Autonomous Navigation Visualizations and Interface

Yuxi Luo
Halıcıoğlu Data Science Institute
University of California, San Diego La Jolla, CA, 92093
yul884@ucsd.edu

Seokmin Hong
Halıcıoğlu Data Science Institute
University of California, San Diego La Jolla, CA, 92093
sah073@ucsd.edu

Jia Shi
Halıcıoğlu Data Science Institute
University of California, San Diego La Jolla, CA, 92093
jis283@ucsd.edu

Abstract

Autonomous navigation requires a wide-range of engineering expertise and a well-developed technological architecture in order to operate. The focus of this project and report is to illustrate the significance of data visualizations and an interactive interface with regards to autonomous navigation in a racing environment. In order to yield the best results in an autonomous navigation race, the users must be able to understand the behavior of the vehicle when training navigation models and during the live race. In order to address these concerns, teams working on autonomous navigation must be able to visualize and interact with the robot. In this report, different algorithms such as A* search and RRT* (Rapidly-exploring random tree) are implemented to create path planning and obstacle avoidance. Visualizations of these respective algorithms and a user interface to send/receive commands will help to enhance model testing, debug unexpected behavior, and improve upon existing autonomous navigation models. Simulations with the most optimal navigation algorithm will also be run to demonstrate the functionality of the interactive interface. The results, implications of the interface, and further improvements will be discussed in the following sections.

I. Introduction

An important aspect of path planning and obstacle avoidance with regards to autonomous driving is efficient pathing. In order to create the most efficient path, data must be fed as an input in order to derive the best possible output. Visualizing this output and interacting with the robot will help to identify the most desirable path and help the vehicle avoid obstacles to maneuver from point A to point B. This concept can be applied to any moving robot as it will have to avoid obstacles in order to arrive at the desired destination. Creating an interactive interface platform that allows the user to view the vehicle's current path and navigation sensor information will further help the vehicle efficiently navigate autonomously. In the following section, methodology for developing an interface and efficient path navigation will be described in further detail. Following the methods section, this report will describe the results and impact of allowing the user to view real-time vehicle information while having the ability to control the vehicle.

II. Methods

Due to the current state of the pandemic, all autonomous navigation racing platforms have been pushed back indefinitely. In order to compensate for the lack of off-line races, simulated racing tracks have been created with the help of students from our domain. To continuously improve and test the efficiency of our autonomous navigation while avoiding the risks of the pandemic, most components of this methodology will utilize the Gazebo simulator [1], which will be the platform that supports the simulated online racing tracks. The Gazebo simulator is primarily used in conjunction with the Linux operating system, and as a result, Linux OS is the desired operating system due to increased compatibility. Other important components of this methodology will include OpenCV library, Gazebo Simulator, RViz, Rosbridge, and many ROS packages.

A frequent bottleneck that may arise is the lack of compatibility between some Ubuntu Linux versions with some ROS launch files provided. It must be noted that Ubuntu version 16.04 and version 18.04 would work better with version ROS Kinetic and ROS Melodic respectively. If a different version of ROS were to be installed with another version of Ubuntu, errors are likely to arise frequently. With this information in mind, it should be noted that the methodology presented will be developed in Ubuntu version 16.04 and ROS Kinetic.

The primary step taken was to ensure that each hardware and software components were working

properly to reflect real-life autonomous navigation. Data collection and processing was completed using the TurtleBot robot [2], a personal robot kit with open-source software that supports ROS and more importantly, ROS Navigation packages. Instead of physically using these TurtleBot robots in real-life, they were virtually spawned using the Gazebo simulator, which created a workspace to test different algorithms on the TurtleBot robots in different environments and maps. The TurtleBot uses the Intel Realsense R200 camera and an LDS-0 360 Laser Distance Sensor as its default hardware sensors. Because the robot built for real-life autonomous navigation uses the Realsense D455 camera and a SICK Lidar, the configurations for the TurtleBot's default camera were changed to the Realsense D435 camera and the default lidar to the Hokuyo Lidar. The objective in these changes is to help decrease the hardware discrepancies between the Gazebo simulations and our real-life robot.

Once the TurtleBot was properly configured, our group was able to spawn the UCSD racing track [3] on the Gazebo simulator, depicted in **Figure 1**. This racing track accurately reconstructs the off-line track that our domain used to assess different path planning algorithms before pandemic restrictions were strengthened. After this setup was completed, our group was able to perform G-Mapping using the TurtleBot.

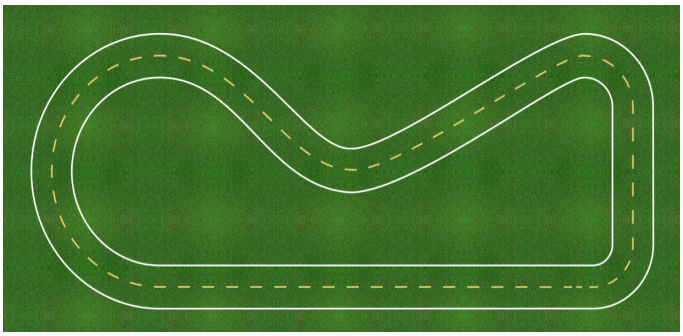

Figure 1: UCSD Racing Track on Gazebo Simulator

G-Mapping [4] uses ROS Navigation and ROS Perception packages to provide a laser-based 2-D occupancy grid map from laser and position data collected by the robot. To create the grid map using G-Mapping on the Gazebo simulator, our group launched multiple ROS nodes using "roslaunch." After launching into the UCSD race track map using Gazebo, we used a keyboard to navigate through the map while the lidar saves its LaserScan data into rostopics along with positional data. After scanning through the map, our group was able to save this scanned map into a ".yaml" file which consists of the map's meta-data, depicted in **Figure 2**.

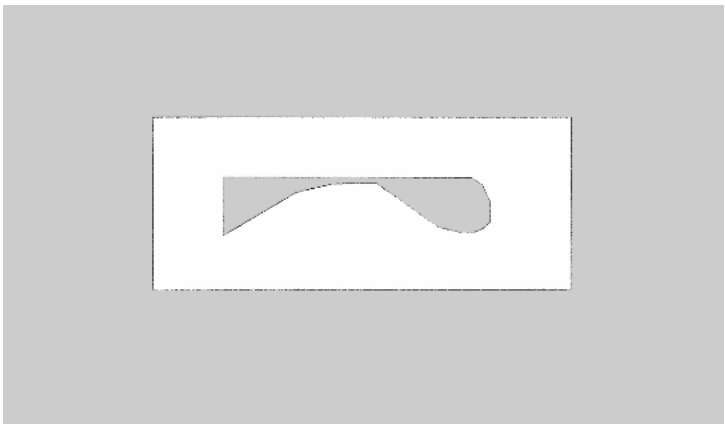

Figure 2: Visualization of ".yaml" file map meta-data

Using the 2-D occupancy grid map created using G-Mapping, different path planning algorithms could be implemented to help the TurtleBot navigate throughout the race track. Our group implemented two path-planning algorithms, the first algorithm is a search-based algorithm called A*. The A* algorithm is a heuristic search to find the shortest path in the least number of computations. Given a starting and end point, the algorithm is tailored to explore paths only in the direction of the goal. To navigate in the shortest path, priority is given to the nodes that have a lower estimated distance to the end point, which is calculated using the euclidean distance. The second algorithm our group implemented is a sampling-based algorithm called RRT*. The RRT* algorithm creates a path by building a tree from the starting position of the robot. Different points are sampled from the initial position, and they are checked for any collisions with obstacles. If the point does not cause collisions, it is added to the tree with the nearest point as its parent node. This process repeats until a path is found from the initial to the destination point.

Using the grid map generated, these theoretical algorithms generate a /move_base/goal rostopic that moves the robot from a starting point to an initial point. This process can be computed and visualized in RViz [5], a robot visualizer ROS package. Rviz allows the

move_base rostopic generated to be visualized inside its own interface. Using ROS nodes, Rviz allows the user to input a "goal" for the vehicle to autonomously navigate using the 2D Nav Goal button, depicted in **Figure 3**. When the user uses the goal button to set a destination point, the robot's current location will be the starting point. Once the robot begins moving towards its given destination, a local and global planner could be seen that visualizes the robot's current pathway and final pathway. The local planner (green line) displays the current pathway of the robot, and the global planner (blue line) displays the calculated path generated from the algorithms that the local planner will eventually follow. A demonstration of the RViz interface can be found in the hyperlink below:

RViz Navigation

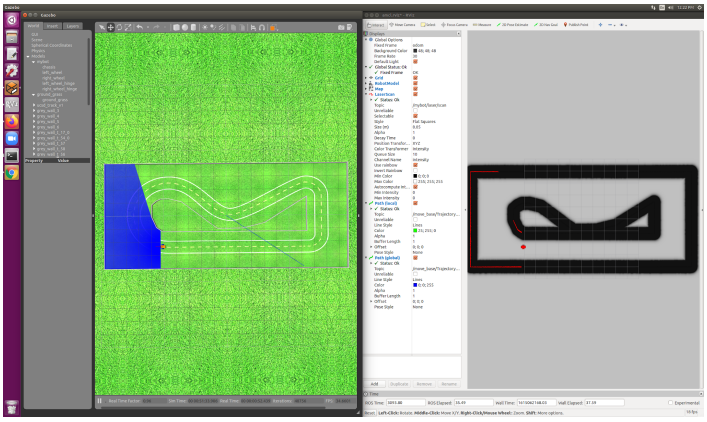

Figure 3: RViz 2D grid map visualizing G-Mapping

RRT* and A* algorithms are two common algorithms used in the field or robotics for search and navigation, with the preference skewed towards the former. Our group was able to implement these two navigation algorithms using G-Mapping, and now our group will try to compare these two algorithms based on different metrics that will determine which algorithm is more efficient in autonomous navigation.

For precise and direct comparisons, each respective algorithm was implemented using the Python programming language [6], where they will be run and tested against each other to evaluate performance. Instead of running the navigation algorithms in a Gazebo or ROS based simulation, each respective navigation algorithm was run using binary grayscale mapping or track images. In order to evaluate the speed and robustness of each of these algorithms, each algorithm was tested on a base maze image that is labeled "maze.png," depicted in **Figure 4**.

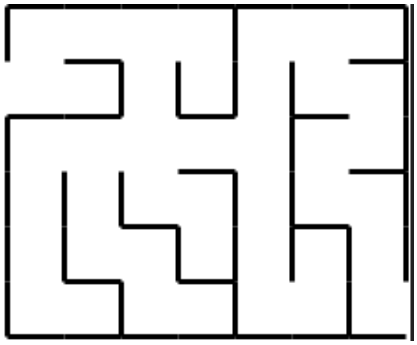

Figure 4: Base image referring to "maze.png"

In **Figure 5**, the RRT* algorithm is run on the base mapping image of "maze.png". The red nodes refer to each node that is created and branched out of the tree starting from the starting point. The green lines represent the paths connecting each node that has been branched out since the original starting point. Once the nodes continue to branch, they will iteratively continue through the map in search of connecting to the destination node. Once the nodes can be connected through the RRT* branching method, the blue line represents the best path created in the tree based iterative node connecting process seen in RRT* algorithm. In **Figure 6**, the A* algorithm is run on the base mapping image as well. It should be noted that this image is inverted when computing for the best path. When loading in the data with CV2 [7] functions, there were default settings that inevitably inverted the image; this leaves room for improvement in the future to fix the inverted image effects of CV2. The yellow aspects in **Figure 6** shows the graph traversal process that the A* algorithm is performing to ultimately connect the starting node to the destination node. The red line in the respective figure represents the best path that is created after the traversal process has been completed.

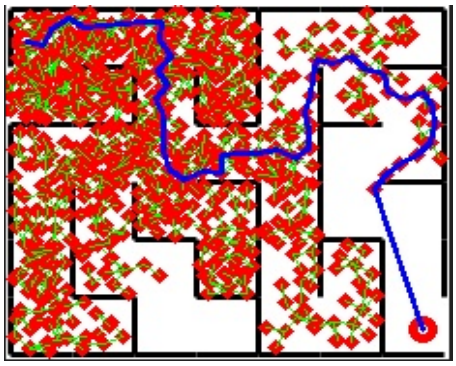

Figure 5: RRT* algorithm path creation on “maze.png”

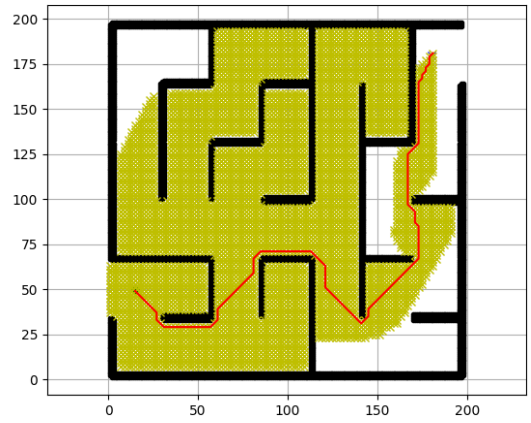

Figure 6: A* algorithm path creation on “maze.png”

Both RRT* algorithm and A* algorithms were run 10 times each to determine the average runtime of finding its best respective path. The results can be found in the following table below referenced as **Figure 7**. As seen in the results, the metrics ultimately decided on for determining the algorithm robustness and performance were average completion time in seconds given 10 iterations and number of algorithm failures throughout all 10 iterations. Intuitively, it makes sense to yield a lower completion time to find the best path as that implies that the algorithm will find the best path faster.

Additionally, the fewer failures with the algorithm in a given number of runs will mean that the algorithm is more robust and is less likely to run into failures. As can be seen in **Figure 7**, RRT* algorithm yields better results in both of these metric categories. After realizing these results, it was determine that RRT* is the better algorithm and was used for further testing in navigating the Thunderhill mapped track through simulations.

Further visualizations and the integration process of such visualizations using RRT* algorithm will be discussed further in the following results section.

	RRT*	A*
Average completion time in seconds (s)	143.707	215.229
# of node connection failures in 10 runs	1	2

Figure 7: Metrics for algorithm performance

In this report, our group’s main objective is to test different path planning algorithms that would be efficient for racing in different environments. To fully incorporate this idea, we must be able to visualize what path the vehicle is currently taking, and determine what algorithm is best suited for each different racetrack. In order to make these kinds of decisions, we need to implement a user-oriented interface, that will allow the user to monitor the efficient paths that these different algorithms generate via visualization, and allow the user to control the robot through this interface.

To bring this theoretical process into action, we implemented an interactive interface using Rosbridge [8], which is a package that provides a JSON API that will implement ROS functionalities to programs (web-browser) that does not normally process ROS-related programs. Using HTML [9] and Javascript [10], we will create a web-browser that will connect to the Rosbridge servers to allow interactive usage for users, depicted in **Figure 8.** This interactive interface can subscribe to rostopics that will take input data from the robot’s different sensors, as well as allow the user to input specific destination coordinates to start autonomous navigation using the different path planning algorithms. The interface will also include the python visualized navigation algorithms that will show the calculated path plans in real-time on the webpage. The web-browser will allow the user to monitor the vehicle’s status, similar to the dashboard of a car.

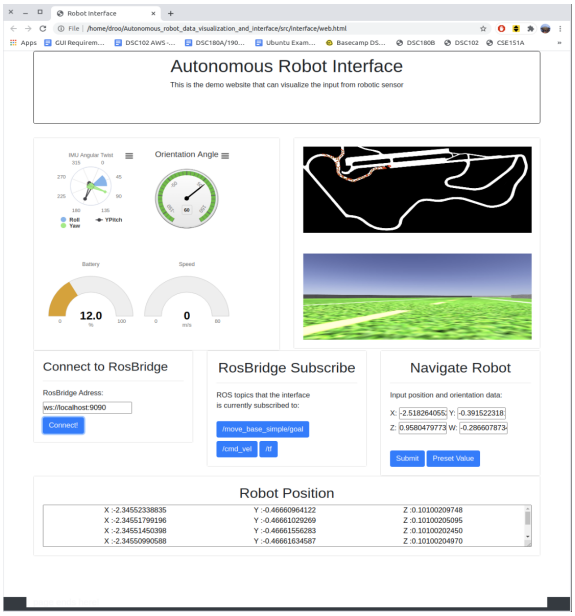

Figure 8: Interactive interface using Rosbridge

III. Results

Building an interactive interface is crucial, because it allows the user to monitor the vehicle's current path as well as sensory information that will be useful for determining if the autonomous navigation algorithm chosen is efficiently transporting the robot from one location to another. As mentioned previously in the methods section, our group implemented RViz, which was able to visualize the vehicle autonomously navigating with the help of the 2-D grid map created from G-Mapping and the path planning algorithms such as A* and RRT*.

Although RViz is visually appealing and shows the local and global planners that allow the user to view what the robot's current path is using the path planning algorithms, it lacks support in visualizing sensory information such as the vehicle's current position, speed, odometry, IMU, and even battery life. Also, RViz may be difficult for users to navigate around, because it requires understanding certain rostopics to view certain data such as images from the Realsense depth camera. The user would have to add different rostopics with several clicks inside RViz, which may cause confusion.

In order to allow the user to effortlessly understand what path planning algorithms are being utilized and also be able to view different sensory information that the autonomous vehicle is outputting, the interactive interface using Rosbridge was created. The interactive interface implemented will also allow the user to move the vehicle from its initial position to the final destination by pressing the "Submit" button, as long as the user knows the final position coordinates. In case the user does not know the final position coordinates, the interface will provide the track's finish line coordinates, which the user can obtain by pressing "Preset Value" that is installed in the interface to autonomously navigate the vehicle. A demonstration of the Interactive interface can be found in the hyperlinks below:

Interactive Interface (Preset Value)

Interactive Interface (Input Value)

Our group was able to utilize ROS node commands such as rostopic echo and rostopic info to understand how different navigation sensors could be displayed in the interface, depicted in **Figure 9**. For example, we used the above commands to understand that the /move_base/goal topic generated from the autonomous path planning algorithms could directly move the vehicle from its initial position to the final position. Using HTML and Javascript, our group was able to create the interface without the user having to search through ROS nodes to receive sensory information outputted by the vehicle. By creating a web-browser that allows the user to control and view the robot in real-time, the user will understand how the vehicle is thinking, and if the path planning algorithm selected is efficient in the different race track characteristics.

```
droo@droo: ~
droo@droo:~$ rostopic info /cmd_vel
Type: geometry_msgs/Twist

Publishers: None

Subscribers:
 * /gazebo (http://droo:42771/)


droo@droo:~$ rostopic echo /cmd_vel
linear:
  x: 0.216315789474
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -1.0
---
linear:
  x: 0.216315789474
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -1.0
---
```

Figure 9: Rostopic info and echo demonstration

As mentioned previously in the methods section, RRT* algorithm was determined to be the better navigation algorithm for finding the most efficient path given the start and end points on a binary grayscale image. It must also be noted that previously RRT* was tested on a binary masked grayscale image that resembled the makeup of a maze. Nevertheless, the main application in which RRT* is being applied to is a race. The specific race being catered to is an autonomous navigation race at the Thunderhill track. Therefore, our tuned RRT* algorithm must be tested against track like settings, and once deemed satisfactory, the RRT* algorithm will be tested against a masked grayscale image of the real two mile Thunderhill track [11].

A test_track.PNG is generated, which can be seen in the outline in **Figure 9.** This track illustrates more curvatures and changes in outline that is different from that of **Figure 4**. These changes in a mapping setting would represent different challenges for the search and navigation algorithm as it would have to find the most optimal path. In **Figure 9**, it can be seen that the RRT* algorithm has found its most optimal path from one designated point to the other.

Although the blue line representing the best path may seem jagged and seem to run off course, the line is the best possible path because of potential delay of computing power associated with the necessary calculations to find the best path. It also occurs that the node creation process branches out to find the best direction, and coincidentally, the path seems to be jagged. This RRT* algorithm also runs relatively fast and as a result of the node branching out process, the jagged behavior in the path creation occurs. It should be noted that that if RRT* runs through longer iterations, the path will curve out to be more smooth and less jagged behavior will be seen. Nevertheless, the path outputted is still the best path for RRT* to navigate from the start point to the end point.

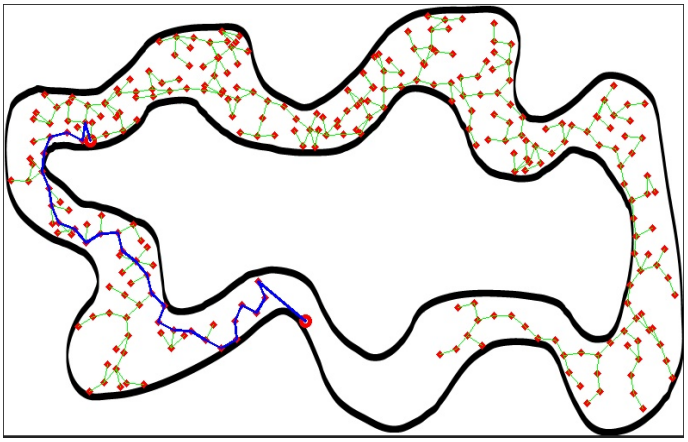

Figure 9: Python visualization of RRT* algorithm on "test_track.PNG"

After confirming that RRT* works on a map-like binary grayscale setting, the base image for the Thunderhill track was created from the Thunderhill track website. As seen in **Figure 10**, the image represents the basic layout of the two mile Thunderhill track. This image or figure may also be referred to as "thunderhill_cropped.PNG". As previously mentioned, the track data was taken from the Thunderhill website and converted into a binary masked grayscale image, which is the output seen in **Figure 10**. In this respective figure, the starting line had also been masked and can be seen as a black bar on the track on the top center-left corner of the image. After this masked grayscale image of the Thunderhill track is successfully created, the RRT* algorithm can be applied to this masked environment.

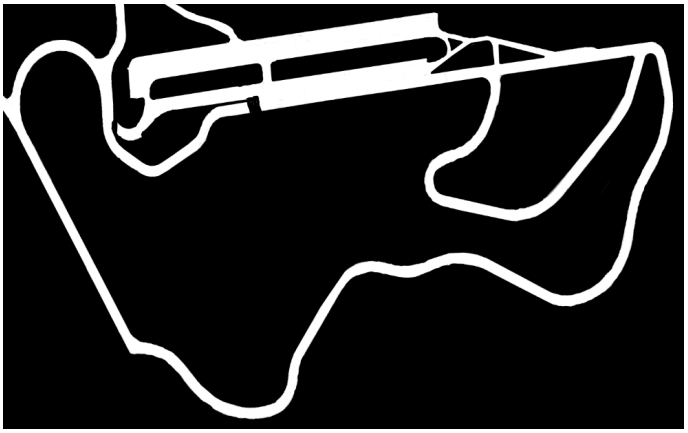

Figure 10: Binary masked grayscale image of the two mile Thunderhill track ("thunderhill_cropped.PNG")

In **Figure 11**, the entire Python bird's eye point of view visualization of the RRT* algorithm on the "thunderhill_cropped.PNG" is visualized. Similar to previous figures, the red dots refer to the nodes of the tree branches that traverse from the original node (starting point) to the destination node (end point). The green lines refer to the connections between nodes, and the blue line represents the final best path that connects the original node to the destination node. This line ultimately represents the path that the autonomous vehicle will follow on the Thunderhill track.

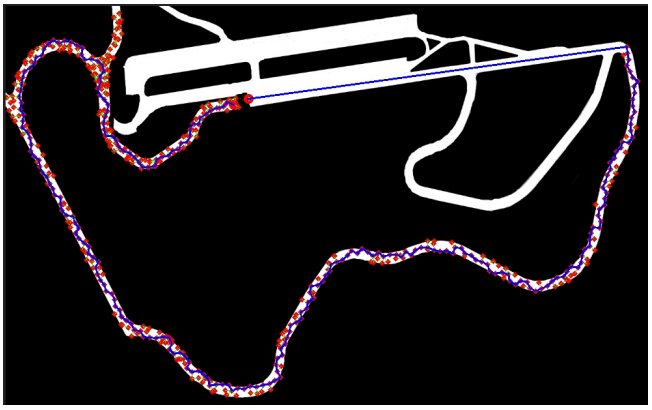

Figure 11: Python visualization of RRT* algorithm on "thunderhill_cropped.PNG".

A demonstration of the python visualizer can be found in the hyperlink below:

Python Visualizer

This RRT* visualization is intended to merge with the interactive interface. As a result, the visualizations that are outputted will serve as a useful tool for the user to identify how the autonomous vehicle is deciding its path and how it eventually navigates. Watching the node creations will inform the user of the thought process of the navigation, and thus inform the user how the car will behave. Following the blue line will allow the user to visualize and understand how the car moves and decide if the path currently navigating is correct according to human intuitions. By producing a real-time image of how the algorithm is creating the most efficient pathway, it allows the user to better understand the entire autonomous navigation behavior process.

The significance of these visualizations and the interface is that the user will be able to monitor the autonomous vehicle and make decisions accordingly. Racing performance and debugging capabilities can be significantly enhanced as a result. For instance, if RRT* algorithm seems to be outputting abnormal pathing, the users monitoring the interface could have control and change the navigation algorithm to another navigation algorithm such as A* in the hopes of improving path planning and obstacle avoidance.

IV. Discussion

The main objective of this report is to show the progress that is being made to eventually allow our domain to autonomously race in different events such as F1Tenth [12] and Thuderhill. In order to successfully compete against different participants, our group's main goal is determine the most efficient algorithm for racing in these different tracks. By using our interactive interface, the user will be able to view path planning algorithm behavior as well as real-time sensory information outputted by the vehicle during navigation.

Currently, the interactive interface shows sensory information from the TurtleBot spawned inside the Gazebo simulation. These include the vehicle's current speed, position and RGB image. The interface is also subscribed to the /move_base_simple/goal ROS node that is generated from the path planning algorithms that help the vehicle to navigate autonomously. By clicking a single button, the user will be able to navigate the vehicle to the intended destination. In addition to all these features, the interface also displays the current path planning algorithm generating the most efficient path given a specific racetrack.

Future improvements to the interactive interface currently include optimizing visualizations to be more clear, subscribe to more nodes to receive more input data, and test potential latency with larger datasets or streams of data. Currently, several plots and tools on the interactive interface contain data that is self generated on a small scale, often referred to as 'dummy data'. Future ambitions include implementing the interface with more advanced datasets and streams of live input data.

V. Conclusion

In this report, our group successfully integrated navigation hardware into the TurtleBot that directly reflects the real-life robots built, as well as implemented the UCSD racing track inside the Gazebo simulator that mirrors the real-life track. G-Mapping was correctly implemented to create a 2-D grid map that was the basis for implementing path planning algorithms in the Gazebo Simulator such as A* and RRT*. The performance of these algorithms were tested based on different metrics, and these algorithms were able to visualize their performance on real map images of the real-life racetracks that have been postponed due to the on-going pandemic. Our main objective was to implement an interactive interface that will allow the user to control the vehicle and view significant sensory information obtained from the vehicle during autonomous navigation, and this was achieved by displaying different real-time sensory data, creating a platform for the user to monitor the path planning algorithms, and thus allowing the user to autonomously navigate the vehicle with ease.

VI. References

[1] Gazebo Simulator
http://gazebosim.org/tutorials?tut=ros_overview

[2] TurtleBot Robots
http://wiki.ros.org/Robots/TurtleBot

[4] UCSD Racing Track
http://github.com/garrettgibo/ucsd_f1tenth_simulator

[4] G-Mapping
http://wiki.ros.org/gmapping

[5] RViz
http://wiki.ros.org/rviz

[6] Python
https://www.python.org/

[7] CV2
https://pypi.org/project/opencv-python/

[8] Rosbridge
http://wiki.ros.org/rosbridge_suite

[9] HTML
https://html.spec.whatwg.org/

[10] Javascript
https://www.javascript.com/

[11] Thunderhill Racing Track
http://selfracingcars.com/

[12] F1Tenth Racing Track
https://f1tenth.org/

[13] Web Video Server
http://wiki.ros.org/web_video_server

[14] Adaptive Monte Carlo Localization (AMCL)
http://wiki.ros.org/amcl

[15] A. A. Zhilenkov and I. R. Epifantsev. "Problems of a trajectory planning in autonomous navigation systems based on technical vision and AI." 2018, https://ieeexplore.ieee.org/abstract/document/8317265

[16] Motion Planning for Urban Driving using RRT, http://acl.mit.edu/papers/KuwataIROS08.pdf

[17] D. Ma and N. Zhou. “Web-Based Robot Control and Monitoring.” 2019, http://www.cs.binghamton.edu/~szhang/teaching/18spring/reports/Luo-Ma-Zhou.pdf

[18] Calisi, Daniele and Nardi, Daniele. “Performance evaluation of pure-motion tasks for mobile robots with respect to world models.” 2009, https://www.researchgate.net/publication/200744624_Performance_evaluation_of_pure-motion_tasks_for_mobile_robots_with_respect_to_world_models