Fusing offline and online trajectory optimization techniques for goal-to-goal navigation of a scaled autonomous vehicle

Author, co-author (Do NOT enter this information. It will be pulled from participant tab in MyTechZone)
Affiliation (Do NOT enter this information. It will be pulled from participant tab in MyTechZone)

Abstract

Over the past few decades, enabling self-driving vehicles with the capacity to navigate a collision-free path efficiently and autonomously through an obstacle-filled environment has been of paramount interest. Motion-planning frameworks, encapsulating both path and trajectory-planning, have played a dominant role in realizing the deployment of a "sense-think-act" intelligence for autonomous vehicles. However, verification and validation of such intelligence on actual self-driving autonomous vehicles has been limited. Simulation-based verification and validation has the advantage of permitting diverse scenario-based testing and comprehensive "what-if" analyses – but is ultimately limited by the simulation fidelity and realism. In contrast, testing on full-scale real-world systems is constrained by the usual challenges of time, space and cost engendered in reproducing diverse scenarios in practice. Further, motion-planning frameworks often engender a mixture of global-planning (typically performed offline) coupled with a sensor-based local-planning (typically done online) which requires both simulation and physical testing.

Thus, scaled vehicle experimentation provides researchers with an interesting via-media to evaluate the performance and robustness of motion-planning algorithms on actual physical hardware – especially in real-time sensor-based motion planning setting. In this paper, we analyze the performance of a $1/10^{th}$ scale RC vehicle in simulation environment as part of software in loop testing for motion planning. A global planning algorithm is used to provide the waypoints for a feasible collision-free path between the start and goal configurations in the environment. We explored the deployment of Rapidly exploring Random Tree (RRT) and Rapidly exploring Random Tree* (RRT*). The Time Elastic Band local trajectory planner in ROS is then used for the realization of smooth, feasible paths between the waypoints. A comparison of validation in simulation has been provided with a detailed discussion of the parametric tuning for improving the performance in each case.

Introduction

Since the late 1990s, the automotive industry has observed growing interest and effort in developing technology to empower self-driving cars with the ability to operate efficiently on the streets in a collision-free manner. Computationally efficient, reliable, and robust motion planning techniques, as well as sophisticated validation methodologies, are crucial for achieving this goal.

The motion planning challenge is addressed from 2 principal perspectives: 1) Path planning and 2) Trajectory planning. In path planning (often treated in a global setting) the objective is to generate a set of feasible (and optimal) waypoints from the start to the goal configuration without causing a collision with objects in the environment. These waypoints are then sequentially tracked by the autonomous vehicle to navigate through/in the environment and traverse the desired path. Several methods for global planning has been explored in the literature including cell decomposition (Roadmap Methods vs. Cell Decomposition in Robot Motion Planning 2007), potential field method (Roadmap Methods vs. Cell Decomposition in Robot Motion Planning 2007), adaptations of graph search methods like A* [10] and Dijkstra's [4], variants of Rapidly-exploring Random Tree algorithms [5].

Global planning methods typically require a pre-computed map of the environment where objects and obstacles are typically static. The desired path from start to goal is computed offline considering only the static environment. Consequently, the effectiveness of global path planners is typically only measured based on their computational complexity and the major drawback is their inability to efficiently tackle dynamic objects in the environment. Some modifications to global planners exist which permit global map update based on sensory inputs. However, this results in slow processing due to the large computational complexity of the entire map being updated in real-time. On the contrary, the task performed by the trajectory planning or local planning algorithm is to ensure a physically realizable path from waypoint to waypoint.

Local planning approaches like the Dynamic-Window Approach [2] or Time-Elastic Band [1] consider the proximity of the robot in the world to compute velocity profiles. This myopic view of the world leads faster and simpler computation at the expense of subpar optimization with the local minima. Frequently, motion-planning frameworksoften engender a mixture of global-planning (typically performed offline) coupled with a sensor-based local (re-)planning (typically done online) which requires physical testing.

Verification and validation of the motion-planning algorithms are essential for the successful deployment of intelligent autonomous vehicles – which requires either physical testing or simulation. Currently, challenges ranging from lack of sophisticated computation platform/sensors/infrastructure etc. to the usual limitations of time/space/cost limit physical testing and validation on actual autonomous vehicles. In such situations, simulation-based testing environments offer the opportunity to exploit comprehensive "what-if" analyses and scenario-based coverage. Ultimately, the scale and scope of verification and validation is limited solely by simulation-

fidelity – nonetheless this fundamental inability to bridge the sim-to-real gap offers a real barrier and limits deployments. With the recent development of scaled vehicles offer a fair alternative to physically validate the algorithms while retaining the benefits of low-cost and (relative) ease of testing. They offer an opportunity of verifying the behavior and performance of blended global/local motion-planning while also examining the tradeoffs of simulation vs physical testing (and bridging the sim-to-real gap in the context of these scaled autonomous vehicles).

For example, Marin-Plaza et al [6] analyzed the performance of an Ackermann steered non-holonomic wheeled vehicle blending Dijkstra's global planning with a Time Elastic Band (TEB) local planner. In our work paper, we explore this dimension even further by using a $1/10^{th}$ scale RC vehicle, operated through Robot Operating System (ROS), in a Gazebo-based simulation as well as on the actual F1/10 hardware vehicle. The path planning algorithms developed for this analysis were Rapidly Exploring Random Tree (RRT) and Rapidly Exploring Random Tree* (RRT*) which is developed in Python and provides the collision-free, kinematically feasible waypoints between the start and goal configurations in the environment. The robot navigation through the environment has been performed using the TEB local planner. We compare and contrast the differences in the performance of the simulated system against the actual system with both motion-planning techniques.

Literature Review

Over the years, there has been a tremendous amount of research and experimentation for applying different techniques and strategies for motion planning for mobile robots. These implementations can range from offline/online trajectory optimization, or fusion of both the techniques to tackle dynamic obstacles in the environment. The paradigm put forward by global planning alone is to facilitate offline trajectory generation from one point to another in static environments which are highly map dependent. This is well highlighted by E. G. Tsardoulias et.al [11] in their study exploring global planners for motion planning ranging from Occupancy Grid Maps (OGMs): Probabilistic Roadmaps (PRMs), Visibility Graphs (VGs), Rapidly exploring Random Trees (RRTs) and Space Skeletonization.

However, in the real world, unless for a specific controlled environment, it is not feasible for a motion planning application to strictly rely on offline optimization methods and not consider dynamic obstacles in its path. This is where the paradigm of local planning made massive strides enabling robot motion in dynamic and highly uncertain environment. Development of local planners started from simple methods like edge detection to generated contour path around the obstacles and potential field methods simulating repulsive forces from the obstacles while the goal position entails an attractive force [6]. Post these earlier approaches and with the increase in onboard computing power, newer instantaneous methods like Vector Field Histogram and Dynamic Window approach started becoming more popular. [7]

As an alternative to the Dynamic Window approach, the Time-Elastic Band approach enabled local planning for car-like robots while providing with computational benefits and much more optimal trajectory as highlighted by the observations put forward by Christopher et al. [1] in their study and comparative analysis of the two motion planners. On top of that, a ROS based library specifically designed to implement TEB local planner on car-like robots made it the obvious choice for local planning on our scaled autonomous vehicle.

*Offline Trajectory Optimization with RRT and RRT**

Unlike traditional approaches to motion planning like grid-based or interval-based search, the RRT and RRT* algorithms do not have existing nodes between the start and endpoints. Thus, these algorithms have to create both the graph and path for navigation. RRT and RRT* are one of the most popular sampling-based probabilistic algorithms for motion planning [3].

RRT algorithm

The rapidly exploring random tree algorithm is a motion planning algorithm which primarily satisfies the single query applications [11]. While the versions of this algorithm are available for 2D and 3D motion planning both, for our purpose we will use the 2D and single query algorithm for our experimentation purposes. This algorithm uses an existing map, generates sample nodes, and starts building the tree (from a single branch to a fully developed tree), which eventually reaches our query configuration. A query configuration is usually a user-defined point in the map where we want our vehicle to reach, also known as goal configuration. At each iteration of building the tree, a random sample is generated anywhere in the map and a new connection is added to the nearest tree node in direction of the new sample by a pre-defined length. This can be seen in the following pseudo-code.

As proved in previous experimentations [3], the path generated by RRT as seen in Figure[1] is cubic in nature as newly generated nodes are attached to the nearest neighbor. But the benefit of this algorithm is its high speed (in sampling-based planner category) and easy implementation. The path generated is not necessarily a traversable path by an Ackermann steered vehicle, some further smoothening is needed so that the path can be traversed by a vehicle in motion.

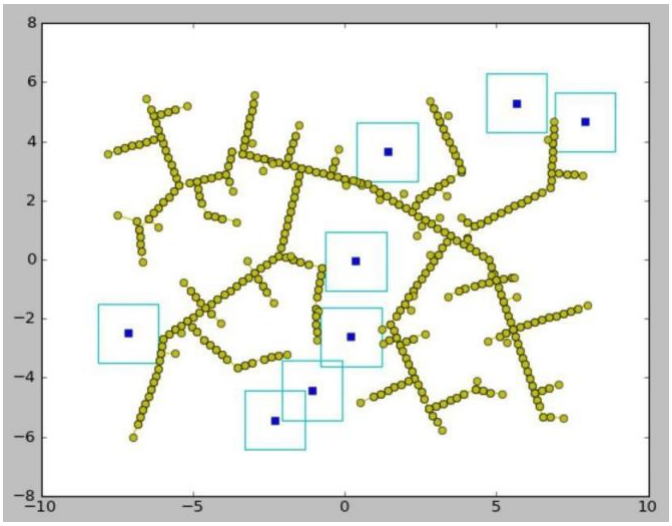

Figure 1: RRT algorithm [3]

RRT* algorithm

RRT* is an optimized version of the RRT algorithm [5]. While iterating to sample through new nodes, the samples are not simply added to the nearest node in the tree, but a comparison is made among nearby tree nodes to calculate which tree node is connected to the start node more closely. An edge is created

between the new node and the existing node which reduces the overall cost of reaching the new node. For doing this a cost to reach a certain node must be maintained along with the node location. This extra calculation increases the space complexity and searching through the nearby nodes to find the most optimal node to connect to increases the computational complexity of the algorithm at each iteration. Hence this implementation is relatively computationally complex than simple RRT [3]. But as expected through this additional step, it creates the shortest possible path to the goal configuration. If we tune this algorithm well considering vehicle space, it is also able to give us a smooth path which can be traversed by an Ackermann steered vehicle, but this is not necessarily true in all cases.

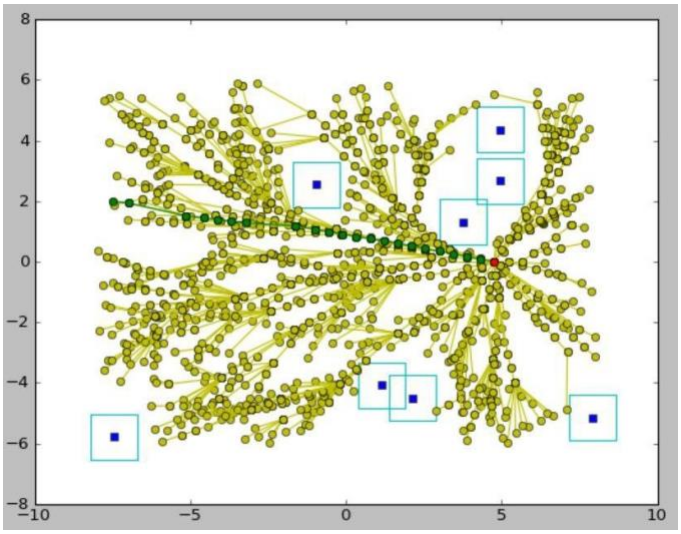

Figure 2: RRT* algorithm [3]

TEB local planner

The concept of elastic band deals with deforming the path generated by global planners as discussed previously in section II. This allows the vehicle to generate a path which avoids an obstacle in its path. The deformed path takes this obstacle into account and creates a safe trajectory around the path. Although it is clearly stated in [1] and [9], this does not consider vehicle dynamics, but this process shows great reliability for Ackermann vehicles which are navigating into the unexplored areas. This entire process, when visualized, can be compared to an elastic band. This problem is formulated in a weighted multi-objective optimization framework. The objectives for this framework are locally formed and deployed, which makes this system highly modular [10]. This makes Timed Elastic Band (also known as TEB) approach flexible as seen in Figure[3] and hence it is chosen as a local planner in our experimentation. The simulations and results of this planner have shown that it is a computationally efficient and robust approach for online obstacle avoidance [1]. From our previous experiments with this algorithm, we can proceed and rely on this algorithm for local trajectory planning as it shows good confidence in real hardware and can successfully avoid obstacles.

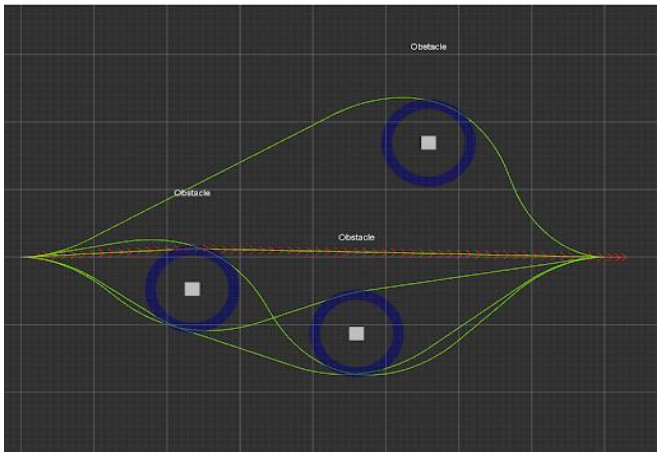

Figure 3: TEB local planner

Methodology and Implementation

F 1/10th scaled autonomous vehicle hardware

The F 1/10th scaled vehicle platform is conceptualized and developed by University of Pennsylvania F1tenth.org enabling students to develop and deploy autonomous driving functionalities simply and cost-effectively. We have used the second generation of the F1/10th platform car which is a modified remote-controlled TRAXXAS Ford Fiesta ST equipped with a lower-level chassis (refer Figure 4). This houses the brushless DC motor for powering the four wheels using a differential and propeller shaft along with a servo motor to enable steering. A platform is mounted on the base chassis with the help of M3 standoff screws which house all the autonomy components of the car including the Hokuyo 10LX LiDAR, the VESC MK3 speed controller, providing PWM signals based on the digital wheel velocity command inputs and the Nvidia TX2. The Nvidia TX2 SOC is the main brain of the entire system. It gives commands to the VESC which controls the Servo and the Brushless Motor. The TX2 is essentially a supercomputer on a module. We attach it to the Orbitty Carrier board so that we can access the TX2's peripheral. An array of size 1x1080 over an arc of 270° is obtained from the Hokuyo LiDAR which is used for Localization and Mapping. The system design is illustrated in Figure[5].

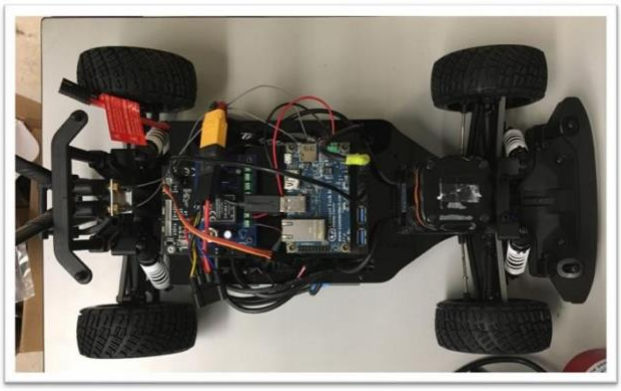

Figure 4: TRAXXAS f1/10th scaled vehicle

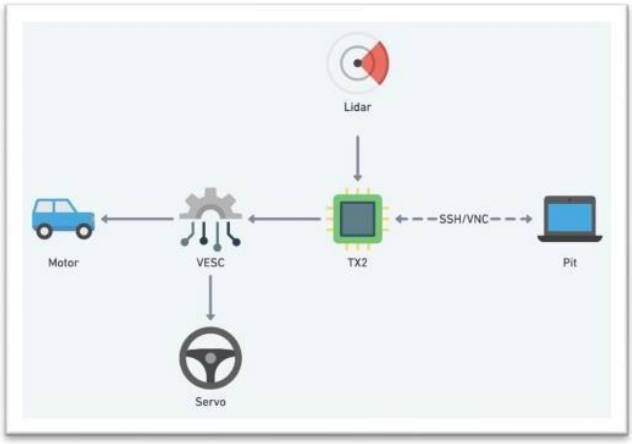

Figure 5: System Integration

Gazebo simulator environment

The environment selected for this project was a Gazebo simulation environment (refer Figure [6]) provided by the F1/10 research group from the University of Virginia that was customized to provide users with a realistic simulation environment for experimentation in the domains of perception, motion planning and control with a scaled Ackerman-steered vehicle.

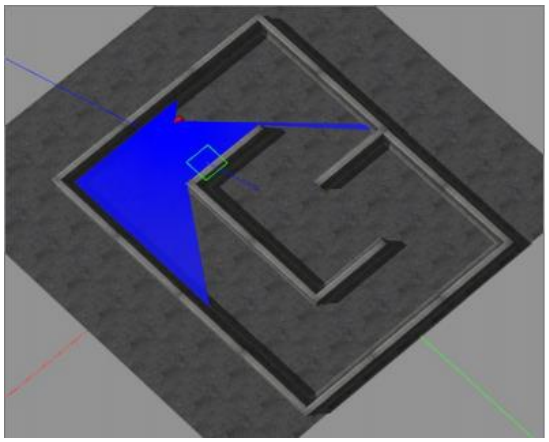

Figure 6: UVA F 1/0th simulator

The F1/10 Gazebo Simulator complements the hardware by emulating its modular properties and was created to help the user get started with the simulator out-of-the box. It builds on the MIT-Racecar gazebo simulation baseline implementation but also contains additional features like a world map, and Gazebo plugins that provide better odometry and control. The F1/10 console package provides you with the option of using either keyboard control or joystick control (built in compliance with Logitech F710 game controller). In particular, the F1/10 platform is based on a 1/10 scale Ackermann steered RC car whose kinematics need to be captured and used as part of the simulation – this is facilitated through the use of the TEB local planner. At this stage, we exploit the simulator support for Basic Navigation (Wall following with tunable PID control), Simultaneous Localization and Mapping (SLAM using Hector Mapping) to build occupancy grid RVIZ maps, as well paving the way for advanced navigation (using the TEB local planner)

*Global Planning with RRT and RRT**

To develop a road map using the RRT or RRT* algorithm, it is essential to obtain a detailed map of the environment. The F1/10 autonomous vehicles come with an onboard LiDAR sensor. The car was manually teleoperated around the simulated environment with the help of keyboard control and at the same time, the LiDAR scans were processed through a ROS package for Hector SLAM (Simultaneous Localization and Mapping) algorithm. The figure shows the Gazebo environment used and the original map generated from SLAM (dimensions: 2048x2048 pixel). In order to represent the map in tkinter GUI library in Python for sampling-based planning approaches using a road map, it was necessary to represent the occupancy grid map obtained from SLAM. To do that, a .PNG image file representing the SLAM map was first loaded onto MATLAB and cropped from 2048x2048 size to 300x200 dimension to represent the bounded region/environment. The coordinates of the corners of the walls were recorded approximately using the data-tip feature in MATLAB. The coordinates were then saved in a .CSV file and loaded in the tkinter GUI to represent the walls as rectangular obstacles.

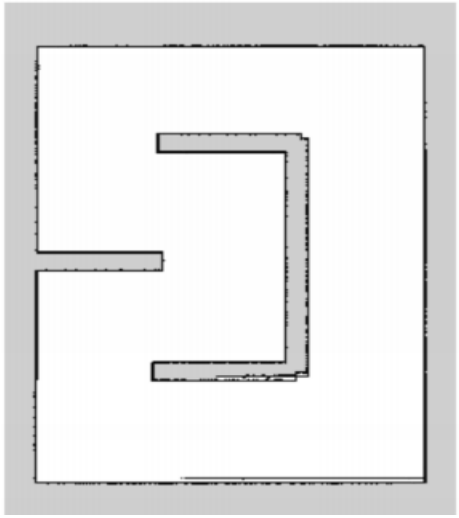

Figure 7: Hector SLAM of simulation environment

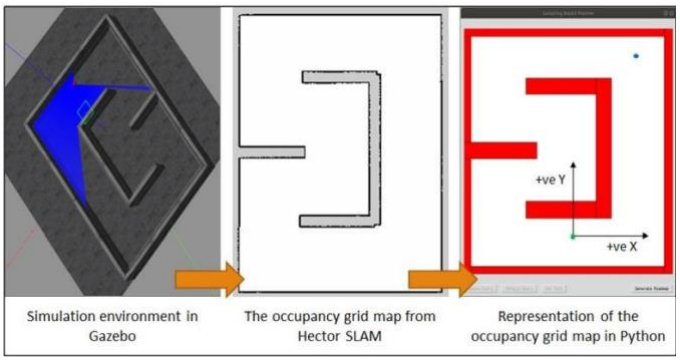

Figure 8: SLAM Map --> Grid map (Python)

*Methodology for RRT and RRT **

We implemented a Vanilla RRT algorithm since the controller used to drive the F1/10 car incorporates the steering action, meaning no explicit consideration for vehicle motion is necessary. Using the following algorithms from start configuration, where the vehicle is spawned in the environment, we generate the path using RRT and RRT* algorithms, respectively.

RRT Pseudo code

```
Qgoal //region that identifies success
Counter = 0 //keeps track of iterations
lim = n //number of iterations algorithm should run for
G(V,E) //Graph containing edges and vertices, initialized as empty
While counter < lim:
    Xnew = RandomPosition()
    if IsInObstacle(Xnew) == True:
        continue
    Xnearest = Nearest(G(V,E),Xnew) //find nearest vertex
    Link = Chain(Xnew,Xnearest)
    G.append(Link)
    if Xnew in Qgoal:
        Return G
Return G
```

Figure 9: RRT algorithm Pseudo code [3]

RRT* Pseudo code

```
Rad = r
G(V,E) //Graph containing edges and vertices
For itr in range(0…n)
    Xnew = RandomPosition()
    If Obstacle(Xnew) == True, try again
    Xnearest = Nearest(G(V,E),Xnew)
    Cost(Xnew) = Distance(Xnew,Xnearest)
    Xbest,Xneighbors = findNeighbors(G(V,E),Xnew,Rad)
    Link = Chain(Xnew,Xbest)
    For x' in Xneighbors
        If Cost(Xnew) + Distance(Xnew,x') < Cost(x')
            Cost(x') = Cost(Xnew)+Distance(Xnew,x')
            Parent(x') = Xnew
            G += {Xnew,x'}
    G += Link
Return G
```

Figure 10: RRT* algorithm Pseudo code [3]

The path obtained from these algorithms is in pixel coordinates and is multiplied by the resolution of the map (0.05m/pixel) to obtain the waypoints of the trajectory in meters. This is because the local coordinate frame of the F1/10 cars is in meters.

The generated map has a co-ordinate frame originated from the spawn location of the vehicle in the environment. This co-ordinate frame is set to match the coordinate frame of the starting location of the vehicle. It further aids in generating waypoints from vehicle's start configuration to the goal configuration. In case of Vanilla RRT, the path generated is cubic in nature, resulting into a non-feasible vehicle traversal. Such paths need to be smoothened for non-holonomic car like vehicles in order for trivial controllers to adapt to the cubic shaped nature of the path. To solve this problem, one of the solutions is to incorporate vehicle kinematics into path generation. Alternatively, a robust controller can also take care of the non-traversable paths by smoothening out vehicle motion while going over cubic path waypoints. A path generation algorithm like RRT* naturally generates a traversable path in most of the simple start and goal configurations. With large number of nodes, the path tends to get even smoother. Robust local planners like TEB can react to such path aberrations and make vehicle traversal easy. It has been observed that the path obtained from the RRT algorithm is notoptimal and requires further smoothing while RRT* provides us with an optimal path.

Figure [11] and [12] show a python-based map visualization, which converts the existing map into a C_{free} space. The boundaries are bloated considering the vehicle width in real world, so that a vehicle would not select a RRT / RRT* node in the red region, when in reality these will act as a virtual obstacle which are too close to any boundary or obstacle. The white space in the figure is the traversable map by the vehicle. The nodes are getting populated only in the white region of the map. Figure [11] and [12] also shows the start and the end points of the vehicle in green and blue respectively. The branches connecting each point in the map is the generated RRT/ RRT* tree as a result of the respective algorithms. Figures [11] and [12] also highlight the path generated by the RRT and RRT* algorithms when different goal locations are provided to the vehicle. The path generated by RRT* varies in Figure [12] due to the nature of the RRT* algorithm enabling more optimized path from start to goal location.

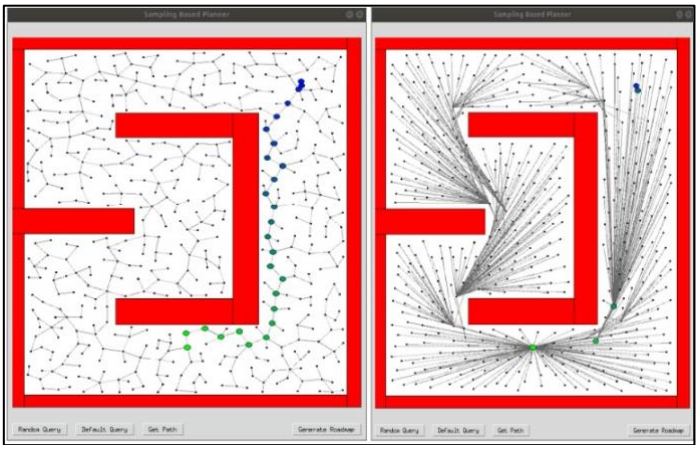

Figure 11: The trajectory between start → (0,0) and goal → (75,215) from RRT and RRT* (r) with 400 nodes

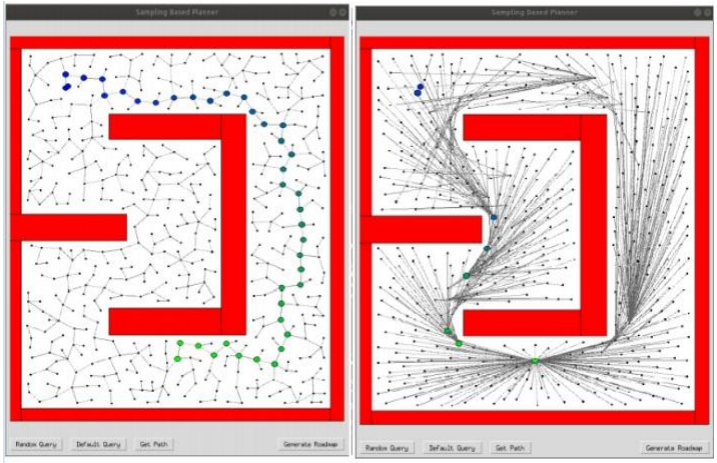

Figure 12: The trajectory between start → (0,0) and goal → (-75,215) from RRT (l) and RRT* (r) with 400 nodes

Local Planning with ROS Navigation Stack

ROS Navigation Stack

A ROS Navigation stack is a meta-package which is developed for navigating any custom robot to a given goal point by taking into consideration multiple sensor data inputs. It reduces the cumbersome task of the user to write their own codes of navigation elements (localization, mapping, control and planning). The basic functionality provided by the navigation stack is similar to a framework which perceives the sensor data (odometry, LiDAR data etc.) as input, provides a platform for the control

algorithm to make use of this sensor data, and outputs a velocity to the mobile base. In order to run the ROS navigation stack on our robot, a certain set of prerequisites need to be met. These range from having onboard compute capable of running ROS on Ubuntu, the knowledge pertaining to the frame transformations on the robot and publishing the sensor data using the appropriate ROS topics.

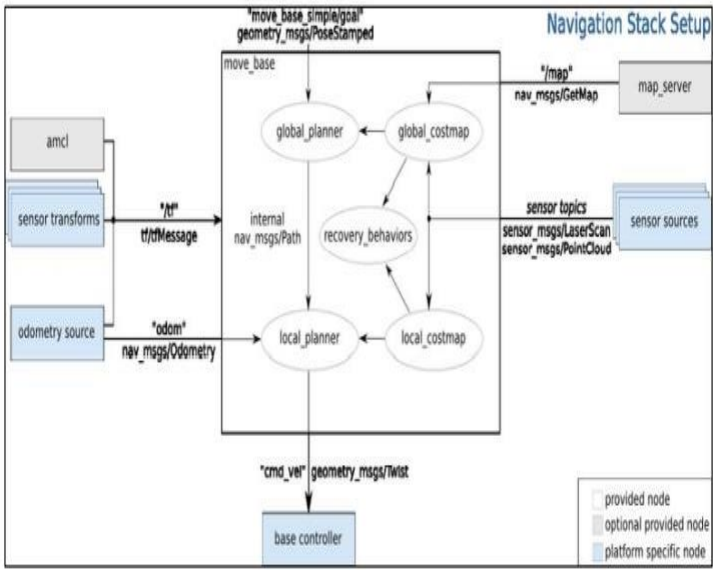

Figure 13: ROS Navigation Stack

Implementing the TEB local planner for online trajectory optimization in ROS

This package implements an online optimal local trajectory planner for navigation and control of mobile robots as a plugin for the ROS navigation package. The initial trajectory generated by a global planner is optimized during runtime w.r.t. minimizing the trajectory execution time (time-optimal objective), separation from obstacles and compliance with kino-dynamic constraints such as satisfying maximum velocities and accelerations. The current implementation complies with the kinematics of non-holonomic robots (differential drive and car-like robots). Support of holonomic robots is included since ROS Kinetic version. The optimal trajectory is efficiently obtained by solving a sparse scalarized multi-objective optimization problem. The user can provide weights to the optimization problem in order to specify the behavior in case of conflicting objectives.

Topics published by TEB local planner:

~/global_plan (nav_msgs/Path)
~/local_plan (nav_msgs/Path)
~/teb_poses (geometry_msgs/PoseArray)
~/teb_markers (visualization_msgs/Marker)
~/teb_feedback (teb_local_planner/FeedbackMsg)

Topics subscribed by TEB local planner:

~/obstacles (costmap_converter/ObstacleArrayMsg)
~/via_points (nav_msgs/Path)
~/odom (nav_msgs/Odometry)

Timed Elastic Band approaches the planning problem using a hyper-graph based nonlinear optimization technique and the implementation with an open-source C++ framework called general (hyper-)graph optimization (g2o) which solves graph based nonlinear optimization problems.

Implementation of Navigation Stack/TEB local planner on UVA simulator:

The implementation detail involves creating a SLAM map of the environment. For this purpose, a simple ROS based Hector Mapping approach has been used. This map is saved in PNG and YAML extensions which saves map grid data as well as meta data. A map node is created using ROS tools which is then used further for localization purpose. An advanced Monte Carlo localization technique has been used to locate the vehicle based on laser scan and odometry data. ROSLAUNCH based execution makes it easy to simultaneously open all the related nodes in ROS and making it a smooth process for implementing all these ROS based packages simultaneously. A TEB planner node is also executed using this ROS based tool.

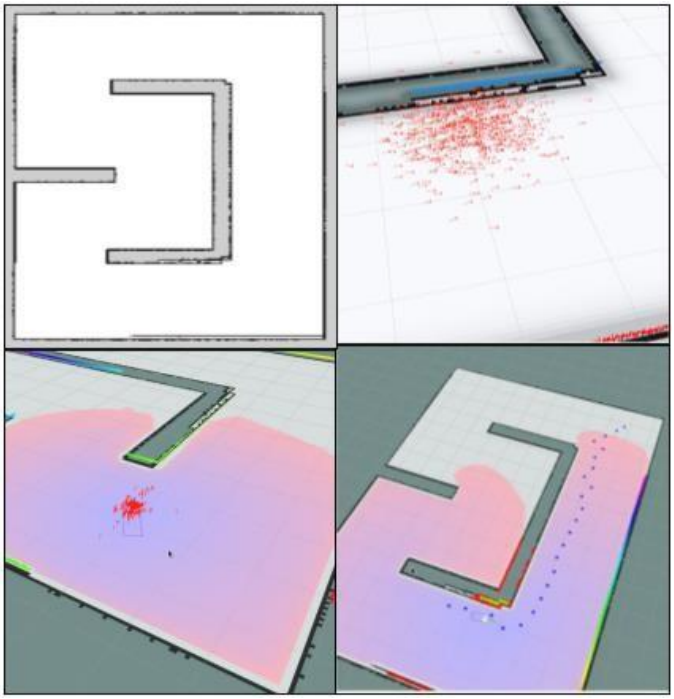

Figure 14: Process of Motion Planning

Results and Discussion

The results section shows the actual implementation of codes created by the team members to generate RRT/RRT* global planner and simulate the path-following in a Gazebo custom environment. To conclude the entire project, we can say that this mechanism can be deployed straight up on any environment given the dependencies are installed in a correct manner. The RRT and RRT* algorithms

generated based on given SLAM map result into the paths denoted by colored dots in the Figure [14].

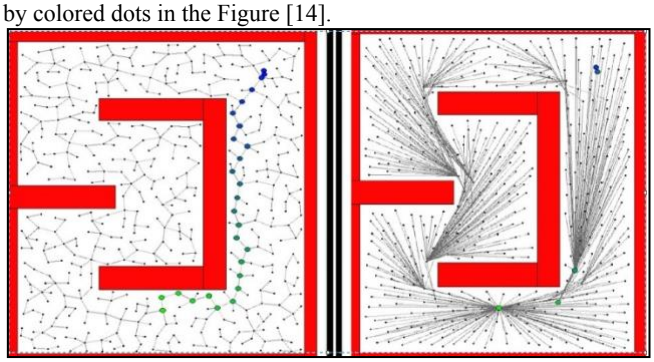

Figure 15: RRT and RRT* comparison

These points are then translated into relevant coordinates in the SLAM map and a local TEB planner is used to navigate using this global plan as can be seen in the Figures [16] and [17]. Figure [16] and Figure [17] are a combined imagery of Gazebo simulator (left) and RVIZ simulator (right). The blue region in the left figure is the lidar scan originating from vehicle's simulated lidar (It can be referred as scan region of the vehicle.) The color gradient image on the right in both figures is a potential gradient field spread out throughout the map to describing areas with least resistance (blue shaded region) and areas with high resistance or areas close to boundaries and obstacles (orange shaded region). This potential field is generated using the TEB local planner's artificial potential field algorithm, which is an inbuilt tool which helps the local planner to avoid obstacles in the given map.

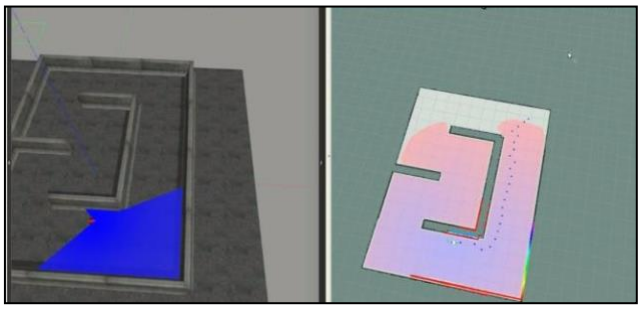

Figure 16: RRT + TEB fusion on simulator

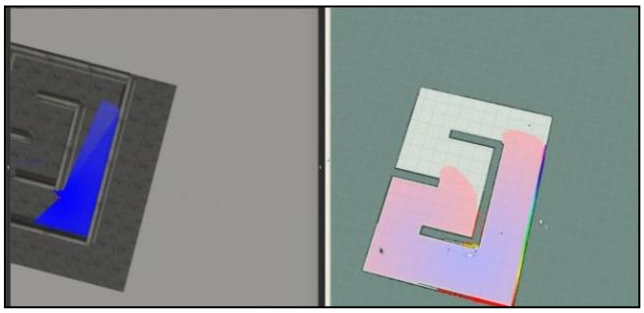

Figure 17: RRT* + TEB fusion on simulator

The path generated by RRT is observed to be more cubic and in contrast the path generated by RRT* appears to be smoother and more traversable. A cost function generated by TEB local planner is shown to be spread out throughout the map which gives the local planner an estimate of vehicle and obstacle poses.
Due to the nature of the tree, it is observed that RRT* yield optimal yet computationally intensive paths. Furthermore, increasing the number of nodes also impacts the runtime. However, it has been observed that since the target configuration has been sampled at a probability of 10% around the actual goal configuration, the algorithms can ensure a feasible path with as low as 100 nodes. This would not be possible if the goal configuration is unknown and the algorithm is expected to explore the environment in all directions. Another reason why RRT* takes significantly longer to run is that, during rewiring, the algorithm compares against all the nodes in the tree to look for the optimal path cost [3]. This can be overcome if the rewiring approach is modified to look for only K-nearest-neighbors to determine the optimal cost to reach the newly added vertex:

Serial No.	No. of nodes	Time in sec (RRT)	Time in sec (RRT*)
1	500	233.76	678.11
2	400	72.45	297.7
3	300	25.73	226.91
4	200	10.73	91.9
5	100	2.83	34.39

Figure 18: Time taken by RRT and RRT* to compute path based on different nodes

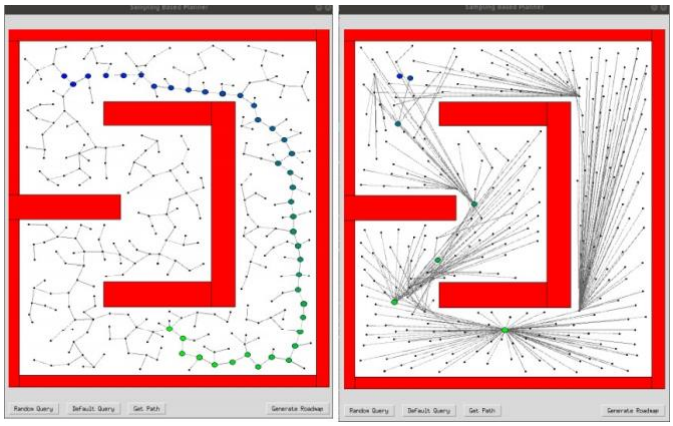

Figure 19: RRT and RRT* with 300 nodes

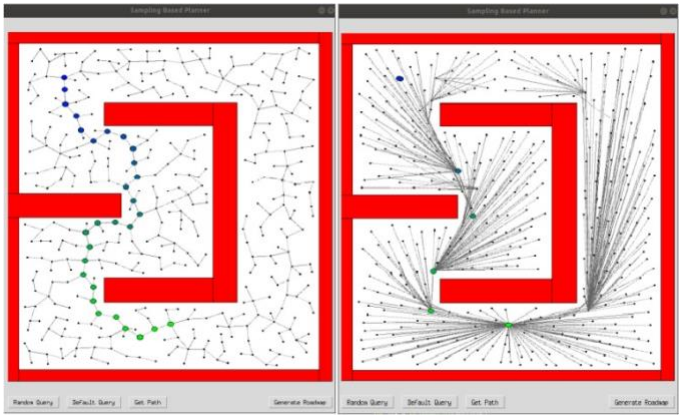

Figure 20: RRT and RRT* with 400 nodes

From the paths from RRT and RRT*, it is observed that although a feasible path is always generated by both algorithms, RRT may generate different waypoints due to the probabilistic nature of the algorithm. Also, RRT generated paths are not always optimal. The path generated may not optimize even though the nodes are increased, once the goal is reached at a given number of nodes. In contrast, the path generated by RRT* is always optimal and due to its rewiring nature, as the number of nodes increase, the optimality of path also increases.

Future work scope

By fusing the global and local path planning techniques over the simulation environment, the vehicle can attain goal-to-goal navigation. As a continuation of our work, we would be implementing the test in real world under lab conditions and presenting our findings in the final manuscript. The following picture highlights the experimental setup for our real-world test. The MoCap cameras shown in the Figure [21] will be used for tracking the actual position in world frame and it will be used to compare the accuracy in path tracking by TEB local planner.

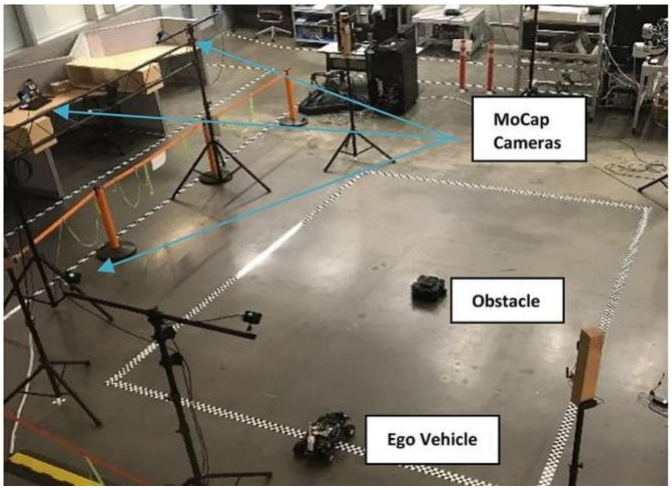

Figure 21: Lab setup for experimentation

References

1. C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann and T. Bertram. 2012. "Trajectory modification considering dynamic constraints of autonomous robots." *7th German Conference on Robotics, Germany, Munich.*

2. D. Fox, W. Burgard and S. Thrun. 1997. "The dynamic window approach to collision avoidance." *IEEE Robotics & Automation Magazine* 4: 22-23.

3. Tim Chin, "*Robotic Path Planning, RRT and RRT**"

4. Javaid, Adeel. 2013. "Understanding Dijkstra Algorithm." *SSRN Electronic Journal.*

5. Naderi, Kourosh & Rajamäki, Joose & Hämäläinen, Perttu. 2015. "RT-RRT*: a real-time path planning algorithm based on RRT*."

6. Pablo Marin-Plaza, Ahmed Hussein, David Martin, and Arturo de la Escalera. 2017. "Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles." *Hindawi Journal of Advanced Transportation* 10.

7. Portugal, David Bina Siassipour. n.d. "A study on local planning techniques."

8. 2007. "Roadmap Methods vs. Cell Decomposition in Robot Motion Planning." *Proceedings of the 6th WSEAS International Conference on Signal Processing, Robotics and Automation.*

9. Rosmann, Christoph, Frank Hoffmann, and Torsten Bertram. n.d. "Planning of Multiple Robot Trajectories in Distinctive Topologies."

10. Sharma, Shrawan Kr. 2015. "Shortest Path Searching for Road Network using A* algorithm." *International Journal of Computer Science and Mobile Computing.*

11. Tsardoulias, E. G. 2016. "A Review of Global Path Planning Methods for Occupancy." *J Intell Robot Syst.*

12. Karaman, Frazzoli. 2011 "Sampling-based Algorithms for Optimal Motion Planning"

Contact Information

Ajinkya Joglekar
Mailing Address: 112 Folkstone St, Greenville, 29605
Email Address: ajoglek@clemson.edu
Phone: +1 8645531798

Acknowledgements

The authors would like to acknowledge support of their colleagues from ARM Lab at the Clemson University International Center for Automotive Research. We also acknowledge partial support for this work from the National Science Foundation via the Computing Community Research Infrastructure grant (CNS-1925500).

Definitions/Abbreviations

RRT	Rapidly Exploring Random Tree
RRT*	Rapidly Exploring Random Tree RRT
ROS	Robot Operating System
TEB	Timed Elastic Band
GUI	Graphical User Interface
SLAM	Simultaneous Localization and Mapping
LiDAR	Light Detecting and Ranging
PRM	Probabilistic Road Mapping

Updates in response to comments on MyTechZone portal

➔ Comments by Reviewer #275489

1. In Abstract, the reader is expecting the presentation of full-scale real-world testing on the scaled vehicle. However, at the end of the paper it is presented only as a future work topic:
   Resolution: Abstract updated based on project progress.
2. In page 5, pseudo codes for both RRT and RRT* algorithms seem identical:
   Resolution: This has been addressed in the form of Figures [9] & [10]
3. Figure 9 and Figure 10 legends are identical. The number of nodes should be added:
   Resolution: The number of nodes have now been added to these figures. Refer Figures [11] and [12].
4. In Figure 10 there is a remarkable change of trajectory from RRT* algorithm, at the left of the center wall, which is not commented:
   Resolution: A paragraph above the figures now explains the reason for different paths. A further explanation is also provided in the results section.
5. In References, the author of Reference 3 is suggested be added, as well as the date of page access:
   Resolution: Reference fixed according to the required format.

➔ Comments by Reviewer #275493

1. The paper should reflect scientific papers quality. For example, please refrain from using snapshots/screenshots for figures in scientific / technical papers:
   Resolution: Screenshots were removed, and figures were cleaned up. Many of the images are however essential for illustration of the concepts as this is primarily a simulation-based implementation of motion planning.
2. Pseudocodes of RRTs in page 5 - this should be written in either proper pseudocodes or flowcharts:
   Resolution: Changes made.
3. Figure 11 should be enlarged:
   Resolution: Changes made, now Figure [12]
4. Figure 14-18 - what is what in these figures? Where is the legend? Proper plotting should be done:
   Resolution: Paragraph before Figure [15] explains the details in the simulator (Figure [15]) along with detailed explanation of the process.