Adaptive Trajectory Library Planner for Fast Outdoor Robots

Nguyen Thanh Trung Le, Edward Bray, Ki Myung Brian Lee, Graeme Best
University of Technology Sydney, Australia
{nguyenthanhtrung.le, edward.bray, kmbrian.lee, graeme.best}@uts.edu.au

Abstract

High-speed autonomous operation in outdoor environments requires fast computation of dynamically feasible, collision-free paths. To this end, we propose a new local path planning algorithm called *Adaptive Trajectory Library*. This approach relies on the selection of a suitable trajectory from a precomputed library to reduce online computation. A subset of trajectories that are dynamically feasible are considered, from which one is chosen to best move the robot towards a goal location while avoiding obstacles. In simulated experiments, the proposed algorithm significantly outperforms the Dynamic Window Approach [Fox *et al.*, 1997] with 84.2 % less travel time and 39.5 % shorter path length. Hardware experiments in outdoor environments show that the proposed planner is able to reliably compute paths for a robot to follow to reach goals at high speeds in off-road terrain while avoiding obstacles.

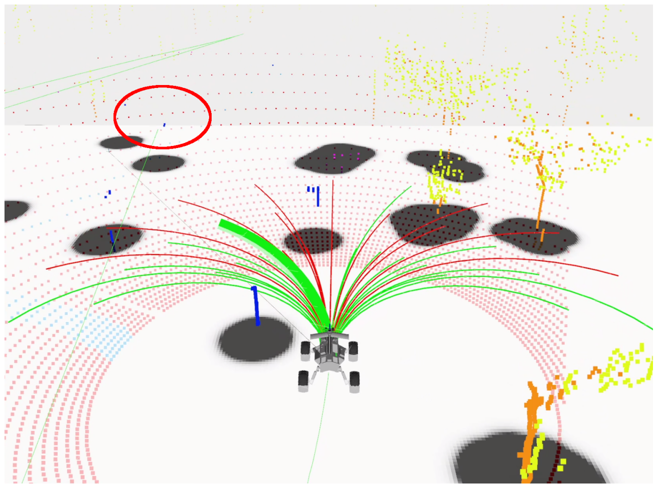


Figure 1: *Adaptive Trajectory Library* (ATL) in action as the robot moves to the red circled waypoint. Obstacles are detected by a lidar and represented on a 2D costmap (black). A dynamically feasible subset of the library is considered, and trajectories that result in collision are discarded (red). A cost function is evaluated for the collision-free trajectories (green), and the robot moves along the one with the lowest cost (thick green).

1 Introduction

Autonomous robots that move at high speeds through outdoor environments could offer huge benefits in a variety of scenarios, including agriculture and defence. Such robots would be able to effectively operate in much larger areas than traditional systems that operate indoors at low speeds, increasing the range of tasks they could complete. The recent boom in electric vehicle technology has increased the availability of crucial components such as high torque electric motors, meaning high-speed ground-based autonomous robots can now be constructed by researchers with modest budgets. Despite their potential, deploying these robots presents new challenges compared to traditional systems.

Robots operating autonomously in real-world environments, both indoors and outdoors, must be able to safely navigate their environment. A crucial component of such autonomous navigation is *local path planning*, where a robot must decide how to reach a given goal location while obeying certain constraints, such as avoiding obstacles or choosing a short route [Karur *et al.*, 2021]. The local path planner chooses which trajectory the robot should follow at any instant [Coulter, 1992; Fox *et al.*, 1997; Rösmann *et al.*, 2012].

Local path planning for a high-speed outdoor robotic vehicle such as the one in Figure 2 operating in an unstructured outdoor environment is more demanding than for traditional indoor robots [Likhachev and Ferguson, 2009]. An obvious challenge associated with controlling such vehicles is to minimise the time taken for computation: since the robot is expected to achieve high velocities while avoiding any obstacles, accurate decisions

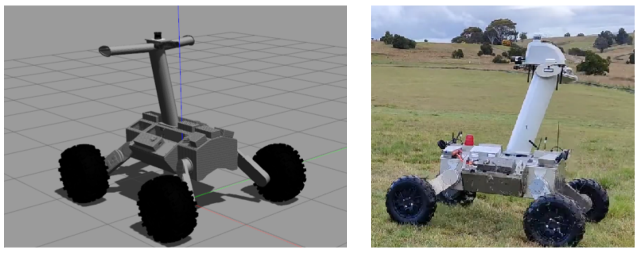

(a) (b)

Figure 2: The Fast Off-Road Vehicle (FORV) developed at UTS in (a) simulation and (b) hardware.

must be made quickly. Increasing the speed of a robot will increase its inertia, thus reducing its responsiveness. A local planner for such scenarios should ensure planned trajectories are feasible to execute given the dynamics and current state of the robot. Furthermore, outdoor operation presents difficulties not experienced in structured indoor arenas, such as unpredictable terrain and changes to the environment during operation [Chu *et al.*, 2012; Goswami, 2017].

This paper propose a new local planner for a fast outdoor robotic vehicle, called *Adaptive Trajectory Library* (ATL, Figure 1). A library of possible trajectories is first precomputed, containing combinations of linear and angular velocities that can be achieved by the robot. At each planning instant, a subset of the library is considered which is adapted to contain only those trajectories that are feasible to achieve given the current state of the robot. A trajectory is selected from this subset that is the most suitable at this time, considering the goal location and obstacle avoidance. This approach manages computational resources more efficiently than calculating possible trajectories online, while ensuring that selected actions are actually feasible. The responsiveness of the robot is therefore increased compared to other methods, even when travelling at high speed.

We apply our approach to the Fast Off-Road Robotic Vehicle developed at UTS (Figure 2). We compare the performance of the proposed ATL planner to the existing DWA planner [Fox *et al.*, 1997] in realistic simulation experiments as illustrated in Figure 3, and show that the ATL planner offers 84.2 % reduction in travel time and 39.5% in path length owing to explicit consideration of dynamic feasibility, such as excluding spot turns. We then demonstrate the effectiveness of ATL in real-world operation through experiments with a full-scale robot in an outdoor environment with uneven terrain containing several obstacles.

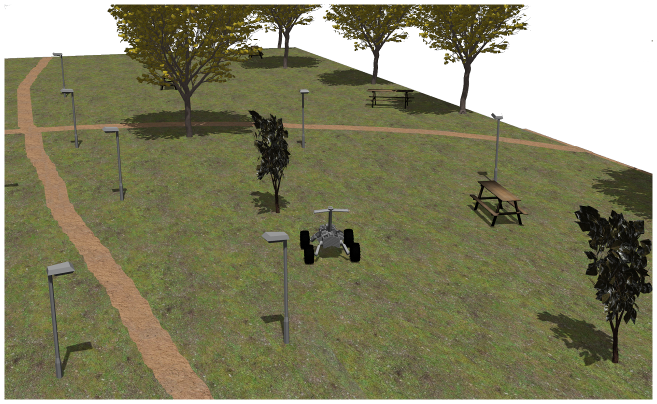


Figure 3: FORV within the NEGS-UGV park terrain used in Gazebo simulations [Sánchez *et al.*, 2022].

2 Related Work

Local path planning has been the subject of research for decades. One of the earliest methods proposed was Artificial Potential Fields [Khatib, 1986], in which obstacles are modelled as repulsive fields and goal locations as attractive fields. Robots follow descending gradients in the resultant of these fields to reach the goal. This approach gives good obstacle avoidance, but robots can get stuck in local minima.

Another traditional method of path planning is Pure Pursuit (PP) [Coulter, 1992]. Robots following this scheme move at a constant forward velocity and the controller calculates instantaneous angular velocities to move the robot to a point a specified distance ahead of it. Extensions to PP, including Adaptive Pure Pursuit [Campbell, 2007] and Regulated Pure Pursuit [Macenski *et al.*, 2023], improve this algorithm by allowing the robot to vary its linear velocity, reduce sharp turns, and react to newly-observed obstacles. PP controllers consider only the geometry of the current scenario without taking the dynamics into account, so are not suitable for robots with significant dynamic feasibility constraints.

The popular DWA algorithm considers the dynamics of the robot during planning [Fox *et al.*, 1997]. Here, the potential paths resulting from a range of linear and angular velocities are simulated over a short time period. A suitable instantaneous trajectory is chosen by assigning a cost to each that trades off between the distance to obstacles and how closely the given path is followed. DWA has been found to perform well at calculating dynamically feasible motion plans relatively quickly, but online computation of the paths takes time, and can lead to poor or unpredictable path selections when run at the high frequencies required for high-speed robots.

Optimisation-based methods such as TEB [Rösmann *et al.*, 2012; 2017] have also been applied to local path planning. TEB uses least-squares optimisation to deform a commanded path to avoid obstacles while respecting dynamic feasibility. However, similar to DWA the solution quality is subject to trade-off against computation time, thus is not suitable for high-speed operations.

To increase the responsiveness of motion planners, researchers have recently explored the use of a library of dynamically feasible trajectories that is generated offline. Such libraries of collision free, dynamically feasible trajectories have been used to efficiently calculate paths for robotic manipulators to achieve repetitive tasks, such as removing weeds from fields [Lee *et al.*, 2014] or pruning trees [You *et al.*, 2020]. A similar approach has also been applied to quadrotor aerial robots [Best *et al.*, 2023]: given the commanded path from an upstream planner, a path planning algorithm selects the closest collision-free trajectory from a precomputed library that is also dynamically feasible given current state of the robot. In this paper, we explore how this principle could be applied to a ground robot moving at high speeds.

Graph search methods are also commonly applied to robot path planning. These approaches typically begin with generating a *Probabilistic Roadmap* (PRM) [Kavraki *et al.*, 1996], a graph where nodes represent possible robot states, and edges are added between pairs of nodes within a set distance if there is a collision-free path. Algorithms such as that of Dijkstra [Dijkstra, 1959], A* [Hart *et al.*, 1968], D* [Stentz, 1994] are then used to find paths through the PRM that the robot can follow. Other popular methods of generating and searching trees include the Rapidly Exploring Random Trees [LaValle and Kuffner Jr, 2001] and Fast Marching Tree [Janson *et al.*, 2015] algorithms. These methods are well established and yield optimal solutions with respect to the distance travelled in reaching the goal. However, the construction of the graph is computationally expensive, thus reducing its suitability for robots operating at high speeds where decisions must be made quickly. Two dimensional graphs are commonly used, but capturing constraints such as ensuring dynamic feasibility requires constructing the graph in higher state spaces. The search problem quickly becomes intractable as the PRM grows with larger spatial scale or with additional constraints.

The high computational cost of graph search methods can be reduced by minimising the size of the graph. This can be done by modelling the environment as a multi-resolution lattice state space, so the search can be performed at different scales to increase efficiency [Likhachev and Ferguson, 2009]. However, this approach relies on heuristics which can be hard to tune, and remains computationally expensive for the graphs necessary to operate in very large areas or with several dynamic feasibility constraints. Another common approach is to use a separate global planner to find a path through a low density graph, and assign the low-level navigation to a suitable local path planning algorithm such as those mentioned previously [Best *et al.*, 2023].

3 Problem Formulation

We consider a robot described by an N-dimensional state $\mathbf{x}_t$ (e.g., position and orientation) and an M-dimensional control action $\mathbf{u}_t$ (e.g., translational and rotational velocity command) at each time t. Feasible states and control actions are limited to subsets $\mathbb{X} \subset \mathbb{R}^N$ and $\mathbb{U} \subset \mathbb{R}^M$ respectively. The state $\mathbf{x}_t$ and the control actions $\mathbf{u}_t$ are governed by a dynamic model $\mathbf{f}$:

$$\dot{\mathbf{x}}_t = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t). \tag{1}$$

Importantly, in addition to the limitations modelled by feasible state- and control-spaces $\mathbb{X}$ and $\mathbb{U}$, the robot is subject to a dynamic feasibility constraint:

$$\mathbf{u}_t \in \mathcal{D}(\mathbf{x}_t, \dot{\mathbf{x}}_t). \tag{2}$$

This can model a variety of constraints ranging from simple ones such as turning radius and acceleration limits in a bicycle model, to complex ones such as slip prevention for off-road operation. In addition to dynamic constraints, the robot operates in an environment with obstacles to avoid, denoted by $\mathcal{O} \subset \mathbb{X}$.

We are given a desired path $\mathbf{X}^d = \mathbf{x}_t^d \ldots \mathbf{x}_{t+T}^d$ from a higher-level planner that may not be collision-free due to factors such as a slow update rate. The overall aim of this paper is to plan a minimum-time path $\mathbf{X}$ that satisfies the desired path $\mathbf{X}^d$ while avoiding collisions and respecting the dynamic feasibility constraint:

$$\begin{aligned} \min_{\mathbf{X}} \quad & \text{travel time} \\ \text{s.t.} \quad & \text{dynamics (1)}, \\ & \text{dynamic feasibility (2)}, \\ & \mathbf{X} \cap \mathcal{O} = \emptyset \\ & \mathbf{X} \text{ visits } \mathbf{X}^d \end{aligned} \tag{3}$$

The main challenge in solving (3) is the dynamic feasibility constraint (2). The set of available control actions depends on the state, which introduces significant complications because states are also dependent on control actions. Despite such co-dependence, the planner must be robust against inaccuracies or even a lack of a model of the dynamic feasibility constraint (2), as may be the case for practical robots.

4 Adaptive Trajectory Library Planner

We introduce the ATL planner as an approximate solver for (3). This approach begins with precomputing an offline library of dynamically feasible trajectories as described in Section 4.1. The planner leverages this library to rapidly find the best collision-free path during operation as described in Section 4.2. The retrieval of trajectories *adapts* to the current state to ensure dynamic feasibility. Section 4.3 outlines some particular considerations when implementing the planner on a robotic platform.

4.1 Offline Library Construction

We define the offline library $\mathcal{T}$ as a set of *collections* C, where each collection is a tuple $C = (\mathcal{F}^C, \mathcal{X}^C, \mathbf{U}^C)$ comprising a feasible region $\mathcal{F}^C \subset \mathbb{X}$, a finite set of state trajectories $\mathcal{X}^C$, and a finite set of control actions $\mathbf{U}^C$. If the robot is in the feasible region $\mathcal{F}^C$ of collection C, then all control actions in the control set $\mathbf{U}^C$ are feasible. Each trajectory in the trajectory set $\mathbf{X}_i \in \mathcal{X}^C$ corresponds to a single control action $\mathbf{u}_i^C \in \mathbf{U}^C$.

To construct $\mathcal{T}$, we first sample a range of control actions $\mathbf{U}^C$ from a pre-defined configuration for each collection C. The trajectory set $\mathcal{X}^C$ is then obtained by simulating each control action $\mathbf{u}_i$ for a pre-defined amount of time T using the dynamic model (1). The trajectories are in the egocentric frame of the robot so that they are independent of its pose.

To construct the feasible region $\mathcal{F}^C$, a possible approach is to invert a suitable model for the dynamic feasibility constraint (2) as:

$$\mathcal{F}^C = \bigcap_{\mathbf{u} \in \mathbf{U}^C} \{(\mathbf{x}, \dot{\mathbf{x}}) \mid \mathbf{u} \in \mathcal{D}(\mathbf{x}, \dot{\mathbf{x}})\}. \tag{4}$$

However, the merit of ATL is that such a model need not be explicitly constructed. Rather, control actions may be tested at different states in simulations and hardware trials with diverse configurations and environmental conditions. Similarly, a first-principles model may be adjusted through experimentation.

One important adjustment required is to define a collection with an unbounded feasible region. The purpose is to ensure that for all states in the state space $\mathbb{X}$, there exists at least one feasible collection. This can be achieved by, for example, specifying a range of control actions available when the robot speed is beyond a certain limit.

4.2 Online Planning

Using a trajectory library $\mathcal{T}$ constructed offline, the online planning stage identifies the best collision-free trajectory $\mathbf{X} = \mathbf{x}_1 \ldots \mathbf{x}_T$ over a short time horizon T that is both dynamically feasible and the closest to the next goal $\mathbf{x}^d \in \mathbf{X}^d$. Formally, the planning algorithm solves the following problem at each time t:

$$\begin{aligned} \min_{\mathbf{U}} \quad & J(\mathbf{X}, \mathbf{x}^d) \equiv ||\mathbf{x}_T - \mathbf{x}^d||, \\ \text{s.t.} \quad & \text{dynamics (1)}, \\ & \text{dynamic feasibility (2)}, \\ & \mathbf{X} \cap \mathcal{O} = \emptyset. \end{aligned} \tag{5}$$

Algorithm 1 ATL Planner at time t

Inputs: Current state $\mathbf{x}_t$, desired path $\mathbf{X}^d$, obstacles $\mathcal{O}$
Outputs: Best control action $\mathbf{u}_t^\star$.

```
   ▷ Initialise estimated best cost and control actions
1: $J^\star$, $\mathbf{u}_t^\star \leftarrow \infty, \mathrm{NULL}$
   ▷ Iterate over collections
2: for all collections $(\mathcal{F}^C, \mathcal{X}^C, \mathbf{U}^C) \in \mathcal{T}$ do
     ▷ Skip collection if infeasible for current state
3:   if $\mathbf{x}_t \notin \mathcal{F}^C$ then skip current collection
     ▷ If feasible, iterate over trajectories
4:   for all trajectories in collection $\mathbf{X}_i^C \in \mathcal{X}^C$ do
       ▷ Skip trajectory if in collision
5:     if $\mathbf{X}_i^C \cap \mathcal{O} \neq \emptyset$ then skip current trajectory
       ▷ Compute cost of trajectory. If better, save.
6:     if $J(\mathbf{X}_i^C, \mathbf{x}^d) \leq J^\star$ then
7:       $J^\star$, $\mathbf{u}_t^\star \leftarrow J(\mathbf{X}_i^C, \mathbf{x}^d)$, $\mathbf{u}_i^C$
8: return $\mathbf{u}_t^\star$
```

The process for solving (5) using the library $\mathcal{T}$ is outlined in Algorithm 1, and illustrated in Figure 1. The online planning algorithm takes as input the current state $\mathbf{x}_t$, the desired path $\mathbf{X}^d$, and any obstacles $\mathcal{O}$, and outputs the best control action $\mathbf{u}_t^\star$ for the current time.

The algorithm maintains an estimate of the current best cost $J^\star$ and current best control action $\mathbf{u}_t^\star$, which are initialised as ∞ and a null action respectively (Algorithm 1, line 1). We then loop over all trajectories in all collections (Algorithm 1, lines 2 and 4) while skipping dynamically infeasible collections (Algorithm 1, line 3) and trajectories that result in collision (Algorithm 1, line 5). Computation time is reduced owing to such skipping because the cost function is only evaluated for trajectories that are dynamically feasible and collision-free, and collision-checking is only performed if dynamically feasible. During collision checking of a single trajectory, points along it are sampled in turn from that which is closest to the robot. As soon as a point is found that is coincident with an obstacle, the trajectory is discarded without evaluating any additional points, further reducing computation time. The estimate of best cost and control action is updated if the current cost $J(\mathbf{X}_i^C, \mathbf{x}^d)$ is lower than the estimate $J^\star$ (Alg. 1, line 6). After exhausting the trajectories in all collections, the best control action $\mathbf{u}_t^\star$ is returned.

The algorithm may return a NULL control action if all trajectories fail collision checking. This implies that all dynamically feasible trajectories given the current state will lead to a collision. In this case, a safety behaviour must be triggered. For ground robots, for example, we found stopping at the current position with zero velocity to be a suitable safety behaviour: this is not only because it is safe to do so, but also because the dynamically feasible trajectories when the robot has zero velocity are shorter in length than when moving at speed, and hence more amenable to escaping the obstacles. Meanwhile, the dynamic feasibility check in Algorithm 1, line 3 does not cause a NULL return as long as the feasible sets span the entire state space (i.e. $\bigcup_C \mathcal{F}^C = \mathbb{X}$).

4.3 Considerations for Implementation

Reference Frame

The trajectories in the offline library $\mathcal{T}$ are stored in the egocentric frame of the robot to obviate the need for simulating control actions at runtime. However, the desired path $\mathbf{X}^d$ from the higher-level planner will be typically specified in the global static frame. It is recommended to store the offline trajectories and desired path in this manner as converting $\mathbf{X}^d$ to the egocentric frame of the robot when evaluating the cost function will be less computationally expensive than converting each trajectory into the global static frame.

Obstacle Representation

For practical implementation, we recommend maintaining an independent obstacle representation from that of the higher-level planners. This is because ATL plans over a smaller spatial scale and at much greater frequency compared to typical high-level planners. Two criteria are important for selecting a suitable obstacle representation, which are update time and availability of egocentric operation. In other words, it should be possible to update the obstacle representation quickly, and the obstacle representation should naturally support the egocentric frame as its reference. We found that 2D occupancy grid [Thrun, 2003] implemented in `costmap_2d` [Marder-Eppstein *et al.*, 2023] fits these two criteria well.

5 Experiments

We experimentally demonstrate the effectiveness of our approach in both simulation and field trials. These experiments were conducted using a platform called the *Fast Off-Road Vehicle* (FORV), developed at the University of Technology Sydney (Figure 2). FORV measures 2.4 m × 2 m × 2 m with a maximum design weight of 400 kg, and drives in a skid-steer configuration where each pair of wheels on each side are given the same command velocity. The design enables high-speed traversal of challenging off-road terrains, while containing desktop-level computing hardware to allow complex control and planning algorithms to run at high frequencies. The software is built using ROS Noetic to create a modular architecture that can be extended for different applications.

5.1 ATL Implementation

Our ATL planner is implemented using the standard ROS Navigation stack. The trajectory library is described by the user in a `YAML` file loaded at runtime. The library consists of collections with forward velocity $v \in \{1, 1.5, 2, 3, 4\}$ m/s and angular velocity ω °/s, where:

$$\omega \in \begin{cases} [-10, 10] \text{ step } 2, & v = 1 \\ [-12, 12] \text{ step } 3, & v = 1.5 \\ [-20, 20] \text{ step } 4, & v = 2 \\ [-28, 28] \text{ step } 4, & v = 3 \\ [-36, 36] \text{ step } 9, & v = 4 \\ [-50, 50] \text{ step } 10, & v = 5 \end{cases}$$

These values were chosen through empirical observations of FORV driving around a testing environment. The allowable ω is increased at greater v as we have observed that the skid steer dynamics of FORV allow it to perform tighter turns when travelling at higher speeds. Each trajectory is simulated for 5 s with points sampled at intervals of 0.2 s along it to determine whether it results in collision and to evaluate the cost function. Trajectories are deemed to be dynamically feasible at time t if $(v_t - 3) \le v \le (v_t + 3)$ and $(\omega_t - 115) \le \omega \le (\omega_t + 115)$, where v_t and ω_t are the linear and angular velocity of the robot at time t respectively.

5.2 Simulation Experiment

Simulation Setup

The ATL planner was first evaluated in simulation to determine its suitability for high speed operation in outdoor environments. Simulations were run in Gazebo, a popular simulation package in the robotics community that enables accurate modelling of complex real-world scenarios. The simulated FORV was loaded into a terrain based off that provided in the NEGS-UGV Dataset [Sánchez *et al.*, 2022]. The environment measures 100 m × 50 m, and consists of a flat surface with obstacles such as trees, benches, and street lamps distributed throughout (Figure 3). The spacing between obstacles was chosen to provide both open spaces and areas where the robot would be required to negotiate tight gaps. In order to examine the performance of the planners independent of other factors, the robot obtains its location directly from Gazebo instead of relying on internal odometry,

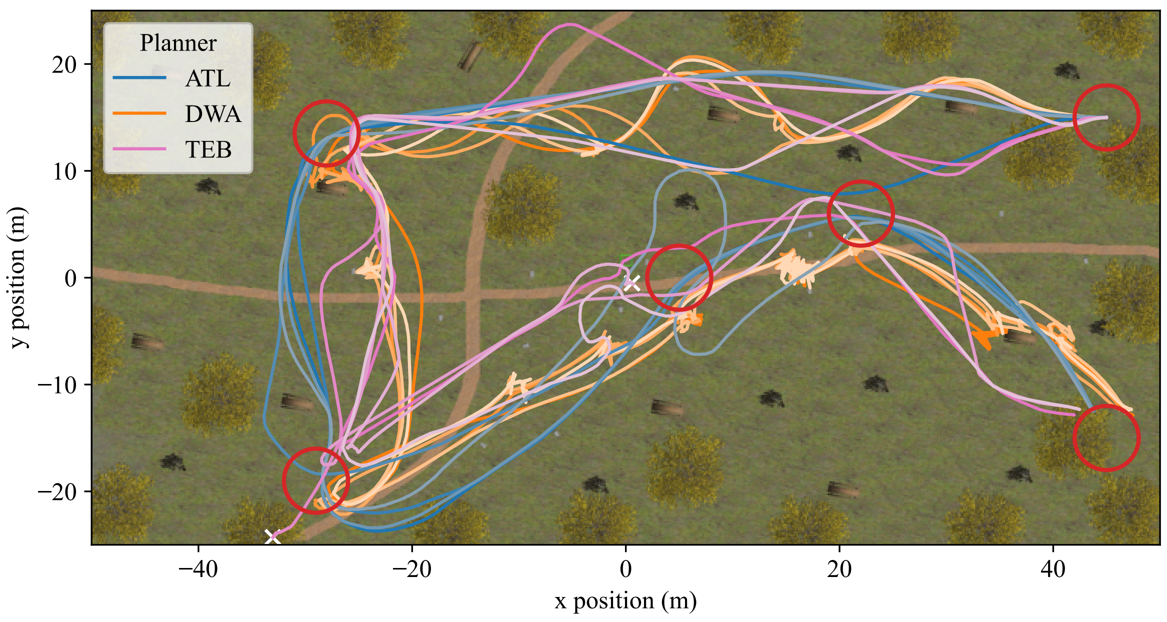


Figure 4: Simulation results for the ATL, DWA, and TEB planners. In each trial, the robot starts in the top right, and moves through the waypoints shown in red, where the circle indicates the acceptance radius. Trajectories followed by FORV under each planner are shown in blue and orange, with different trials indicated by different saturations. White crosses indicate locations where trials ended early due to failure.

and a static costmap was generated from the terrain offline using the `costmap_2d` package [Marder-Eppstein *et al.*, 2023].

FORV was initialised near the top right corner of the simulated environment, and provided with several ordered waypoints to visit; waypoints were marked as visited when the robot reached a location within 3 m of them. In addition to our ATL planner, we implemented DWA and TEB planners to evaluate our method against using the `dwa_local_planner` and `teb_local_planner` packages respectively [Marder-Eppstein, 2023; Rösmann, 2023]. Five simulations were performed for each planner, and the paths followed by FORV were recorded for later analysis.

Results

The paths followed by FORV in each trial are plotted in Figure 4, and summarised in Table 1. It can be seen that our ATL planner performs favourable compared to the DWA and TEB planners. Our method produces paths that are significantly shorter than DWA and similar in length to TEB, and these paths are followed quicker than those produced by either existing planner. ATL and TEB produce smooth paths with wide corners that can be traversed at high speed, while DWA often selects tight or on-the-spot turns, which reduces the linear speed the robot is able to move at. We have also observed that the real FORV is unable to perform such tight cornering manoeuvres, reducing the applicability of this planner on our chosen platform.

The success rate achieved by each planner is another important consideration. Trials can fail when the planner is unable to generate control commands, the robot collides with an obstacle, or the robot becomes immobilised due to its footprint overlapping with an obstacle zone within the costmap. All five trials of the ATL and DWA planners succeeded, but two trials of the TEB planner failed. In proximity to densely populated obstacle areas, TEB generated inefficient commands, often outputting large angular velocities. This tendency caused the robot to oscillate around obstacles in an attempt to find a viable path through the area, eventually driving it into occupied regions of the costmap from which it could not recover.

The planners perform relatively consistently during successful trials, moving through similar trajectories. Occasionally, obstacles would be passed on the opposite side to what was usually selected by that planner, but the paths would soon converge again. This can lead to a ripple effect, where passing one obstacle on a different side results in a longer path overall as further obstacles must be avoided while obeying dynamic feasibility constraints before returning to the usual path.

Planner	Travel time (s)	Path length (m)	Planning time (ms)
ATL	90	205	26
DWA	569	339	198
TEB	170	197	19

Table 1: Simulation results for the ATL, DWA, and TEB planners. Values given are the mean values across multiple trials.

In addition to the metrics relating to the paths produced, the computational cost of each planner was also compared. We measured the time taken to calculate the velocity commands at each planning instant, and found the average planning time of our ATL planner was significantly lower than DWA but slightly higher than TEB. This shows the low computational cost of our method in producing successful paths, which lends itself to the fast planning required for robots travelling at high speeds.

5.3 Field Experiments

Hardware Setup

Our ATL planner was also tested on the physical FORV platform to evaluate its real-world performance. Broadly the same software was run on the real robot as in simulation, however it was not possible to isolate the performance of the planner from the localisation and costmap generation. In real-world experiments, the robot calculates its position by fusing data from motor encoders, an inertial measurement unit, and GPS using the `robot_localization` package for ROS [Moore and Stouch, 2014]. Similarly a static costmap cannot be used, and instead a live costmap is generated from lidar data.

We conducted a field trial at Sydney Science Park during which two experiments were performed. In each experiment, FORV began near a selection of obstacles including trees and a dummy human. FORV was instructed to move through two waypoints with a tolerance of 3 m then stop. Two experiments were performed, varying the starting position and orientation of FORV with respect to the obstacles as well as the position of the waypoints to determine the performance of the planner in different scenarios.

Results

Results from the field trial are shown in Figure 5. In the first experiment (Figure 5a), FORV was positioned a short distance from a tree and oriented directly towards it. A waypoint was placed directly behind this tree, such that FORV would have to immediately avoid an obstacle. Another waypoint was placed a short distance away to direct FORV to turn and avoid two further obstacles. The ATL planner performed well in this scenario, avoiding all obstacles and moving through a smooth path without sharp turns that would reduce its linear velocity. The robot travelled 96 m in 33 s, achieving an average linear speed of 2.9 m/s.

The second experiment considered a more challenging scenario (Figure 5b). FORV was initially aligned pointing directly down a row of obstacles, and waypoints were selected to direct the robot to slalom between them. This was achieved without any collisions, demonstrating the impressive obstacle avoidance capabilities of the planner even when operating at relatively high speeds. In this trial, the robot travelled 136 m in 28 s, so moved with a higher average linear speed than the previous trial of 4.9 m/s.

6 Conclusion

This paper has presented ATL, a novel local path planning algorithm suitable for robots travelling at high speeds. The approach reduces online computation by using a precomputed trajectory library, an adaptive subset of which is sampled from to reflect trajectories that are dynamically feasible given the current state of the robot. These sampled trajectories are simulated to find the most suitable collision-free trajectory at each instant.

The planner was first evaluated in simulation, and found to perform favourably compared to the existing DWA planner. We also tested the planner in field trails, where it showed good performance in two scenarios, reaching goal locations quickly and without collision.

In future, we will investigate how the trajectory library can be further adapted to encapsulate more real-world constraints. In particular, the dynamics of the robot are dependent on the terrain, so the library could be expanded to allow for trajectories to vary as the robot moves across different surfaces. We also intend to incorporate ATL with a global planner that generates suitable sparse waypoints to demonstrate autonomous navigation across a larger area.

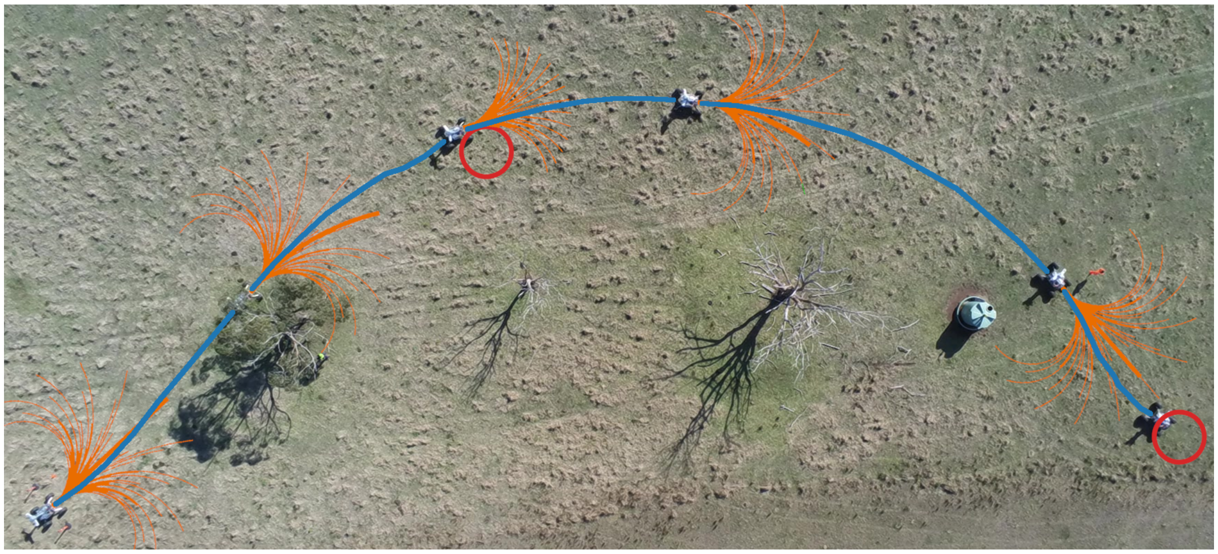
Acknowledgements

We would like to thank the UTS MMR technical team for hardware and field trial support. This work was supported in part by the Trusted Autonomous Systems D-CRC.

References

[Best *et al.*, 2023] Graeme Best, Rohit Garg, John Keller, Geoffrey A. Hollinger, and Sebastian Scherer. Multi-robot, multi-sensor exploration of multifarious environments with full mission aerial autonomy. *International Journal of Robotics Research*, 2023.

[Campbell, 2007] Stefan Forrest Campbell. *Steering control of an autonomous ground vehicle with appli-*


(a)

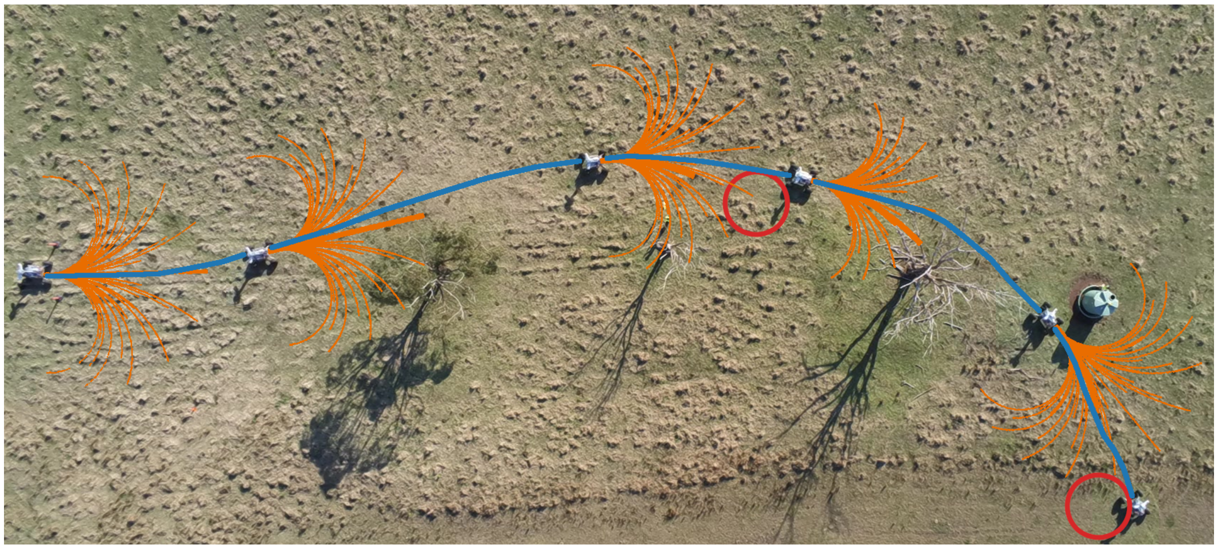

(b)

Figure 5: Real-world experimental results. Waypoints are shown in red, where the circle indicates the acceptance radius. The trajectories followed by FORV are plotted in blue. FORV is shown at several points equally spaced in time (5.5 s in (a) and 6.6 s in (b)). At each of these points, the current feasible trajectories (orange) and the selected path (thick orange) are also visualised.

cation to the DARPA urban challenge. PhD thesis, Massachusetts Institute of Technology, 2007.

[Chu *et al.*, 2012] Keonyup Chu, Minchae Lee, and Myoungho Sunwoo. Local path planning for off-road autonomous driving with avoidance of static obstacles. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1599–1616, 2012.

[Coulter, 1992] R Craig Coulter. *Implementation of the pure pursuit path tracking algorithm.* Carnegie Mellon University, The Robotics Institute, 1992.

[Dijkstra, 1959] E W Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[Fox *et al.*, 1997] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[Goswami, 2017] Angshuman Goswami. *Hierarchical Off-Road Path Planning and Its Validation Using a*

Scaled Autonomous Car. PhD thesis, Clemson University, 2017.

[Hart *et al.*, 1968] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Janson *et al.*, 2015] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *International Journal of Robotics Research*, 34(7):883–921, 2015.

[Karur *et al.*, 2021] Karthik Karur, Nitin Sharma, Chinmay Dharmatti, and Joshua E Siegel. A survey of path planning algorithms for mobile robots. *Vehicles*, 3(3):448–468, 2021.

[Kavraki *et al.*, 1996] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

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

[LaValle and Kuffner Jr, 2001] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[Lee *et al.*, 2014] James Ju Heon Lee, Kris Frey, Robert Fitch, and Salah Sukkarieh. Fast path planning for precision weeding. In *Proc. of Australasian Conference on Robotics and Automation (ACRA)*, 2014.

[Likhachev and Ferguson, 2009] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *International Journal of Robotics Research*, 28(8):933–945, 2009.

[Macenski *et al.*, 2023] Steve Macenski, Shrijit Singh, Francisco Martín, and Jonatan Ginés. Regulated pure pursuit for robot path tracking. *Autonomous Robots*, pages 685–694, 2023.

[Marder-Eppstein *et al.*, 2023] Eitan Marder-Eppstein, David V. Lu, and Dave Hershberger. costmap_2d, 2023. `http://wiki.ros.org/costmap_2d`.

[Marder-Eppstein, 2023] Eitan Marder-Eppstein. dwa_local_planner, 2023. `http://wiki.ros.org/dwa_local_planner`.

[Moore and Stouch, 2014] T. Moore and D. Stouch. A generalized extended Kalman filter implementation for the robot operating system. In *Proc. of International Conference on Intelligent Autonomous Systems (IAS)*. Springer, July 2014.

[Rösmann *et al.*, 2012] Christoph Rösmann, Wendelin Feiten, Thomas Wösch, Frank Hoffmann, and Torsten Bertram. Trajectory modification considering dynamic constraints of autonomous robots. In *Proc. of German Conference on Robotics*, 2012.

[Rösmann *et al.*, 2017] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Integrated online trajectory planning and optimization in distinctive topologies. *Robotics and Autonomous Systems*, 88:142–153, 2017.

[Rösmann, 2023] Christoph Rösmann. teb_local_planner, 2023. `https://wiki.ros.org/teb_local_planner`.

[Sánchez *et al.*, 2022] Manuel Sánchez, Jesús Morales, Jorge L Martínez, JJ Fernández-Lozano, and Alfonso García-Cerezo. Automatically annotated dataset of a ground mobile robot in natural environments via gazebo simulations. *Sensors*, 22(15):5599, 2022.

[Stentz, 1994] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3310–3317, 1994.

[Thrun, 2003] Sebastian Thrun. Learning occupancy grid maps with forward sensor models. *Autonomous Robots*, 15:111–127, 2003.

[You *et al.*, 2020] Alexander You, Fouad Sukkar, Robert Fitch, Manoj Karkee, and Joseph R Davidson. An efficient planning and control framework for pruning fruit trees. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3930–3936, 2020.