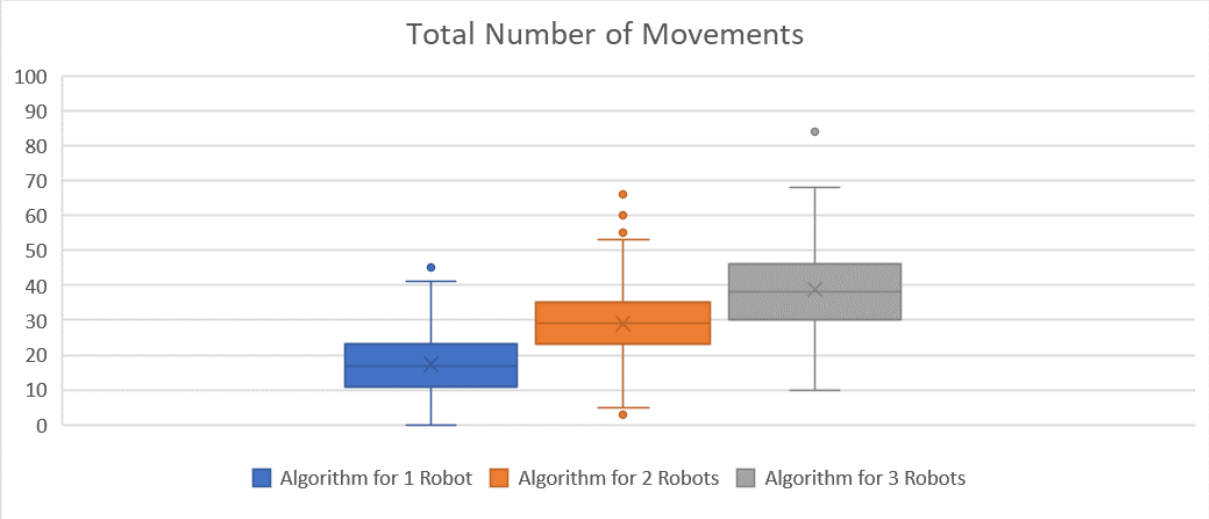

Figure 10: Average Total number of movements vs bots used

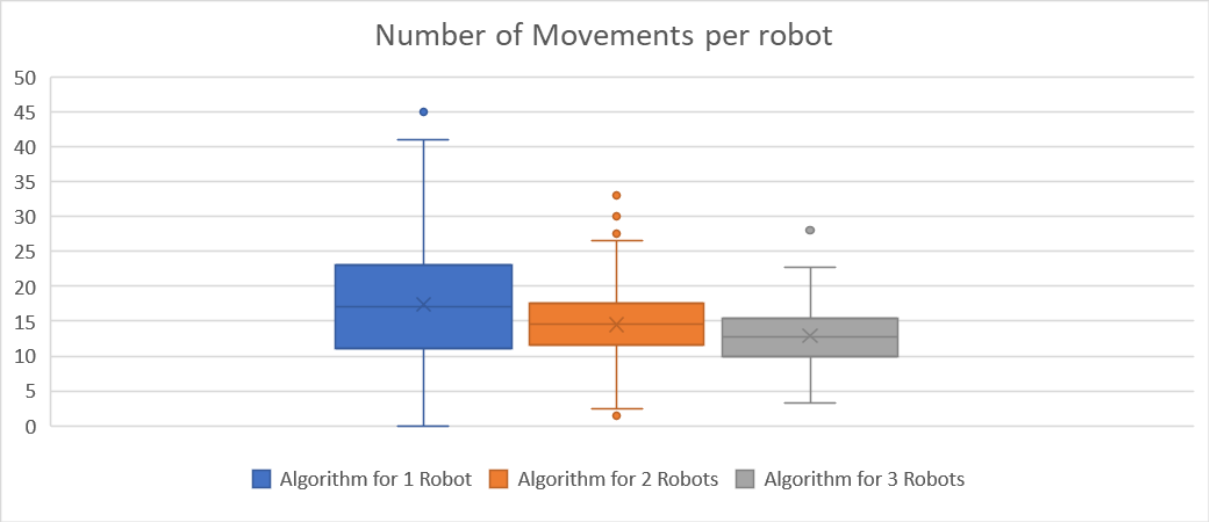

Figure 11: Number of movements per bot vs bots used

4.1.6 Constraints:

This implementation was only tested for the map in room ML k31. Other problems may occur when testing on other maps.

4.2 Final Location Accuracy Component:

The final accuracy was measured on each robot that successfully reached the correct tile. This value was given as the Euclidean Distance between the requested position and actual final position of the robot. This distance was then divided by the tile length to give an answer in fraction form. Thus, the accuracy is defined as:

$$Accuracy = \sqrt{(\frac{\delta X}{T_X})^2 + (\frac{\delta Y}{T_Y})^2}$$

Where T_X, T_y is the tile length in the X and Y axes respectively. The values δX and δY are the differences between the requested and the actual termination positions in X and Y axes respectively. As a disclaimer, since the angle between the requested and actual position is not relevant to the algorithm but mostly on the success of lane following, the angle difference is neglected. Moreover, the angle difference will not affect the performance of initialization since lane following sets the car to the direction of the tile as it will always be used. Accordingly, this metric is related to both localization system and planning as well as indefinite navigation. However, in cases where indefinite navigation was not working as expected and was performing poorly in intersections, the measurement would not count as unsuccessful for this metric and was just neglected. Therefore, the presented measurements regard only the cases where the robot followed the planning given to it and reached the state where it should stop.

The data were measured for 50 tries. Those tries involved 5 different termination positions and were further split between the amount of turns the robot needed to follow before it reaches the final tile.

Table 1: Precision for 0, 1 and 2 turns

Number of Turns	δX [m]	δY[m]	$\frac{\delta X}{T_X}$ [%]	$\frac{\delta Y}{T_Y}$[%]	$Accuracy$ [m]	$Accuracy$ [%]
0	0.047	0.043	8.037	7.379	0.0638	10.910
1	0.079	0.033	13.462	5.662	0.0854	14.604
2	0.072	0.054	12.332	9.219	0.09	15.397

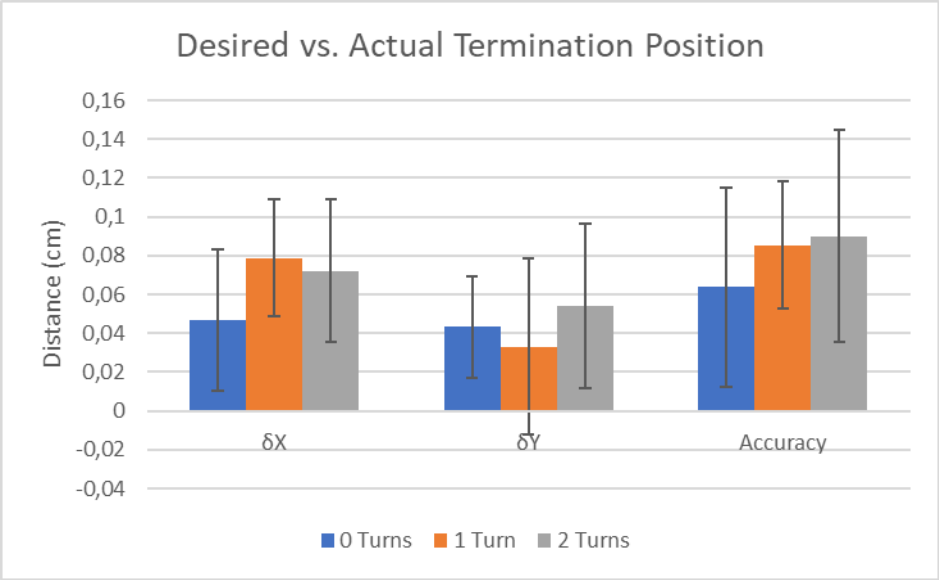

Figure 12: Desired vs actual termination position for 0, 1 and 2 turns.

4.3 Complete Pipeline Success Rate Component:

This final component is consisted by the whole pipeline, which means localizing, planning, sending messages to the robots, execution by robots and reaching the correct tile. As shown by previous components, the parts mostly relying on the scope of the project were considered alone. However, in this case other already existing parts were also used such as localization and indefinite navigation. Therefore, given the random or uncontrollable components of this metric, this is the one reflecting the applicability of the project in current systems but not the quality of work which was given by the previous metrics. Accordingly, given the extended time required for each try, the number of data are small and split in three subcategories, 1 robot, 2 robots and 3 robots.

The data were consisted by 15 tries with each configuration 1,2 or 3 robots. Moreover, the trial would stop if robots would lose the initial trajectory and require re-plan. Although the re-planning function implemented on the algorithm would make some of those tries successful, the required time does not help the extraction of more data. Especially in the case were the robots are 3, the re-plan and retrying function would be too time consuming, highlighting the need of better intersection navigation functionality. The data takes show:

- 1 Robot: 12/15 Successful trials, thus success rate of 80%
- 2 Robots: 8/15 Successful trials, thus success rate of 53.3%
- 3 Robots: 3/15 Successful trials, thus success rate of 13.3%

5.Data Analysis:

For the analysis of the shown data, each component was treated individually. However, the general conclusions from all components are:

- The parts related to the work of this project appear to be robust and well designed.
- The whole pipeline is not robust for increased robots but mostly given uncontrolled parameters from this project rather than implementation mistakes. Therefore, improvements in areas such as indefinite navigation and intersection navigation, could help improve the robustness of the entire system
- The goal of this project was achieved in high degree as the planning and final accuracy are very good and the pipeline design works.

5.1 Planning Component:

- This component has shown that the planning appears to work very well for all cases (1,2 and 3 robots).
- The average number of moves per robot is relatively good in all cases, since each map tile is split in 2 parts which means that approximately every robot has to move about 8 tiles for random initial and termination positions.
- The fact that the average movements per robot get reduced as robots increase is supporting the quality of planning, since it shows that if there are more choices for termination positions and combinations where each robot will end up, the algorithm will improve the choice and choose the best configuration to reduce the movements. This is achieved by checking the closest termination to each robot in every possible order and choosing the one with the lowest total movements.
- The zero amount of mistakes supports the robustness of the planning algorithm.
- The standard deviation of moves per robot is reduced as robots increased because one random initial and termination positions require only one plan which the robot has to follow no matter how close or far it is. However, for multiple robots and terminations the planning searches the best configuration which reduces the moves making it less prone to outliers requiring 1 robot to go to the other side of the map.
- The standard deviation of total moves for all robots is increased as robots and terminations increase as there are more combinations for those poses.
- The expectations and goals for this part are very successful as data show zero errors for tries for up to 3 robots.

5.2 Final Location Accuracy Component:

- The average final location distance is less than 10 cm, thus it is very accurate and achieves the goal.
- That average distance is within the acceptable and expected limits since it is less than the robot length and it is measured along the centre of the car.
- There is no direct relationship between number of turns before the final tile and final accuracy as expected.
- Orientational data was not collected as we assumed that the robot would be restarted by running lane following, rendering small orientational changes mute.

5.3 Complete Pipeline Success Rate Component:

- As the number of Autobot is increasing the success rate decreases.

- There were issues with the performance of indefinite navigation and intersection navigation which resulted in a success rate of 13% for three Autobot scenarios

6.Conclusion & Future Work:

6.1 Conclusion:

As discussed in section 5, we were able to reach two out of the three previously determined success criteria. The accuracy node was able to complete the desired +/- 10cm Euclidean Distance and proved that the implemented system could satisfy the requirements of this project. Furthermore, the planner also performed as hoped. It was able to converge for 400+ iterations for one, two and three bots respectively without failure, and this proves the planner's ability to plan accordingly for n number of bots (until 3). Finally, the success criteria for the entire pipeline was not achieved. As mentioned, only the one bot instance was able to complete the desired 70%+ success rate. Cases with two and three bots failed to achieve our desired benchmark and produced scores of 53.3% and 13.3% respectively. The reason for these disappointing scores was discussed in the previous sections and have to do with external issues affecting the robots ability to follow the lane and correctly navigate intersections without crashing. The project allows for some avenues of future work that will improve the overall performance of the system. Even though the success rate drops drastically as more robots are introduced, following up with the below mentioned avenues of future work would help improve the performance of the entire system.

6.2 Future Work:

While there are numerous avenues for future work, we would like to highlight the following three as the most important.

1. **Improving Intersection Navigation and April Tag Detection:**
   A common cause for failure in the testing scenarios was the poor performance of the intersection navigation and April Tag Detection. To be able to achieve the desired termination position, the Autobot has to execute the desired waypoint commands at a given intersection. However, we have found during testing, that the Intersection Navigation of the individual Autobots did not always result in the desired movements. In many instances, the Autobot was given instructions to turn left and would proceed to drive right/straight instead at the intersection (with the danger of additional crashes).
   Furthermore, April Tag Detection at the intersections also proved to be a challenge. The Autobot executes our commands once it sees the corresponding intersection. However, during testing, we had numerous instances where the Autobot would arrive at an intersection, read the required April tag, perform the turn. While performing the turn, it would detect an unrelated April tag (from the same intersection, stop sign, etc.) and interpret this as the next intersection. Consequently, it would execute the command too early.
   We have found the above to be two major issues contributing to the failure of the three bot scenarios. We also found that the more bots were in operation, the more likely one of the bots was to fall victim to the problems above, the higher the chances for a failed test. As a result, with the improvement of these two functionalities, it could be possible to improve the performance of the system further.

2. **Implementing a more sophisticated closed loop controller for the accuracy node:**
   The second area of future work relates to the final accuracy node of the system. Although the accuracy node was able to meet the requirements we set before the project, it is still a very basic system that is prone to error. For instance, it is unable to deal with situations in which the Autobot drives past the required location, or in cases where it is close but not exactly at the required position. Therefore, it could be interesting to develop and implement a closed-loop controller that is able to navigate the Autobot to the desired position more accurately. In addition, it would be beneficial if the controller could deal with instances where the Autobot drives too far and cases in which it fine tunes the final location exactly.
3. **Integration into Duckietown World:**
   The final avenue of future work could be the integration of our system with Duckietown world. Although our path planner is currently able to plan the path for an arbitrary map configuration, it does not follow the same approach as Duckietown world. Consequently, it could be useful to integrate our system into Duckietown world to increase the cohesiveness and potentially functionality of the system and the platform.