A Trajectory Library-Based Local Planner for Ground Robots in Unstructured Environments

Rohan Chandrasekar
Department of Mechanical Engineering,
Carnegie Mellon Univeristy,
5000 Forbes Ave, Pittsburgh, PA,USA
email: rohancha@andrew.cmu.edu

Matthew Travers
The Robotics Institute,
Carnegie Mellon University,
5000 Forbes Ave, Pittsburgh, PA, USA
email: mtravers@andrew.cmu.edu

Navigating in cluttered environments presents significant challenges for autonomous systems. Although finding the shortest path from a start to a goal location is a well-addressed problem, as robots traverse this path, they encounter numerous unforeseen obstacles that are not accounted for in the initial global plan. These obstacles are dynamic in nature, making it difficult to establish a reliable path over time. Several works like A and RRT generate real-time plans for robots as they navigate through an environment but often fail to incorporate the dynamics involved. In this work, we introduce a trajectory-library-based local planner designed for high-speed, non-holonomic autonomous ground robots. This planner not only adeptly navigates the dynamic challenges of the environment but also accounts for the dynamics of the robot. In this trajectory-library-based local planner, the knowledge of the robot's dynamics is leveraged to generate trajectories at varying speeds. These pre-computed trajectories are then overlaid onto a grid while also considering the robot's footprint. This grid is then convolved with the obstacle map, effectively filtering out any paths that intersect with obstacles. This streamlined collision-checking process is executed in less than two milliseconds, facilitating rapid decision-making. To validate our approach, we tested our methodology both in simulated environments using Gazebo and through practical trials with an RC car in real-world scenarios.*

Keywords: Path Planning, Robotics, Local Planning

1 Introduction

Autonomous robot navigation has seen substantial advancements in algorithms due to its critical applications in various demanding scenarios in search and rescue missions. In the 2020 DARPA Subterranean Challenge, robots were tasked with exploring unfamiliar spaces, requiring efficient navigation to map and traverse these areas efficiently in a short time. These challenges typically unfold in complex and unstructured environments, where the autonomy must be robust to dynamic changes. Successfully navigating such environments is crucial, necessitating a highly effective local planner to ensure precision and reliability in real-time decision-making.

Local planning for autonomous robots navigating unstructured terrain presents considerable challenges, especially due to the complex and unpredictable nature of real-world scenarios. Conventional methods, which focus on generating paths online at every timestep, can be slow and computationally demanding. In this work, we develop a robust trajectory-library-based local planner for high-speed navigation. This approach involves pre-generating paths offline and employing efficient collision-checking techniques, significantly reducing the computational load during operation. Our collision-checking approach draws inspiration from the method described in [1], which effectively removes trajectories that intersect with obstacles through the use of bitwise operations. Additionally, we aim to validate the generated trajectories through testing in both simulation and hardware environments.

The main contributions of this work are outlined as follows:

- Implementing a trajectory-library-based local planner for unstructured environments.
- Implementing a methodology for effective grid-based collision checking that operates in real-time, enhancing navigational safety and responsiveness.
- Developing a navigation stack that can be directly implemented and tested on real robots, facilitating practical application and validation of the theoretical models.

Version 1.18, May 27, 2024

2 Related Work

Local planners can be categorized into several types, including reactive planners, sampling-based planners, optimization-based planners, and graph-based planners.

Reactive planners include potential field-based techniques [2], where the goal acts as an attractive force to the vehicle while obstacles exert a repulsive force. The dynamic window method [3], on the other hand, samples a set of velocities that satisfy the dynamic constraints of the robot and then evaluates each for collision probability, adherence to the global path, and progress toward the target. Reactive planners are not well-suited for complex or highly dynamic environments, as their simplistic decision-making process can be overwhelmed by rapid changes and multiple variables. In settings with closely spaced obstacles, they may cause the robot to oscillate or switch between paths due to conflicting sensor data, resulting in inefficient motion and increased mechanical wear.

Sampling-based planners navigate through complex or high-dimensional spaces by randomly sampling the configuration space and incrementally constructing a path or a graph that connects these samples. Methods such as PRM [4], RRT [5], and RRT-connect [6] have been proposed to solve the local planning problem. While these methods are probabilistically complete, they do not guarantee the shortest or most optimal path and can produce paths that vary significantly with each execution due to the randomness of the sampling process, leading to inconsistency in performance or path quality.

Trajectory planning for autonomous cars often utilizes numerical optimization techniques. Techniques that integrate sampling with numerical optimization, taking into account the non-holonomy of the robot and differential constraints through a series of equality and inequality constraints, are discussed in [7]. Additionally, methods that combine states, control inputs, and time intervals into a unified trajectory representation facilitate the planning of time-optimal trajectories within the model predictive control framework, as described in [8]. Optimization-based local planners face challenges such as computational complexity, susceptibility to local minima, and dependence on accurate models, which can impede

their real-time application and effectiveness in dynamic environments. These planners also require careful tuning of parameters and can struggle with scalability and robustness to noise and uncertainties. These characteristics make optimization-based planners less suitable for high-speed local planning in practical, real-world situations.

In graph-based planners, nodes represent possible states or locations, and edges denote viable transitions or movements between these nodes. This graph serves as the framework for navigating from an initial point to a target destination. Utilizing well-established graph traversal techniques, such as Dijkstra's algorithm[9] or A* search[10], graph-based planners effectively find optimal or near-optimal paths by minimizing the cost function. Graph-based planners for autonomous robot driving can face challenges in highly dynamic environments where frequent re-planning is necessary due to their reliance on pre-defined graphs that may not quickly adapt to sudden changes in the environment. Additionally, the computational overhead associated with maintaining and updating large graphs can become impractical in scenarios where the environment is extensive or exceptionally complex.

In this study, we aim to overcome the limitations of existing planners documented in the literature by concentrating on reducing computational demands and developing a robust planner suited for unstructured environments. This approach intends to enhance efficiency and adaptability in complex scenarios where traditional planners may falter.

3 Method

3.1 Offline Phase.

3.1.1 Path Generation. As the initial step in the path planning process, paths are generated offline using the Kinematic Bicycle Model (KBM). The reference point selected for generating these paths is the rear axle of the RC car, as illustrated in Figure 1. Upon applying the instantaneous center of rotation, the rate of change of the states is calculated as:

$$\dot{x} = v\cos(\theta),$$

$$\dot{y} = v\sin(\theta),$$

$$\dot{\theta} = \frac{v\tan(\delta)}{L},$$

$$\dot{\delta} = \phi$$

Once the rates of change are calculated using the velocity v, heading angle θ, steering angle δ, length of the vehicle L, and the rate of change of the steering angle ϕ, the next state is then computed as:

$$x_{t+1} = x_t + \dot{x}\Delta t,$$

$$y_{t+1} = y_t + \dot{y}\Delta t,$$

$$\theta_{t+1} = \theta_t + \dot{\theta}\Delta t,$$

$$\delta_{t+1} = \delta_t + \dot{\delta}\Delta t.$$

Utilizing this model, paths are generated across a spectrum of initial conditions: four different steering angles and three different initial acceleration values are considered. The speed range for these simulations spans from -2 m/s to +6 m/s, discretized in 1 m/s increments. The speed is limited to a maximum of -2 m/s when moving backward to prevent instability while driving. The paths are generated for 2 seconds, which means that a 2m/s path will span 4m, and a 6m/s path will span 12m. This makes sure that the paths are sufficiently long and, therefore, gives enough time for the robot to stop when it encounters an obstacle.

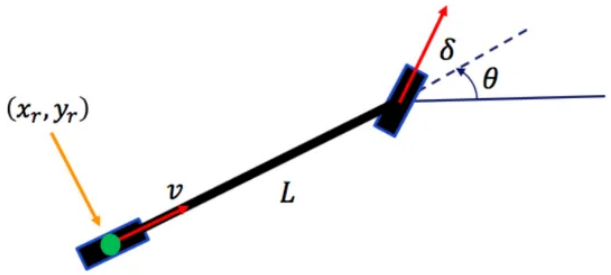

Fig. 1 The kinematic bicycle model is derived from the rear axle of the vehicle. Here, θ represents the heading angle, δ denotes the steering angle, v is the velocity, L is the length of the bicycle, and x_r and y_r are the coordinates of the center of the rear axle.

During the path rollout, each point is evaluated to determine if the lateral acceleration exceeds the friction constraints. Paths that violate these constraints are discarded, ensuring that only feasible trajectories are considered in the planning process. This is explained in Algorithm 1.

Algorithm 1 Path Generation

```
for each steering do
    for each acceleration do
        next step ← KBM(prev. step)
        if CheckFrictionConstraints then
            continue ▷ Skip to next iteration if constraints are violated
        else
            Add the point to the path
        end if
    end for
end for
```

3.1.2 Correspondences Generation. Once the paths are generated, they are overlaid onto a grid designed to accommodate the dimensions of the longest path. During this process, each point on every path is evaluated. If the distance from a point to the center of a grid cell is less than half the length of the longest side of the robot (representing the robot's footprint), that path's unique identifier is recorded in the cell. The grid utilizes an unordered map to efficiently link path IDs with their corresponding cells.

Accounting for the robot's size is essential, as ignoring it can lead to paths that seem clear but result in collisions. Integrating the footprint measurements into the grid mapping offline eliminates the need for ongoing online adjustments and avoids the computationally intensive obstacle inflation process.

Furthermore, these path-cell correspondences are stored in a text file and loaded a single time at runtime, streamlining the process. This setup also allows users the flexibility to generate numerous paths as needed. Since paths are associated with fixed-dimension grid cells, collision checking during runtime is confined to the grid's dimensions and not to every individual point on each path.

These correspondences are loaded during runtime when the planner starts, setting the stage for the online phase of the planner, which is detailed in the subsequent section.

3.2 Online Phase.

3.2.1 Initial Setup. The planner operates at 10Hz, providing a path every 0.1 seconds. When a waypoint is assigned to the robot via RViz or a joystick, the first step involves setting the robot's position. Once the position is established, the planner determines

the direction in which the robot should move to reach the waypoint. If the angle difference between the robot's orientation towards the waypoint and its yaw angle is less than 90 degrees, the robot is directed to move backward. If not, the robot maintains the input target velocity.

3.2.2 Collision Checking. The planner receives the obstacle map of the environment as input, which updates every 0.1 seconds. For collision checking, the grid of offline-generated correspondences is transformed into the frame of the obstacle map and overlaid onto it. Ensuring that the resolution of the grid does not exceed that of the obstacle map is crucial; a higher resolution grid increases the likelihood of path cancellations because a single grid cell may overlap with multiple cells on the obstacle map. This might lead to the cancellation of a path even though only a small portion of the path overlaps with an obstacle on a lower-resolution map. The resolution for both the grid and the map is maintained at 0.15 meters to optimize the accuracy of the collision detection. Cells of the grid that overlap with filled cells on the obstacle map (indicating the presence of an obstacle) result in the cancellation of corresponding paths. This is illustrated in Figures 2 and 3.

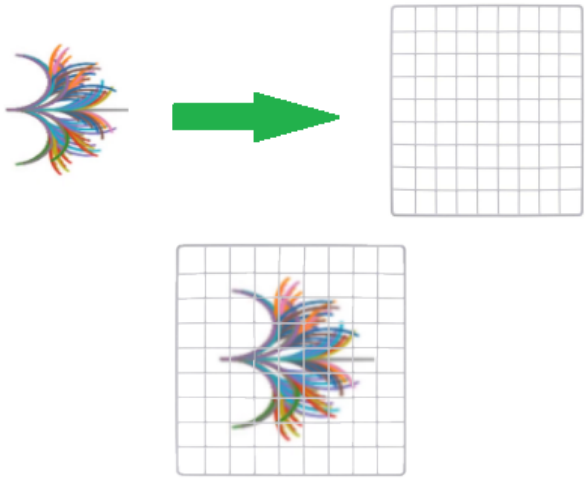

Fig. 2 The paths are overlaid on a grid for efficient collision-checking during the online phase.

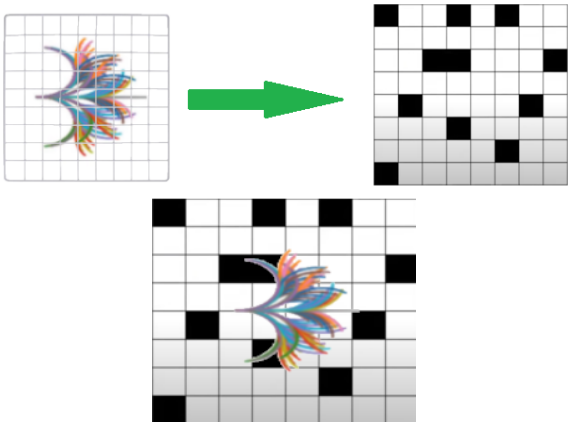

Fig. 3 Transforming and overlaying the correspondences grid on the obstacle map enhances obstacle detection efficiency and speed.

3.2.3 Path Scoring. After the collision check, the remaining viable paths are assessed using a cost function to determine the optimal path. Paths that intersect obstacles are discarded, ensuring that only feasible routes are considered for evaluation.

The paths are evaluated using the following cost function:

$$f = \alpha \cdot d + \beta \cdot a + \gamma \cdot p$$

where:

- d represents the Euclidean distance between the robot and the waypoint,
- a denotes the angle of deviation between the robot and the waypoint,
- p measures the degree of persistence of a path, which helps to maintain consistency in the chosen route and avoid frequent path switching.

The coefficients α, β, and γ are weights assigned to each term, balancing distance, alignment, and persistence, respectively. Adjusting these weights is crucial: while a high value for γ increases path consistency, it must be carefully tuned to prevent excessive rigidity, allowing the planner flexibility to switch paths when a better path becomes available.

According to this function, the path yielding the lowest cost is selected as the best path. This path is then forwarded to the iLQR controller for execution. The offline and online algorithm is detailed in Algorithms 2 and 3.

Algorithm 2 Offline Phase

```
Generate paths using Kinematic Bicycle Model (KBM):
for each steering angle and acceleration do
    Calculate the next states based on previous states using KBM
    Check lateral acceleration against friction constraints for the new states
    if constraints violated then
        Skip path and continue to the next iteration
    else
        Add point to the current path
    end if
end for
Overlay paths on a grid and generate correspondences
for each point in each path do
    if distance to grid cell center < half robot's length then
        Record path ID in grid cell
    end if
end for
Store path-cell correspondences in a text file
```

4 Results

Path Generation

Figure 4 shows paths generated for 2m/s and 6m/s for 2 seconds using the kinematic bicycle model. Noticeably, some paths begin to curve at specific points due to inflection points where the steering angle increases. This feature allows for more curved paths, enhancing the robot's ability to execute sharp turns. Furthermore, with approximately 300 paths per speed, the planner achieves precise performance, adapting flexibly to different driving requirements.

Correspondences Generation and Collision Checking

The path correspondences generated at a speed of 2m/s are illustrated in Figure 5. Once overlaid on the obstacle map, the correspondence grid effectively highlights the obstacles detected in blue. This visualization confirms that the obstacles are accurately and efficiently identified through the transformation and overlay process.

Algorithm 3 Online Phase

```
Load path-cell correspondences into the planner
Every 0.1 seconds (10Hz):
while true do
    Determine motion direction:
    if angle difference < 90 degrees then
        Set velocity to -1 times target velocity to move backward
    else
        Maintain target velocity
    end if
    Perform collision checking by overlaying grid on obstacle map:
    for each cell in the grid do
        if cell overlaps with obstacle then
            Cancel corresponding paths
        end if
    end for
    Evaluate remaining paths using a cost function
    Select path with minimum cost and pass to iLQR controller
end while
```

Simulation Setup and Benchmarking

Simulations were conducted using Gazebo to evaluate the performance of our proposed planning algorithm and the trajectory library was benchmarked against Fast Likelihood-based Collision Avoidance (FALCO)[13]. Additionally, the cross-track error performance of the Iterative Linear Quadratic Regulator (iLQR) controller was analyzed across both planners to determine their precision in trajectory following.

4.1 Implementation Details. The trajectory library operates at a frequency of 10Hz, dynamically responding to changes in the environment by adjusting the vehicle's trajectory based on the latest obstacle maps received from the perception stack at the same rate. The selected optimal path is then processed by the iLQR controller, which runs at a higher frequency of 50Hz, ensuring responsive control adjustments. To keep it uniform, FALCO also runs at 10Hz.

4.2 Simulation Environment. The simulations were executed on Gazebo using the MIT-racecar world setup, chosen for its accurate representation of typical indoor environments, particularly corridors prone to complex navigational challenges. The simulations have been carried out on an Intel i7 CPU (4.7GHz). This setting provided a robust platform for testing and validating the adaptability and effectiveness of the planning algorithms under the same conditions. The simulation environment is shown in Figure 6.

4.3 Simulation Results.

4.3.1 Cost Function Analysis in Relation to Path Quality. The effectiveness of the trajectory library is quantified primarily through its cost function, which is formulated based on three critical factors:

- **Distance from the Goal:** This metric measures the Euclidean distance to the goal from the current position.
- **Deviation from the Goal:** Assesses how much the current path deviates from the goal.
- **Degree of Persistence of a Path:** Determines how consistently a path is maintained before switching to an alternative path. The persistence value is set at 0.001. A very high persistence value would restrict the planner's ability to switch paths when necessary, while a non-existent persistence value would cause frequent, unnecessary path switches.

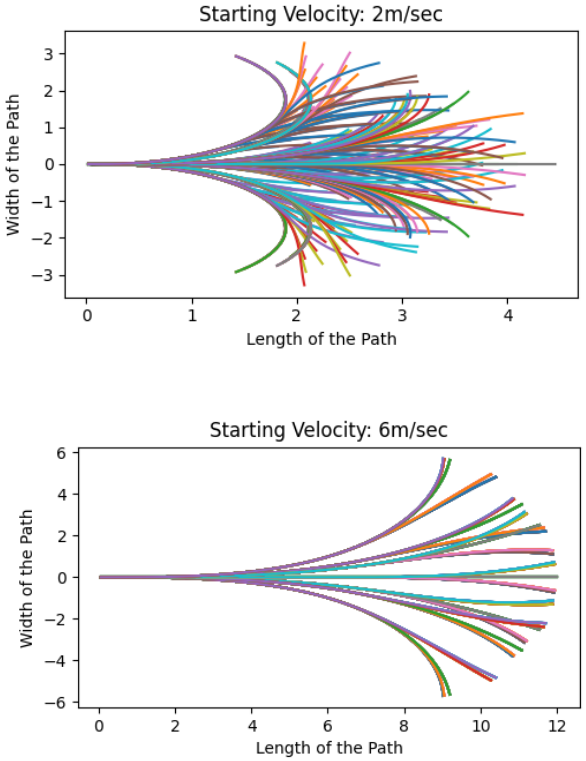

Fig. 4 Paths generated using the kinematic bicycle model for 2m/s and 6m/s that have been generated for different steering angles and accelerations.

The performance impact of these factors was evaluated in simulation at a constant speed of 4m/s, focusing on the interaction between the weighting coefficients, α (for deviation) and β (for distance), of the cost function for the values shown in Table 1.

Table 1 Hyperparameter Testing

Test	α	β	Comments
1	2	0.5	Increased α sharpens turns, prioritizes directness, risks aggressive steering.
2	1	1	Reducing α enables wider, less aggressive turns.
3	0.5	2	Further reducing α results in even wider turns.

(1) **Test 1:** With a higher weight on minimizing the angle of deviation (α increased), the resulting paths during turns were observed to be sharper. This configuration prioritizes a more direct approach to the goal, reducing lateral deviations but potentially leading to aggressive steering maneuvers. This is illustrated in Figure 7.

(2) **Test 2:** By reducing α slightly, allowing for a greater degree of deviation, the vehicle executed wider turns. This adjustment made the path smoother and less aggressive compared to Run 1, trading off some directness for gentler turns, which might be preferable in some scenarios. This is shown in Figure 8.

(3) **Test 3:** A further reduction in α continued the trend observed, producing even wider turns. This setting emphasizes comfort and stability over the shortest path, showcasing the planner's adaptability to different driving styles and requirements. This is shown in Figure 9.

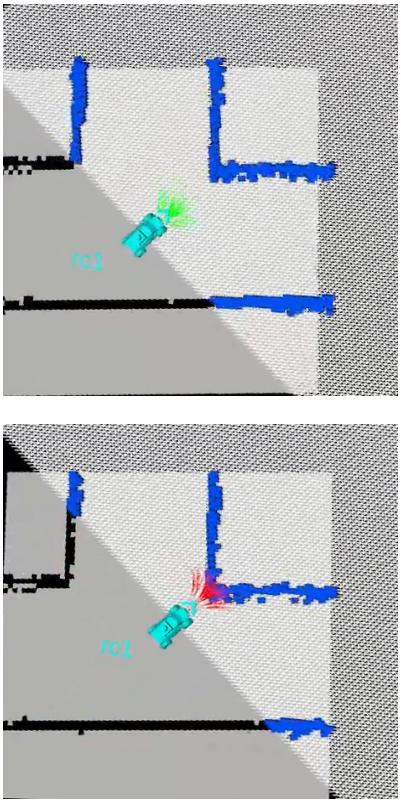

Fig. 5 The white grid in the figure represents the generated path correspondences. This grid is transformed and overlaid on the obstacle map, with overlapping obstacles highlighted in blue. The transformation and overlay effectively detect the obstacles, demonstrating the accuracy of the obstacle mapping process.

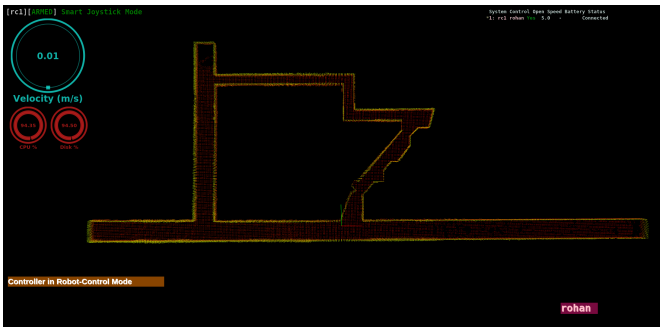

Fig. 6 The simulation environment depicted is used to test and benchmark both planning algorithms, offering a complex setting that mimics real-world conditions to evaluate their performance and effectiveness.

*4.3.2 **Quantitative Comparison between Trajectory Library and FALCO**.* To validate the trajectory library's effectiveness and adaptability, specific test scenarios were designed focusing on the algorithm's maneuverability in constrained spaces. The scenarios included navigating turns of 50°, 90°, and 110°—angles that are particularly challenging and test the precision of any robot.

The planners were assessed based on the following key performance indicators:

- **Success Rates in Making Turns:** This metric evaluates the ability of each planner to successfully execute turns of the specified degrees at various speeds. Success rates are critical for understanding the operational reliability of each algorithm under dynamic conditions.
- **Directional Error at Higher Speeds:** As the vehicle's speed increases, maintaining accuracy in following the operator's input becomes more challenging. This measure helps quantify how much deviation the planner shows as compared to the user's input.
- **Cross Track Error (CTE) of the iLQR Controller:** While the vehicle traverses a set distance of 30 meters at higher speeds, the CTE measures the deviation from the planned path. This is crucial for assessing the precision of the path provided by the iLQR controller, which optimizes path following in real time.

The success rates of path planning algorithms FALCO and the

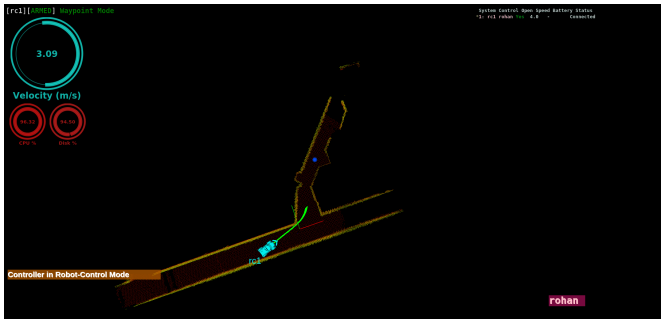

Fig. 7 Increasing the weight on minimizing the angle of deviation (α) results in sharper turns, enhancing a direct approach to the goal but may lead to aggressive steering.

Fig. 8 Reducing the weight on minimizing the angle of deviation (α) allowed for wider, smoother turns, making the vehicle's path less aggressive and potentially more suitable for scenarios requiring gentler maneuvers.

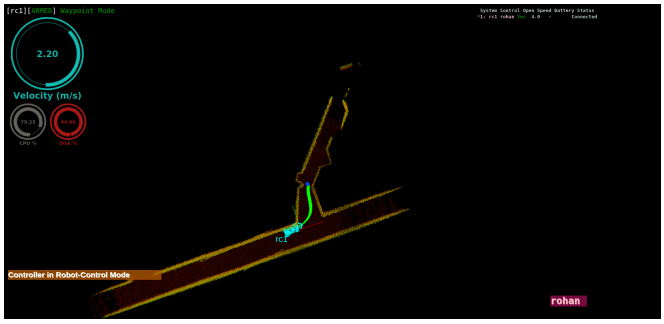

Fig. 9 Continuing to decrease the weight on angle deviation (α), the planner produced even wider turns, emphasizing stability over direct paths, demonstrating adaptability to various driving needs and styles.

trajectory library are distinctly illustrated in Figure 10. These tests were conducted across different turn angles—50 degrees, 90 degrees, and 110 degrees—as specified in the simulation environment above. At a baseline speed of 2 m/s, both algorithms demonstrate competency in handling 50-degree turns. However, at increased speeds, the trajectory library consistently surpasses FALCO, showcasing superior performance in executing sharp turns. This pattern persists with more complex maneuvers: for 90-degree turns, the trajectory library slightly outperforms FALCO at lower speeds and significantly excels as the speeds increase. The trend continues with 110-degree turns, where the trajectory library notably outperforms FALCO, particularly at higher speeds.

This performance disparity stems largely from the differing methodologies in cost function formulations between the two planners. FALCO's cost function prioritizes safety by selecting paths that are surrounded by other potential routes, theoretically ensuring that the chosen path is secure. This approach, while safety-focused, proves restrictive when fewer paths are available—for instance, it may overlook an optimal solitary path because it does not meet the clustering criteria.

Conversely, the trajectory library evaluates paths based on their proximity to the goal and the extent of their deviation from the intended route. This more straightforward assessment allows the trajectory library to commit to an optimal path, thus providing flexibility and efficiency in path selection, particularly in scenarios where minimal paths are viable.

The scoring system used for the paths significantly influences the directional error. For instance, if an operator instructs FALCO to execute a sharp left, the planner might not prioritize that path if a straighter path scores higher. In contrast, the trajectory library is more responsive to the operator's commands, allowing for adjustment of the planner's aggressiveness to suit the operator's comfort. This responsiveness is evident in the planners' performance at high speeds, where the trajectory library exhibits lower directional error on average as compared to FALCO, as shown in Table 2.

Table 2 Average Directional Error at 4m/s and 6m/s for different angles.

Speed	Angle	Trajectory Library	FALCO
4 m/s	50 degree	8.02	13.98
	90 degree	0.43	2.01
	110 degree	3.56	6.02
6 m/s	50 degree	9.6	27.05
	90 degree	2.1	20.6
	110 degree	9.5	11.6

Once a path is generated by the planner, it is passed to the iLQR controller, which tracks the input path while minimizing the cross-track error. A comparative analysis of cross-track errors revealed that paths from the trajectory library consistently exhibit lower errors than those from FALCO. This improved performance is attributed to the trajectory library's use of the Kinematic Bicycle Model (KBM) for path generation, which aligns with the iLQR controller's method for executing the forward pass. Since both the planner and the controller derive paths from the same model, their alignment significantly reduces discrepancies in path tracking. In contrast, the paths of FALCO are generated using splines, which are not kinodynamically feasible.

This is demonstrated in Figure 9. The cross-track error has been compared for high speeds(at 6m/s) for 50-degree, 90-degree, and 110-degree turns over 30m paths. It can be seen that the trajectory library has lesser cross-track error overall than FALCO while moving at 6m/s.

This is because the trajectory library provides precise velocity and angle data for each path point, eliminating the need for the controller to engage in velocity profiling. In contrast, FALCO generates paths using spline interpolation, which does not guarantee kinodynamic feasibility for each point. Consequently, the controller must attempt to approximate a path that closely follows the spline, often resulting in higher cross-track errors.

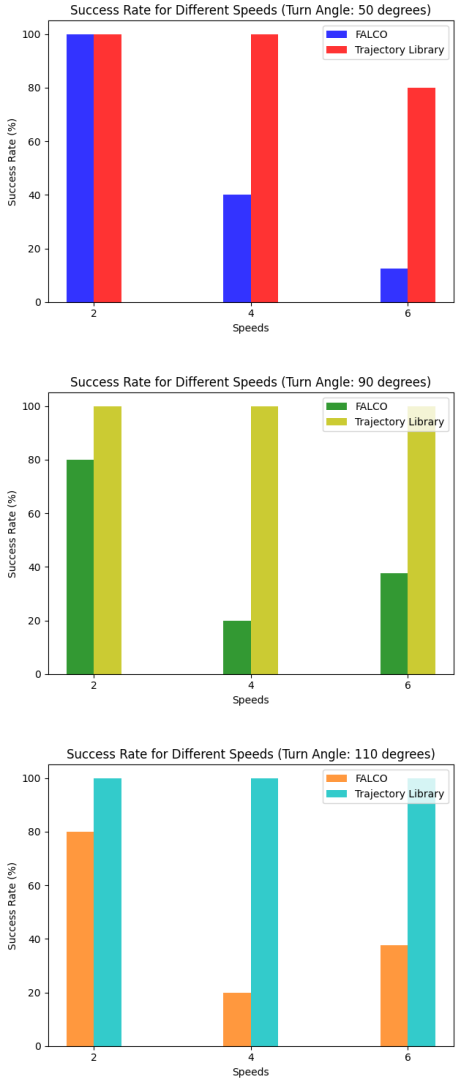

Fig. 10 In testing across turn angles of 50, 90, and 110 degrees, the trajectory library consistently outperforms FALCO, particularly at higher speeds, demonstrating its ability to execute sharp turns efficiently.

Hardware Setup and Benchmarking

The experimental setup for testing both algorithms included an RC car equipped with a Velodyne VLP16 LiDAR, NVIDIA Jetson AGX Xavier for processing, an Epson IMU for motion tracking, a Pixhawk for navigation control, and a VESC for electronic speed control. This is shown in Figure 12.

4.4 Implementation Details. Just like the simulation, the trajectory library operates at 10Hz, and the iLQR controller runs at 50Hz. To keep it uniform, FALCO also runs at 10Hz.

4.5 Hardware Environment. The testing was done at Carnegie Mellon University in the Newell-Simon Hall. The environment, depicted in Figure 10, features two narrow corridors branching off from a central straight corridor. To access these side corridors from the main one, the car must execute a 90-degree turn.

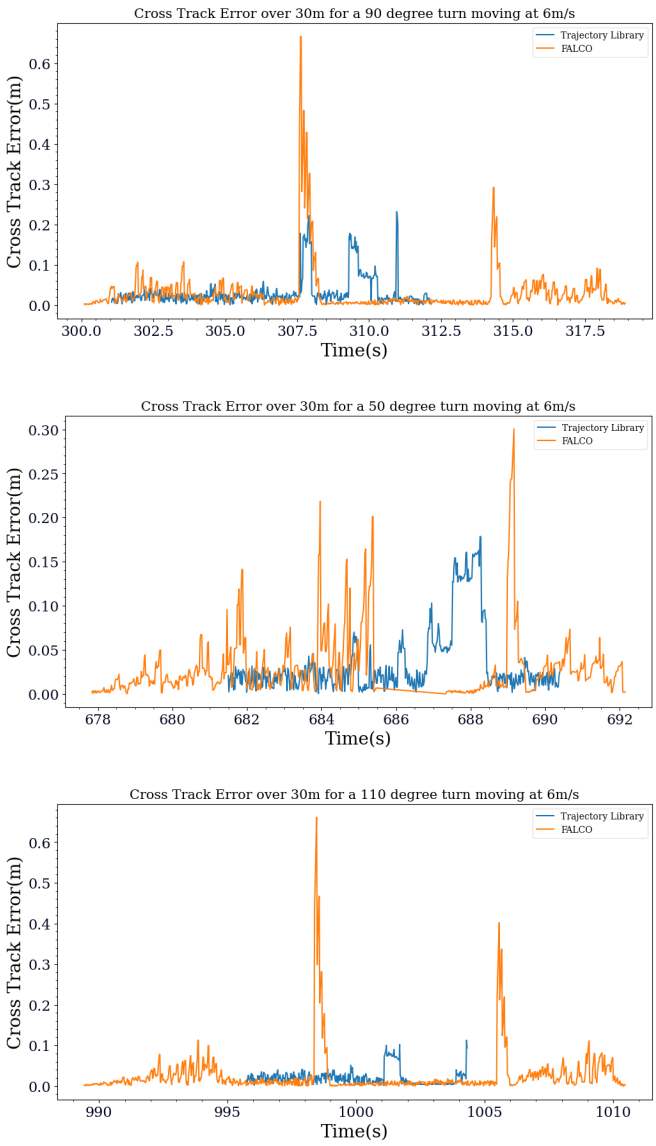

Fig. 11 At 6m/s speeds, the trajectory library consistently achieves lower cross-track errors in 50, 90, and 110-degree turns over 30 meters when compared to FALCO.

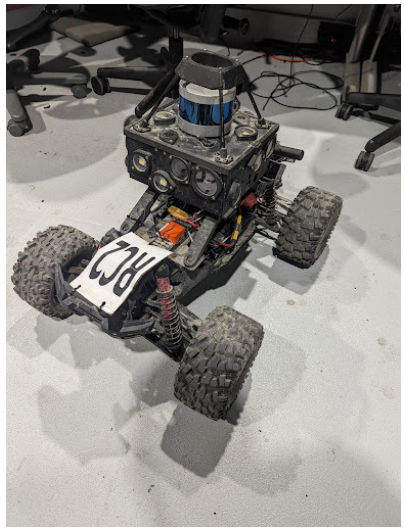

Fig. 12 The RC car used to test and validate both planners.

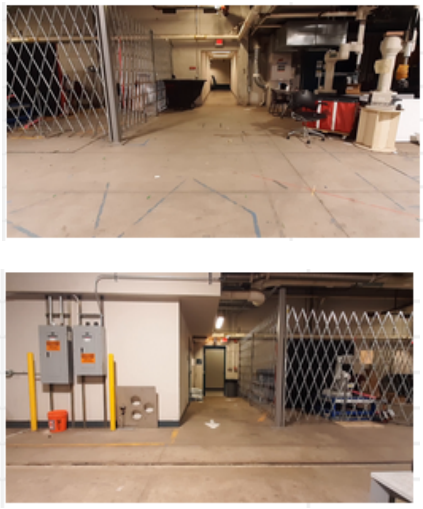

Fig. 13 Th two 90-degree corridors used for testing and validating the trajectory library.

4.6 Hardware Results.

4.6.1 ***Cost Function Analysis in Relation to Path Quality.*** The real-world efficacy of the planning algorithm was qualitatively analyzed through its cost function, which remained unchanged from the simulations to facilitate direct comparisons. The same three key metrics that were evaluated in the simulation were evaluated.

The impact of these factors was assessed by moving the robot at a commanded speed of 4m/s, evaluating how the real-world dynamics affect the theoretical models used in the simulations. Results from this analysis were then compared against the simulation data to validate the effectiveness of the algorithm under realistic operational conditions. This is demonstrated in Figure 14.

(1) **Test 1:** Evaluating the impact of minimizing the angle of deviation, resulting in sharper turns executed by the robot, consistent with simulation outcomes.
(2) **Test 2:** Observing wider turns with a slight reduction in α, in line with simulated behavior.
(3) **Test 3:** Continuing the trend with further reduction in α, leading to even wider turns, again being consistent with simulation results.

These real-world tests provided crucial insights into the practical application of the planning algorithms.

4.6.2 ***Quantitative Comparison between Trajectory Library and FALCO.*** To validate the effectiveness and adaptability of the planning algorithm, specific test scenarios were designed focusing on its maneuverability in constrained spaces. The scenarios included navigating multiple corridors with 90° turns. The planner was tested on the same three parameters that were used to test in the simulation.

The comparative success rates of the two planners are depicted in Figure 11. It is evident that at lower speeds, the trajectory library achieves a slightly higher success rate in executing 90-degree turns compared to FALCO. However, at higher speeds, the trajectory library significantly outperforms FALCO. This disparity is largely attributable to the differences in how their cost functions are structured, particularly in how the trajectory library's cost function facilitates making sharp turns at increased velocities. Additionally,

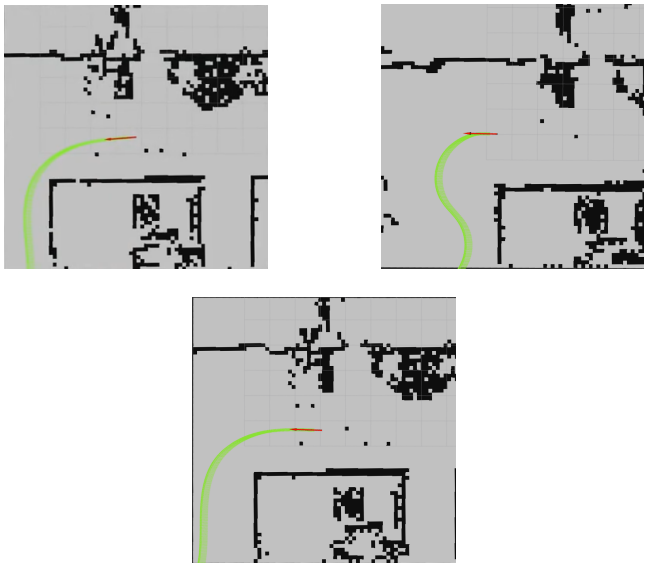

Fig. 14 Variations in path trajectories occur relative to adjustments in the weights assigned to distance and steering angle metrics.

it is observed that both planners exhibit slightly lower performance in hardware than in simulations. This can be attributed to various factors affecting steering and velocity commands at the hardware level, such as the VESC, Pixhawk, and servos. Therefore, proper tuning of these components is crucial for optimizing performance.

Similar to the simulations, the trajectory library displays less directional error than FALCO while making 90-degree turns. This follows the simulation results and also relates to the way the cost function is defined. This is shown in Table 3.

The hardware trials demonstrated a consistent trend with simulation results: the trajectory library exhibited lower cross-track error compared to FALCO for 6m/s over a 30m distance, as depicted in Figure 16. This observation confirms the initial hypothesis regarding the efficiency of the trajectory library in real-world testing.

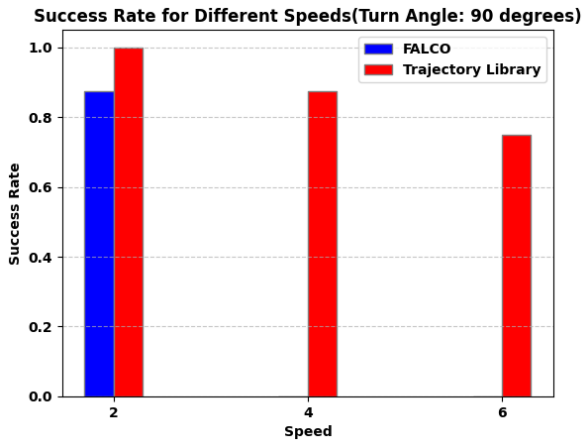

Fig. 15 The trajectory library shows higher success rates in hardware testing compared to FALCO.

5 Conclusion

In conclusion, this work has effectively demonstrated the viability and efficiency of a trajectory-library-based local planner tailored for high-speed, non-holonomic autonomous ground

Table 3 Average directional error at 6 m/s for two 90-degree corridors.

Speed	Angle	Trajectory Library	FALCO
6 m/s	90 degree-Corridor 1	14.6	32.98
	90 degree-Corridor 2	12.3	29.7

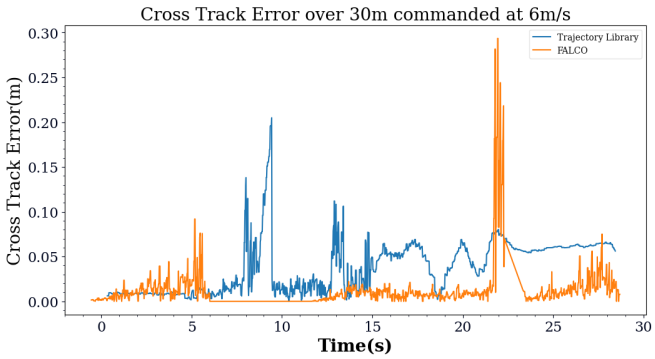

Fig. 16 At 6m/s, the iLQR exhibits lower cross-track error when following paths generated by the trajectory library compared to those from FALCO, consistent with our simulation findings.

robots navigating through cluttered environments. By leveraging pre-generated motion primitives and incorporating an innovative collision-checking mechanism, our approach significantly enhances the performance of the robot. The experimental results obtained from both simulated environments and real-world tests with an RC car demonstrate the planner's effectiveness in handling complex navigation tasks. Additionally, the comparative analysis with existing planning methods highlights our planner's superior performance in reducing computational load and improving navigation accuracy.

Acknowledgment

I would like to thank Dr.Matthew Travers and Dr.Bhaskar Vundurthy for their continuous guidance and support.

References

[1] Viswanathan, V. K., Dexheimer, E., Li, G., Loianno, G., Kaess, M., and Scherer, S., 2020, "Efficient Trajectory Library Filtering for Quadrotor Flight in Unknown Environments," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2510–2517.

[2] Khatib, O., 1986, "Real-time obstacle avoidance for manipulators and mobile robots," International Journal of Robotics Research, **5**(1), pp. 90–98.

[3] Fox, D., Burgard, W., and Thrun, S., 1997, "The dynamic window approach to collision avoidance," IEEE Robotics and Automation, **4**(1).

[4] Kavraki, L., Svestka, P., Latombe, J., and Overmars, M., 1994, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," Stanford University.

[5] LaValle, S. M., 1998, "Rapidly-exploring random trees: a new tool for path planning," *The annual research report*.

[6] Kuffner, J. J. and LaValle, S. M., 2000, "RRT-connect: An efficient approach to single-query path planning," *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, pp. 995–1001, doi: 10.1109/ROBOT.2000.844730.

[7] Lim, W., Lee, S., Sunwoo, M., et al., 2018, "Hierarchical trajectory planning of an autonomous car based on the integration of a sampling and an optimization method," IEEE Transactions on Intelligent Transportation Systems, **19**(2), pp. 613–626.

[8] Rösmann, C., Hoffmann, F., and Bertram, T., 2015, "Timed-Elastic-Bands for time-optimal point-to-point nonlinear model predictive control," *2015 European Control Conference (ECC)*, pp. 3352–3357, doi: 10.1109/ECC.2015.7331052.

[9] Dijkstra, E. W., 1959, "A note on two problems in connexion with graphs," Numerische mathematik, **1**(1), pp. 269–271.

[10] Hart, P., Nilsson, N., and Raphael, B., 1968, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transactions on Systems Science and Cybernetics, **4**(2), pp. 100–107.

[11] Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., and Thrun, S., 2005, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, The MIT Press.

[12] Cowlagi, R. V. and Tsiotras, P., 2011, "Hierarchical motion planning with dynamical feasibility guarantees for mobile robotic vehicles," IEEE Transactions on Robotics, to appear.

[13] Zhang, J., Hu, C., Chadha, R., and Singh, S., 2020, "Falco: Fast likelihood-based collision avoidance with extension to human-guided navigation," J Field Robotics, **37**, pp. 1300–1313.

List of Figures

List of Tables