RAPID: An Algorithm for Quick Replanning under Changed Dynamical Constraints

Sharan Nayak and Michael W. Otte

Department of Aerospace Engineering,
University of Maryland, College Park, Maryland 20742, USA
snayak18@umd.edu, otte@umd.edu

Abstract. There are many scenarios where the dynamical constraints of a vehicle may change mid-mission either due to added payload, vehicle damage or changes to the environment. The vehicle may need to come back to the base safely without undergoing collisions with obstacles. We present RAPID (RApid Planning for Interchanged Dynamics), a sampling based motion planning algorithm for quick replanning when the dynamical constraints associated with a vehicle change. RAPID uses the planning information associated with the original dynamics as a heuristic for quick replanning. RAPID is useful when a vehicle must replan to account for changes to its dynamics but does not have enough time or resources to run a whole planning algorithm (like RRT or SST) from scratch. We compare RAPID with brute-force replan using RRT and SST using a combination of hardware and software experiments. The hardware experiments are performed using a modified Parrot Bebop 2 quadrotor fitted with an Intel UP board traversing an obstacle course. Our experimental results demonstrate RAPID provides considerable improvements over planning from scratch (RRT, SST) in solution completion time, path length, and flight time performance metrics.

Keywords: Planning and control, Mobile robots, Unmanned aerial vehicles.

1 Introduction

We present **RAPID** (**RA**pid **P**lanning for **I**nterchanged **D**ynamics), a sampling-based motion planning algorithm for performing quick replan back to base when the dynamical constraints of a vehicle change. There are many situations in which the maneuvers available to a vehicle may change mid-mission, e.g., due to added payload [1] or damage to the vehicle [2]. After the change, the inbuilt controller may be unable to execute maneuvers along the original plan. In such a case it is imperative that the vehicle quickly replan to account for its new dynamical constraints so that it may return safety to base (for repair, payload delivery, etc.) without colliding with obstacles.

A naive replanning approach is to re-run the whole planning algorithm (like RRT [3] or SST [4]) with the new dynamical constraints and find a new path. However, in high stakes missions, the vehicle may not have enough time or

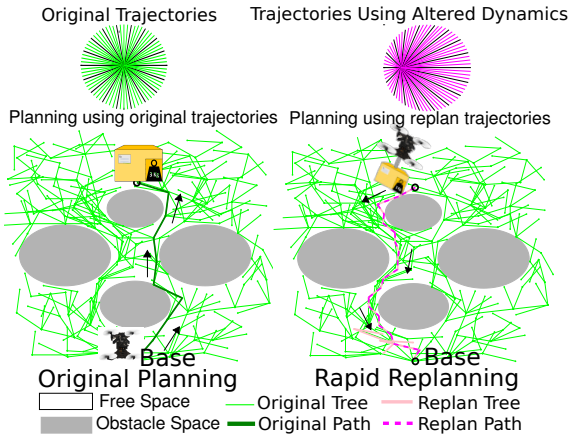

Fig. 1: Left image shows a quadrotor using its original trajectories to plan and move towards a sealed package of uneven weight. The right image shows the same quadrotor lifting the package and experiencing a dynamics change due to change in mass and shifted center of gravity and using RAPID algorithm to perform quick replanning and move back to the base location. Trajectories at 20° increments are shaded black for clarity.

resources to perform replanning from scratch. Instead, our method is able to quickly repair the old plan to respect the vehicle's new dynamics by leveraging the topological information gained about the environment during the initial plan. RAPID uses the original tree's cost information as a heuristic for quick replanning. Fig. 1 depicts a case where a quadrotor undergoes a dynamics change due to lifting a package that causes uneven weight distribution; using RAPID, the vehicle quickly replans a valid return path to the base location.

A benefit of using RAPID is that it can be used for vehicles for which the solution to the two-point Boundary Value Problem (BVP) [5] is computationally expensive or not available. This is important because even though the two-point BVP may be solvable for the vehicle's original dynamics, there is no guarantee that the vehicle's dynamics change in a way such that two-point BVP is still solvable. We compare RAPID with brute force replan using RRT [3] and SST[4] as these are two standard algorithms that are used when two-point BVP cannot be computed.

In our work, we focus on the motion replanning aspects and assume that vehicle automatically detects changes to its dynamics. We model the vehicle's movement using trajectory libraries [6][7] and assume that these libraries exists for vehicle's common configurations and malfunctions.

2 Related Work

A survey of the sampling-based motion planning literature reveals limited work on replanning under changed dynamical constraints but numerous existing work on replanning under changed dynamic environments. For example, Bekris et al. [8] present a tree-based planner that deals with evolving configuration spaces by incrementally updating a tree-based data structure retained from previous steps and then biasing the search with a greedy and probabilistic complete exploration approach. Ferguson et al. [9] make repairs to an already existing RRT

tree, removing invalid parts and then growing the changed tree until a new solution is found to account for the changed configuration space. Yoshida et al. [10] present a reactive method that uses a combination of path replanning and path deformation to account for changed environments. Otte et al. [11] present RRT-X, the first asymptotically optimal sampling based algorithm in dynamic environments that uses ideas from the D* algorithm for quick replanning. While such algorithms work for replanning in evolving configuration spaces, they are not directly applicable to replanning in response to changing vehicle dynamics. Nonetheless, our method is inspired by the same high-level idea of using an existing but outdated solution to speed up the replanning process.

Several existing works show the power of using heuristics to improve the speed and efficiency of searches. Urmson et al.[12] modify RRT to create the heuristically-guided RRT using a quality measure to balance exploration vs exploitation search leading to low cost solutions. Otte et al. [13] and Gammell et al. [14] use the ellipsoidal heuristic to perform rejection sampling and direct sampling respectively to speed up the process of generating improved solutions. Straub et al. [15] use the cost-to-go heuristic obtained from an asymmmetric bidirectional search with the reverse search tree produced using Lifelong Planning A* (LFA*) to speed up the planning process. Littlefield et al. [16] modify SST to produce the Informed-SST algorithm which uses domain-specific user defined heuristics to improve the efficiency of SST. In keeping with this trend, our replanning algorithm uses the cost-to-start heuristic obtained from the original search tree to improve the efficiency of the replan search.

Our work uses trajectory libraries to model the motion of vehicles incorporating dynamical constraints. The idea of using trajectory libraries to represent vehicle motion is not something new. Go et al. [6] presents a method using trajectory libraries and real time motion planning to synthesize animations of autonomous space, water, and land-based vehicles in interactive simulations. They also provide a method for efficient storage of trajectories in memory using only about 40 bytes per path. Frazzoli et al. [7] use trim trajectories and maneuvers to perform autonomous helicopter flight. The advantage of using trajectory libraries as noted in [17] is that no online time is wasted in calculating trajectories that a robot cannot carry out.

3 Problem Definition

In this section, we formally define the problem. Let $\mathcal{X}$ be the D-dimensional configuration space of a robot. Let $\mathcal{X}_{obs}$ be the obstacle space of $\mathcal{X}$. Then the free space $\mathcal{X}_{free}$ is defined as $\mathcal{X}_{obs} \setminus \mathcal{X}$. Let the home base configuration of the robot be denoted by $x_{base} \in \mathcal{X}_{base}$ where $\mathcal{X}_{base} \subset \mathcal{X}_{free}$. Let $\mathcal{X}_{end}$ be the end region with $\mathcal{X}_{end} \subset \mathcal{X}_{free}$. Let β be a feasible path with $\beta : [0,1] \rightarrow \mathcal{X}_{free}$ such that $\beta(0) = x_{base}$ and $\beta(1) = x_{end}$ where $x_{end} \subset \mathcal{X}_{end}$. Let $\alpha \in (0,1)$ and $\beta(\alpha) = x_{replan}$ be the configuration where the dynamics of the robot changes.
Problem: Find feasible path back to base incorporating dynamical changes.

Given the triple $(\mathcal{X}_{free}, x_{replan}, \mathcal{X}_{base})$*, find a feasible path* $\gamma : [0,1] \rightarrow \mathcal{X}_{free}$ *such that* $\gamma(0) = x_{replan}$ *and* $\gamma(1) \subset \mathcal{X}_{base}$ *if one exists.*

4 Technical Approach

The main idea behind RAPID is the use of topological information gained from original search tree for quick replanning. Although the edges in the original tree correspond to old dynamical maneuvers, the nodes are still valid obstacle free configurations which can be used to bias the replan towards obstacle free regions of the search space. At every iteration, RAPID greedily selects the most promising node from the original search tree to replan towards. This guides RAPID's replan away from obstacles and toward the home base.

Algorithm 1: RAPID$(v_{start}, v_{end}, \mathcal{G}_{orig}, \mathcal{T}_{replan}, \delta_v, \delta_{end})$

1 $\mathcal{G}_{replan} \leftarrow \{\mathbf{V} \leftarrow \{v_{start}\}, \mathbf{E} \leftarrow \{\}\}$
2 $\mathcal{Q} \leftarrow \{\}, v_{cur} \leftarrow v_{start}, \mathbf{E}_{proc} \leftarrow \{\}$
3 **for** $i \leftarrow 1$ *to* M_{iter} **do**
4 **if** $v_{cur} \neq$ NULL **then**
5 $\mathbf{V}_{near} \leftarrow$ getNearestNodes$(v_{cur}, \mathcal{G}_{orig}, \delta_v)$
6 updatePriorityQ$(\mathcal{Q}, v_{cur}, \mathbf{V}_{near})$
7 $v_{dst} \leftarrow$ pop$(\mathcal{Q})$
8 **if** $v_{dst} \neq$ NULL **then**
9 $v_{cur} \leftarrow$ NULL
10 $v_{src} \leftarrow$ findClosestNode$(v_{dst}, \mathcal{G}_{replan}, \mathbf{E}_{proc})$
11 **if** $v_{src} \neq$ NULL **then**
12 $\mathcal{E}_{new} \leftarrow$ Propagate$(v_{src}, v_{dst}, \mathcal{T}_{replan}, \mathbf{V})$
13 **if** $\mathcal{E}_{cur} \neq$ NULL **then**
14 $v_{new} \leftarrow \mathcal{E}_{new}$.getFinalNode()
15 $\mathbf{V} \leftarrow \mathbf{V} \cup \{v_{new}\}$
16 $\mathbf{E} \leftarrow \mathbf{E} \cup \{\mathcal{E}_{new}\}$
17 **if** endReached$(v_{end}, v_{new}, \delta_{end})$ **then**
18 **return** OutputPath(v_{start}, v_{new})
19 $v_{cur} \leftarrow v_{new}$
20 $\mathbf{E}_{proc} \leftarrow \mathbf{E}_{proc} \cup (v_{src}, v_{dst})$
21 **else**
22 **break**
23 **return** PerformBaseAlgorithm$(v_{start}, v_{cur}, \mathcal{T}_{replan}, \delta_{end})$

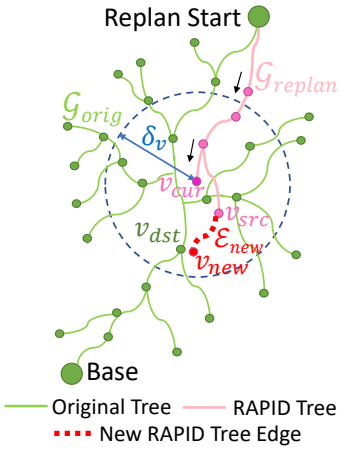

Fig. 2: New edge addition to RAPID tree

The input parameters to RAPID (Algorithm 1) are start node v_{start} (robot's current location), end node v_{end} (home base), original search tree $\mathcal{G}_{orig}$ (generated using RRT or SST), replan trajectory library $\mathcal{T}_{replan}$, neighbor radius δ_v and end radius δ_{end}. The replan search tree $\mathcal{G}_{replan}$ is initialized using a node list $\mathbf{V}$ (initialized to v_{start}) and an empty edge list $\mathbf{E}$ (line 1). We use a priority queue $\mathcal{Q}$ to store a list of potential original search trees to guide replan expansion. The current node v_{cur}, which is used for selecting nodes of $\mathcal{G}_{orig}$ to add to $\mathcal{Q}$ is initialized to v_{start} (line 2). We maintain a list $\mathbf{E}_{proc}$ to keep track of processed edges to prevent cycles in $\mathcal{G}_{replan}$ (line 1).

The execution of one iteration of RAPID occurs as follows: While $v_{cur} \neq$ NULL, (line 4), the algorithm uses getNearestNodes (Algorithm 2) to find all nodes within a radius δ_v from $\mathcal{G}_{orig}$ centered at v_{cur} and assigns it to set $\mathbf{V}_{near}$

Algorithm 2: `getNearestNodes`$(v_{cur}, \mathcal{G}_{orig}, \delta_v)$

1 $\mathbf{V}_{near} \leftarrow \mathcal{G}_{orig}$.`Nodes`()
2 $\mathbf{V}_{near} \leftarrow \{ v \in \mathbf{V}_{near} \mid \|v - v_{cur}\| \leq \delta_v \}$
3 **return** $\mathbf{V}_{near}$

Algorithm 3: `updatePriorityQ`$(\mathcal{Q}, v_{cur}, \mathbf{V}_{near})$

1 **for** v `in` $\mathbf{V}_{near}$ **do**
2 $\quad \mathcal{Q}$.`update`$(v, g(v_{cur}) + d(v_{cur}, v) + \epsilon h(v))$

Algorithm 4: `findClosestNode`$(v_{dst}, \mathcal{G}_{replan}, \mathbf{E}_{proc})$

1 $\mathcal{V}_{replan} \leftarrow \mathcal{G}_{replan}$.`Nodes`()
2 $v_{closest} \leftarrow \min_{\|v - v_{dst}\|} \{ v \in \mathcal{V}_{replan} \}$
3 **if** $(v_{closest}, v_{dst}) \in \mathbf{E}_{proc}$ **then**
4 $\quad v_{closest} \leftarrow$ `NULL`
5 **return** $v_{closest}$

Algorithm 5: `Propogate`$(v_{src}, v_{dst}, \mathcal{T}_{replan}, \mathbf{V})$

1 $\mathcal{T}_{replan\_tf} \leftarrow$ `Transform`$(v_{src}, \mathcal{T}_{replan})$
2 $\mathcal{T}_{sort} \leftarrow \underset{\|\mathcal{E}.\texttt{finalNode}() - v_{dst}\|}{sort} \{ \mathcal{E} \in \mathcal{T}_{replan\_tf} \}$
3 $\mathcal{T}_{sort\_best} \leftarrow \underset{N}{select\_best} \{ \mathcal{E} \in \mathcal{T}_{sort} \}$
4 **for** $\mathcal{E}$ `in` $\mathcal{T}_{sort\_best}$ **do**
5 $\quad v_{fin} \leftarrow \mathcal{E}_{new}$.`finalNode`()
6 $\quad$ **if not** $(v_{fin} \in \mathbf{V}$ *or* `Collision`$(\mathcal{E}))$ **then**
7 $\quad\quad$ **return** $\mathcal{E}$
8 **return** `NULL`

(line 5). The algorithm uses `updatePriorityQ` (Algorithm 3) to update $\mathcal{Q}$ with the nearest nodes $\mathbf{V}_{near}$ (line 6). The algorithm then uses v_{dst} popped from $\mathcal{Q}$ (line 7) to find the closest node v_{src} in $\mathcal{G}_{replan}$ (line 10) using `findClosestNode` (Algorithm 4). We perform forward search using `Propogate` (Algorithm 5) to find the best edge $\mathcal{E}_{new}$ from v_{src} to v_{dst} (line 12) (Fig. 2). If $\mathcal{E}_{new} \neq$ `NULL`, then the new edge $\mathcal{E}_{new}$ and new node v_{new} (final node of $\mathcal{E}_{new}$) are updated in $\mathbf{V}$ (line 15) and $\mathbf{E}$ (line 16) respectively. If the end is reached (checked using `endReached`), the output path $\mathcal{P}_{out}$ is generated using `OutputPath` (line 18). If RAPID is unable to quick replan, then the brute-force replan using the algorithm from the original plan (line 23) (or any other sampling-based planning algorithm) is run to get the output path as a fallback option.

Note that the quick replanning part of RAPID is deterministic and the fallback option (assuming a sampling-based algorithm is used) is probabilistic. Hence the condition for RAPID algorithm to be probabilistic complete is that the base algorithm should be probablisitic complete.

The function `getNearestNodes` (Algorithm 2) performs the task of getting the nearest nodes to v_{cur} in $\mathcal{G}_{orig}$ within a ball of radius δ_v (line 2). The metric defined on the configuration space is used to calculate the distance $\|v - v_{cur}\|$ where $v_{cur} \in \mathbf{V}_{near}$ (set of all nodes of $\mathcal{G}_{orig}$). All nodes $\mathbf{V}_{near}$ that lie within distance δ_v are returned.

The function `updatePriorityQ` (Algorithm 3) updates the priority queue $\mathcal{Q}$ with the nodes $v \in \mathbf{V}_{near}$ using the key [18] $c_{total} = g(v_{cur}) + d(v_{cur}, v) + \zeta h(v)$ where $g(v_{cur})$ is the cost-from-start, $d(v_{cur}, v)$ is the cost from v_{cur} to $v \in \mathbf{V}_{near}$, $h(v)$ is the cost-to-end heuristic, and ζ is the weight factor [19]. We choose $g(v_{cur})$ and $h(v)$ as the corresponding path costs, $d(v_{cur}, v)$ as the euclidean distance between v_{cur} and v and set ζ to 1.

The function `findClosestNode` (Algorithm 4) finds the closest node $v_{closest}$ to v_{dst} in $\mathcal{G}_{replan}$ (line 2). The node pair $(v_{closest}, v_{dst})$ is checked to make sure that is not already in the edge processed list to ensure the same pair of edge nodes is not processed again (line 3). If this condition is satisfied, then $v_{closest}$ is set to `NULL` (line 4).

The function `Propogate` (Algorithm 5) returns the best path that connects v_{src} to v_{dst}. The best path is generated by first transforming (translation and rotation) the trajectory library $\mathcal{T}_{replan}$ (line 1) using v_{src} and then sorting the trajectories $\mathcal{E}$ in $\mathcal{T}_{replan\_tf}$ using the distance metric $\|\mathcal{E}.\texttt{finalNode}() - v_{dst}\|$ (line 2). The N best sorted trajectories are selected from $\mathcal{T}_{sort}$ and assigned to $\mathcal{T}_{sort\_best}$ (line 3). For each of the sorted trajectories $\mathcal{E}$, we check if v_{fin} (last node of $\mathcal{E}$) is already in the node list $\mathbb{V}$ and if the edge $\mathcal{E}$ is collision with the obstacles (line 6). If both of these conditions are not satisfied, then $\mathcal{E}$ is returned.

4.1 Selection of neighbor radius δ_v

The RAPID algorithm has a tuning parameter δ_v that affects its performance. This parameter dictates the number of original tree nodes that gets pushed to $\mathcal{Q}$. A low value chosen increases the prospect of many important $\mathcal{G}_{orig}$ nodes being not considered. A high value selected increases the chances of adding $\mathcal{G}_{orig}$ nodes that are blocked by obstacles which might lead to considering more trajectories for expansion that collide with obstacles. In our experiments, we choose a value that is equal to or slightly greater than twice the length of the largest trajectory.

5 Experiments

We run multiple experiments in hardware and software to test the performance of RAPID. We compare RAPID with brute-force replan using RRT and SST, each having two different variants. The first variant is using the extend-operation [20] for node expansion while the second variant is using random propagation [4]. The extend operation (in our implementation) extends the nearest node in the existing tree by a distance ϵ to a node v_{extend} in the direction of the new sampled configuration. We then select the trajectory in corresponding $\mathcal{T}$ ($\mathcal{T}_{replan}$ for replan and $\mathcal{T}_{orig}$ for original) that minimizes $\|v_{extend} - v_i\|$ where v_i is final node of $\mathcal{E}_i \in \mathcal{T}$. The different variants are used to test the durability of RAPID in using different original tree structures for its planning. All variants have a goal bias [3] of 5% to speed up solution generation.

In our experiments, we choose the algorithm used to run the original plan as the base algorithm if RAPID is unable to quickly replan. We choose solution path length L_p, algorithm completion time T_c and flight time T_f (only hardware) as the performance metrics. We generate the trajectories in the trajectory libraries such that their start and end velocities are zero. This is done to limit the number of trajectories in the trajectory libraries and is not required to use RAPID.

The hardware experiments are performed using a modified Bebop 2 (PRG Husky [21]) quadrotor on an obstacle course (Fig. 3) using parameters shown in Table 1. The quadrotor is fitted with Intel UP board (Fig. 6) having Intel Atom x5-z8350 4-core, 1.44GHz CPU with 4GB RAM to run the algorithms and navigation stack. The obstacle course has 4 obstacles (Fig. 4) and they are modeled using bounding circles for easy collision detection. The bounding circle radii are increased by 20cm to account for non-zero vehicle size, vehicle drift and errors.

Parameter	Value (m) (Hardware)	Value (m) (Software)
Map size	5×4	100×100
Base/Replan start area radius (δ_{end})	0.3	5
Neighbor radius (δ_v)	2.2	10
Max iterations (M_{iter})	2500	2500
Propagate Trajectory Count (N)	5	5

Table 1: Experiment parameters

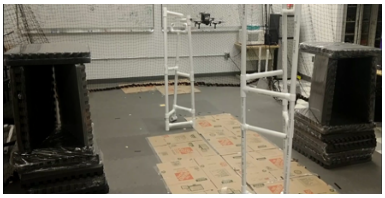

Fig. 3: Quadrotor in obstacle course

The base and replan start locations are set at $(0.2, 2)$ and $(4.2, 2)$ respectively. The quadrotor control is limited to performing maneuvers in the horizontal plane at a fixed height. We use a Proportional-Derivative (PD) controller for position tracking and a Vicon motion capture system for state information. The dynamics change for the quadrotor is represented by changing the gains of PD controller from nominal to one-third of the nominal gains. The trajectories (Fig. 4), both original and altered, are learned by providing a reference 1m away from origin in 10° increments over 4s and then taking the mean of the trajectories over 10 iterations. We use these trajectories to run 10 trials and record the performance metrics. We set ϵ to 0.8 and 0.65 when using original and replanning libraries respectively.

The software experiments are performed in 2D space using parameters shown in Table 1. The obstacles are modeled as circles with uniformly sampled radii in [5, 8]. The base, replan start and obstacle center locations for each scenario are uniform randomly sampled from the parameter ranges ([1, 100], [1, 100]) such that they do not overlap each other and the obstacle surfaces are within the map area. The minimum distance between base and replan start locations is set to 70. We run 100 scenarios for each obstacle number $\{0, 5, 15, 25, 35\}$ with each scenario having 3 trials. We run experiments for a unicycle model [22] with the dynamics change represented by changed reference peak linear velocity (8m/s to 4m/s) and a quadrotor model [1] with a dynamics change represented by changed mass (1 kg to 1.1kg) and shift in the center of gravity (4cm and 1cm parallel to the x- and y-axes). We set ϵ to 4 (for both original and replan) for quadrotor and 4 (original) and 2.5 (replan) for unicycle. The experiments are performed on a system with Intel i7-7700 4-core CPU with 32GB RAM. The stopping criterion for both algorithms in all experiments is when a path is found between the replan start and base locations.

6 Results

The replan paths generated during comparison of RAPID vs. RRT for two trials of hardware experiments with actual path (Vicon) overlaid are shown in Fig. 4. The mean and standard deviation for 10 trials of hardware experiments are shown in Table 2. The replan paths produced during comparison of RAPID vs. RRT-Extend and RAPID vs. SST-Random for one of our software test scenarios are shown in Fig. 7. We have not shown examples of replan paths produced during comparison RAPID vs. RRT-Random and RAPID vs. RRT-Extend due

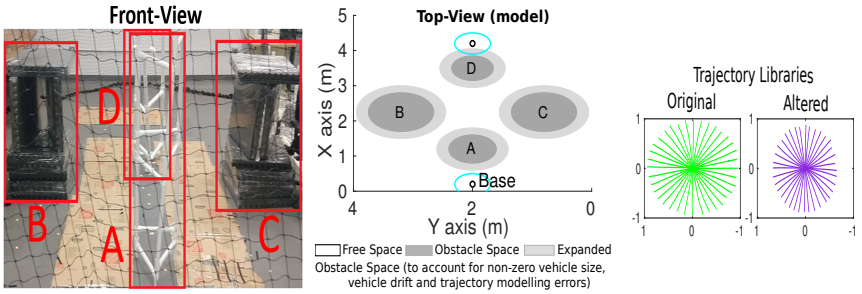

Fig. 4: For hardware experiments, left figure shows the front view of the obstacle course and middle figure shows top view (model). The original and altered (replan) trajectory libraries generated are shown to the right. The replan trajectory library is colored purple to indicate that same library is used for both RRT (blue) and RAPID (magenta).

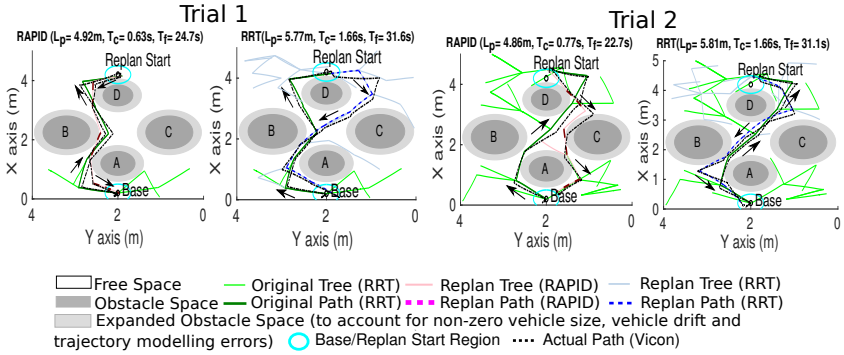

Fig. 5: Figure shows comparison of RAPID and RRT for two trials of hardware experiments. RAPID is only compared with RRT(using extend) as this is the best performing algorithm found in the software experiments.

to brevity and the limit on the number of pages. The results of the software experiments are shown in Fig. 8.

Our hardware and software experiments demonstrate that RAPID performs better than brute-force replan using RRT and SST with respect to all performance metrics that we evaluate (Fig. 8)(Table 2). The completion time for the RAPID algorithm is much faster than replanning from scratch. We believe this speedup can be attributed to RAPID's use of the original plan's topological information within its guiding heuristic. This is notable given that the heuristic is based on the system's original dynamics. The guiding heuristic is also responsible for causing path length to be shorter for RAPID. This is because using the heuristic guides the planning away from obstacles and towards to the goal, in comparison to planning from scratch where the path is much more random. The completion time and path length shows upward trends (Fig. 8) with increase in the number of obstacles. We believe the increase in completion time is because of increased collision checking that is necessary with more obstacles. Path length

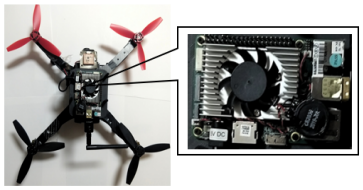

Fig. 6: Quadrotor (with Intel UP board enlarged) used in our experiments

Algorithm	RAPID			RRT		
Metric	T_c (s)	L_p (m)	T_f (s)	T_c (s)	L_p (m)	T_f (s)
Mean	**1.5**	**5.02**	**24.43**	2.55	6.07	33.06
Std.	2.64	0.4	3.35	1.92	1.022	3.69

Table 2: Mean and standard deviation of performance metrics for 10 trials of hardware experiments

increases because paths have to wiggle and curve more to get from the start to end because of more obstacles.

While executing a maneuver in the hardware experiments, we use the same control input that is used to generate the maneuver in the trajectory library rather than perform end to end waypoint tracking. It is observed that due to inherent error present in modelling the maneuver in the trajectory libraries, the actual maneuver executed by the quadrotor when providing the corresponding control input has a slight deviation from the modelled maneuver. This modelling error causes the quadrotor to deviate from its flight path and this deviation increases as the quadrotor progresses towards the terminal position. To decrease this deviation and to keep the quadrotor on the flight path, we select the best maneuver from the library to reach the next waypoint and apply the corresponding control input to the quadrotor.

6.1 Limitations of RAPID

We observe that RAPID is able to quick replan almost all of the time. However, in one trial of hardware experiment (out of 10 trials) and in some software experiments with percentages - RAPID-RRT Extend (2.8%), RAPID-SST Random (0.96%), RAPID-RRT Random (1.13%) and RAPID-SST Extend (3.13%), RAPID was unable to find a quick replan (and thus fell back to using a brute-force replan as specified on line 23 of Algorithm 1). Thus, in our experiments, overall RAPID is successful in quick replanning for a variety of obstacle configurations at least 96.8% (software experiments) of the time. RAPID is unable to perform a quick replan in situations when the original search tree's cost guides it through a narrow passage that the changed dynamics are incapable of maneuvering through. We also observe that RAPID performs worse when the original plan is using the extend operator (see Section 5) since using this operation on average causes less nodes to be added to the graph (than random propagation) and hence less nodes for utilization for RAPID.

7 Conclusion

In this paper, we present the RAPID algorithm for quick replanning under changed dynamical constraints. This algorithm uses the original tree's stored cost information to quickly replan and generate a feasible path to the start location. The advantage of RAPID is that it can be used for various types of

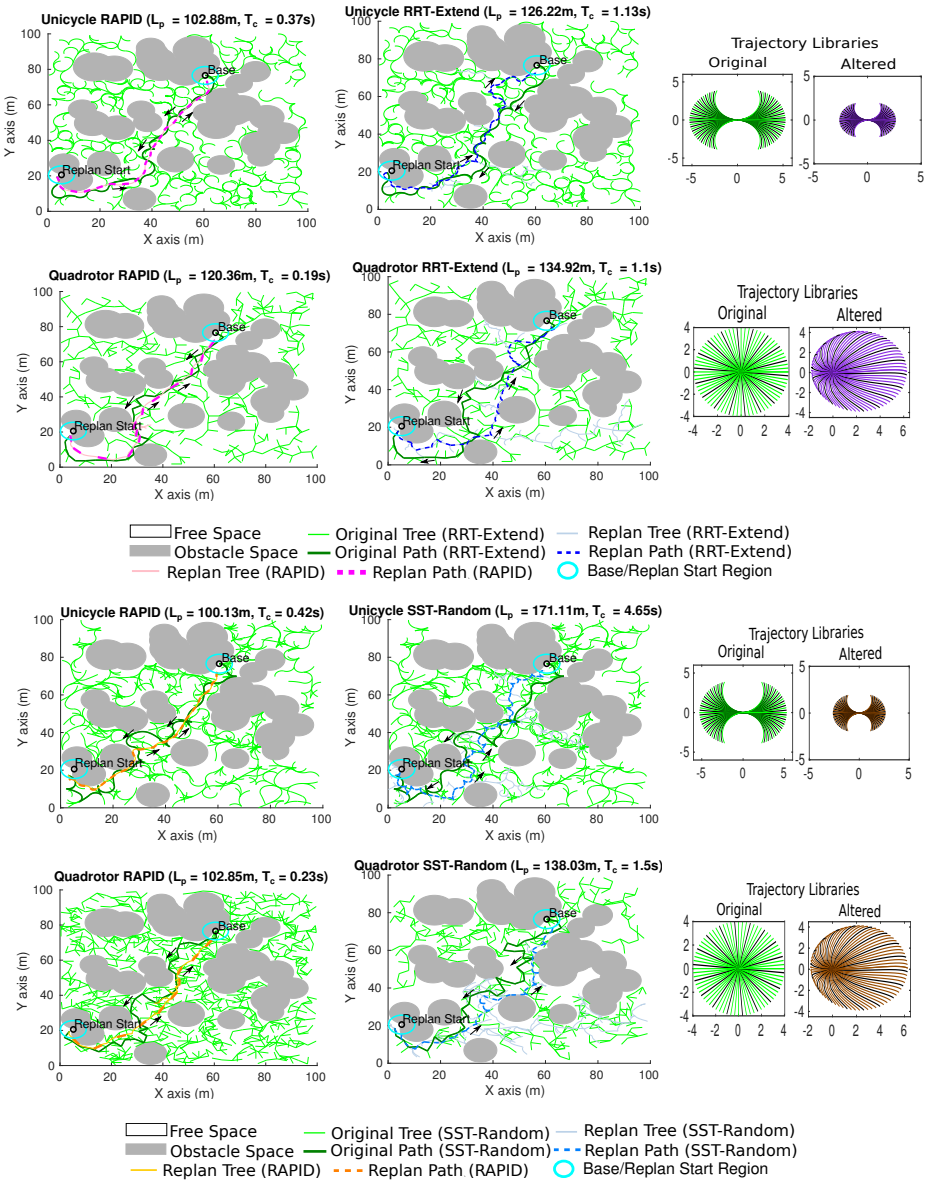

Fig. 7: Figures show replanned paths for Unicycle (top row) and Quadrotor (bottom row) using RRT-Extend (top) and SST-Random (bottom) for original plan. The original and replan maneuver libraries used in the experiments are shown in the last column. The replan trajectory libraries are colored purple and brown to show that the same trajectory library is used for multiple corresponding planning algorithms. Trajectories at 20° increments are highlighted black for clarity.

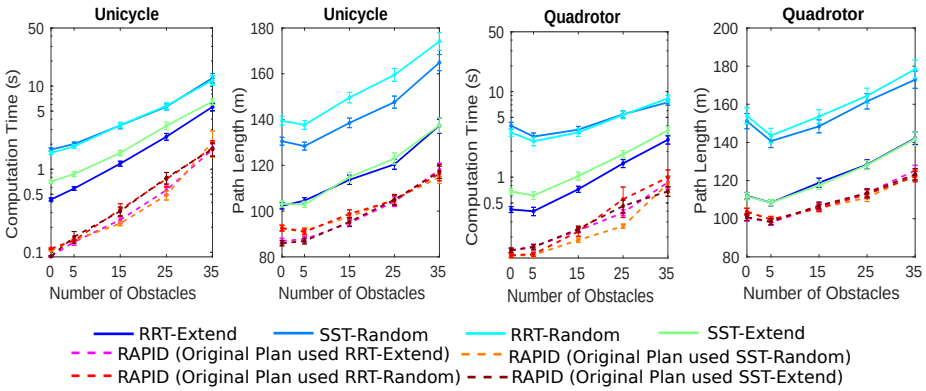

Fig. 8: Figures show completion time (semi-log scale) vs. number of obstacles (left column), path length vs. number of obstacles (right column) for Unicycle (Left) and Quadrotor (Right) respectively. Each point is a mean of values from 100 scenarios with each scenario having 3 trials.

vehicles as it does not require solving the 2-point boundary value problem. We run multiple software and hardware experiments to show RAPID performs better than existing state of the art algorithms (RRT, SST) in computation time, path length and flight time (only hardware) experiments.

Our future work involves testing RAPID in complex environments (e.g. maze) using different changed dynamics for various vehicles. We plan to study using RAPID for replanning to base or goal location whichever is closer to the location of dynamics change. We also plan to analyze the effect of changing heuristic weight ζ on the performance metrics.

Acknowledgements: This work is supported by the Minta Martin Aeronautical Research Fund, University of Maryland (UMD). Sharan Nayak is additionally supported by the Clark Doctoral Fellowship, UMD. We like to thank Chahat Deep Singh and Nitin Sanket from the Perception and Robotics Group (PRG) at UMD for allowing us to use the quadrotor system for our experiments.

References

1. Palunko, Ivana, and Rafael Fierro. "Adaptive control of a quadrotor with dynamic changes in the center of gravity." IFAC Proceedings Volumes 44.1 (2011): 2626-2631.
2. Mueller, Mark W., and Raffaello D'Andrea. "Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers." 2014 IEEE international conference on robotics and automation (ICRA). IEEE, 2014.
3. LaValle, Steven M., and James J. Kuffner Jr. "Randomized kinodynamic planning." The international journal of robotics research 20.5 (2001): 378-400.
4. Li, Yanbo, Zakary Littlefield, and Kostas E. Bekris. "Asymptotically optimal sampling-based kinodynamic planning." The International Journal of Robotics Research 35.5 (2016): 528-564.

5. LaValle, Steven M. Planning algorithms. Cambridge university press, 2006.
6. Go, Jared, Thuc D. Vu, and James J. Kuffner. "Autonomous behaviors for interactive vehicle animations." Graphical models 68.2 (2006): 90-112.
7. Frazzoli, Emilio, Munther A. Dahleh, and Eric Feron. "Real-time motion planning for agile autonomous vehicles." Journal of guidance, control, and dynamics 25.1 (2002): 116-129.
8. Bekris, Kostas E., and Lydia E. Kavraki. "Greedy but safe replanning under kinodynamic constraints." Proceedings 2007 IEEE International Conference on Robotics and Automation. IEEE, 2007.
9. Ferguson, Dave, Nidhi Kalra, and Anthony Stentz. "Replanning with rrts." Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.. IEEE, 2006.
10. Yoshida, Eiichi, and Fumio Kanehiro. "Reactive robot motion using path replanning and deformation." 2011 IEEE International Conference on Robotics and Automation. IEEE, 2011.
11. Otte, Michael, and Emilio Frazzoli. "RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning." The International Journal of Robotics Research 35.7 (2016): 797-822.
12. Urmson, Chris, and Reid Simmons. "Approaches for heuristically biasing RRT growth." Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453). Vol. 2. IEEE, 2003.
13. Otte, Michael, and Nikolaus Correll. "C-FOREST: Parallel shortest path planning with superlinear speedup." IEEE Transactions on Robotics 29.3 (2013): 798-806.
14. Gammell, Jonathan D., Siddhartha S. Srinivasa, and Timothy D. Barfoot. "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic." 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2014.
15. M. P. Strub and J. D. Gammell, "Adaptively Informed Trees (AIT*): Fast Asymptotically Optimal Path Planning through Adaptive Heuristics," 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 2020, pp. 3191-3198, doi: 10.1109/ICRA40945.2020.9197338.
16. Littlefield, Zakary, and Kostas E. Bekris. "Informed asymptotically near-optimal planning for field robots with dynamics." Field and Service Robotics. Springer, Cham, 2018.
17. Branicky, Michael S., Ross A. Knepper, and James J. Kuffner. "Path and trajectory diversity: Theory and algorithms." 2008 IEEE International Conference on Robotics and Automation. IEEE, 2008.
18. Hart, Peter E., 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 (1968): 100-107.
19. Pohl, Ira. "Heuristic search viewed as path finding in a graph." Artificial intelligence 1.3-4 (1970): 193-204.
20. Kuffner, James J., and Steven M. LaValle. "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). Vol. 2. IEEE, 2000.
21. Nitin J. Sanket, Chahat Deep Singh, Cornelia Fermüller, Yiannis Aloimonos. PRGFlyt: AI based indoor quadrotor autonomy framework for navigation and interaction tasks. 2019, https://github.com/prgumd/PRGFlyt/wiki
22. Carona, Ricardo, A. Pedro Aguiar, and Jose Gaspar. "Control of unicycle type robots tracking, path following and point stabilization." (2008).