Trajectory Planning Methods for Exploratory UAV

Griffin Van Anne
University of Colorado Boulder
Boulder, CO
grva7970@colorado.edu

***Abstract*—In this project, several motion planning algorithms are implemented for an exploratory UAV. The simplest planner implemented is a geometric RRT planner, which generates collision free paths to get the UAV from a starting configuration to goal region. A kinodynamic RRT algorithm is used to generate trajectories which respect the motion model of the UAV. To minimize the energy expended along these trajectories, an SST algorithm is implemented which improves the efficiency of resulting solutions. Finally a high level task plan is formalized using an LTL formulation.**

I. Introduction

On April 19th, 2021, NASA's Mars helicopter *Ingenuity* recorded the first powered and controlled flight on a different planet. Since then, the craft has logged over 30 minutes in flight while exploring the terrain of Mars [4]. This is an impressive milestone for the exploration of extraterrestrial worlds and it can only be expected more capable and ambitious missions will take place in the future.

This project focuses on an exploratory unmanned aerial vehicle(UAV), similar in use to the *Ingenuity* copter. The goal of the UAV is to explore several goal regions of interest, while avoiding collisions with environment obstacles. These obstacles resemble large buildings in an urban environment. Several motion planning problems will be addressed, including geometric and kinodynamic planning. Producing energy efficient trajectories is an important objective for exploratory UAVs, so an optimizing algorithm is implemented to reduce total path energy. Operating this UAV in communication inhibited locations could introduce the need for increased autonomy. For this reason, high level task planning for the UAV is also explored.

II. Problem Statement

In this paper, four main subproblems are addressed:

Subproblem 1: Geometric Planner

The first subproblem is to implement a path-planning algorithm which can generate valid paths from the starting configuration to any of the goal configurations. This planner must generate three-dimensional collision-free paths that satisfy only geometric constraints - differential constraints are not considered in this problem.

Subproblem 2: Kinodynamic Planner

For this subproblem, a planner must be implemented which can generate UAV trajectories which take off from the start configuration and land in a selected goal region, while satisfying differential constraints. These trajectories must also avoid collisions with environment obstacles, as well as satisfy the landing conditions(see III-B).

Subproblem 3: Energy-Optimizing Planner

A kinodynamic planner must be implemented which generates optimal trajectories from start to goal with respect to the control energy expended along that path. The results of this optimal planner must be benchmarked against the kinodynamic planner from subproblem 2.

Subproblem 4: Task Planning

In this subproblem a task planning algorithm will be introduced in order for the UAV to complete mission objectives autonomously. This algorithm will create plans for the UAV to execute a sequence of data collection, data transmission, and refuelling steps.

III. Methodology

To implement solutions for each subproblem the Open Motion Planning Library(OMPL) was used [1]. A relevant planner was selected from OMPL and implemented in a dedicated C++ program. Results from these programs were written to text files and processed by MATLAB for visualization.

A. Environment Creation

The environment used for this paper comprised of rectangular obstacles, square goal regions, and a rectangular UAV representation. Each obstacle has a randomly generated size between 30-150 meters in height, and 20-90 meters in width/depth. The position of each obstacles were initially chosen at random, then adjusted manually to create interesting goal locations. Each goal region is 16×16 meters, and strategically placed to present different levels of challenge to the planners. The UAV is represented by a 2×2×0.5 meter rectangle. It is assumed for simplicity throughout this project that the UAV orientation remains approximately flat and does not rotate, which is why a relatively large bounding box was chosen for its representation. The state sampling region, used in the geometric planner, was chosen to be 250×250×200 meters in size. This is ample space to be able to generate valid geometric plans.

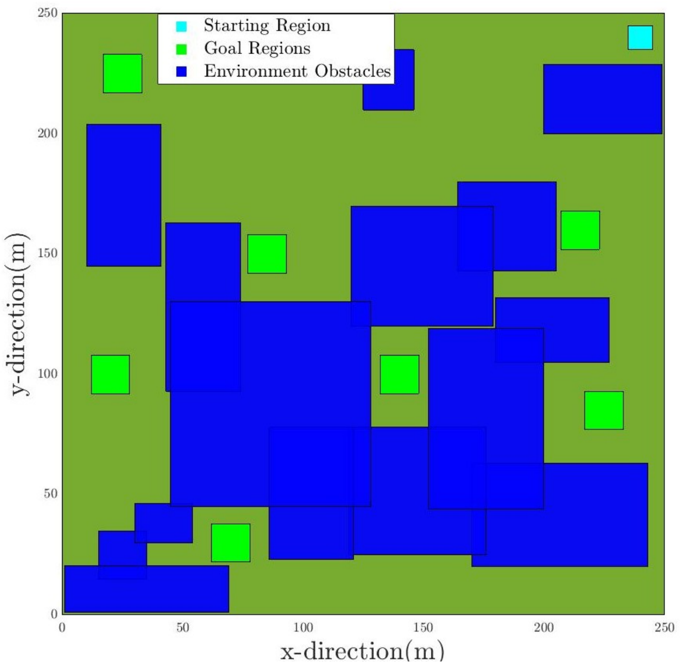


Fig. 1. Overhead view of the planning environment

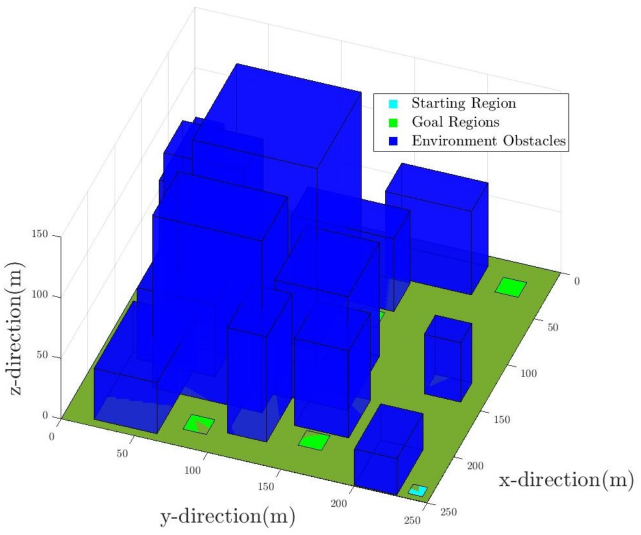


Fig. 2. Side view of the planning environment

B. State Validity and Goal Checking

In each of the subproblems a state validity checker was used to return whether or not a UAV state is in a valid configuration. Since the environment obstacles and UAV are represented by rectangles, an AABB collision checking method was implemented [2].

For the geometric problem, a state has reached the goal if it's x and y positions are within the goal region bounds, and if the z state is below a threshold of 0.1 meters. These same criteria are kept for the kinodynamic planners in addition to checks on the impact velocity of the UAV. Vertical impact velocity, $\dot{z}$, must be less than or equal to $2.5m/s$ in order to have achieved a valid goal configuration. Total horizontal velocity must be less than or equal to $5m/s$ in order to be in a goal configuration. These are both fairly aggressive landing speeds, but were chosen to make the goal criteria easier to satisfy. In order to interface with the OMPL planners, a `GoalSampleableRegion` class is implemented which contains methods for checking distance to goal as well as sampling from the goal regions.

C. Subproblem 1 Planner

The planner used for the geometric problem is a goal-biased RRT. This planner was chosen for its simplicity in implementation and its effectiveness at planning in higher-dimensional space. As mentioned earlier, only translational motion is considered for the UAV, making the topology of the planning space $\mathbb{R}^3$. The OMPL RRT planner has a default goal-bias sampling rate of 0.05 - this was kept unchanged.

D. Motion Model

For the kinodynamic planning problems a simple UAV motion model was developed. This model makes use of a single thruster which can be pivoted to actuate its thrust vector:

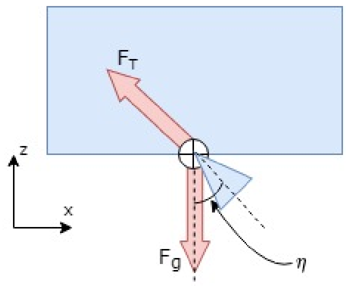


Fig. 3. Free body diagram of UAV in x-z plane.

The angle η drawn in the figure above displaces the thruster in the x-z plane. η is restricted in this model from $-\pi/8$ to $\pi/8$, which was chosen after testing several different ranges with the kinodynamic planner. Note that the thruster pivots around the center of mass - this simplification was made so that the rotational motion of the UAV could be ignored. Shown below is a sketch of the thrust force vector in the x-y plane:

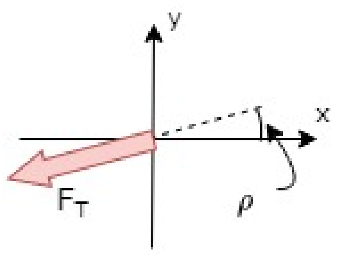


Fig. 4. Thrust force vector in the x-y plane.

The angle ρ rotates the thruster about the z-axis. With allowing this angle to range from $-\pi/2$ to $\pi/2$, the thrust force vector can fully span the x-y plane. Thrust has a lower bound of 0, and an upper bound of $2g$.

The full equations of motion for this system are shown below:

$$\begin{aligned} \ddot{x} &= -T\sin(\eta)\cos(\rho) \\ \ddot{y} &= -T\sin(\eta)\sin(\rho) \\ \ddot{z} &= T\cos(\eta) - g \end{aligned} \tag{1}$$

Note T is normalized by dividing by the UAV mass in order to eliminate that parameter from the EOM.

The resulting state vector for the system is:

$$q = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \eta & \rho & T \end{bmatrix} \tag{2}$$

This is a 9-dimensional state space, with a topology of $\mathbb{R}^9$. Although ρ and η are angles, they exist in $\mathbb{R}$ space because of their fixed range. Only the first 6 states in this vector are used to check for goal satisfaction, as it does not particularly matter what the η angle is upon landing for instance.

The control space for the UAV consists of three real valued inputs:

$$u = \begin{bmatrix} \dot{\eta} & \dot{\rho} & \dot{T} \end{bmatrix} \tag{3}$$

The ranges for each of these inputs are shown below, which were arbitrarily chosen such that the UAV can fully actuate from control extremes in two seconds:

$$u = \begin{cases} \dot{\eta} & \in [-\pi/8,\ \pi/8] rad/s \\ \dot{\rho} & \in [-\pi/2,\ \pi/2] rad/s \\ \dot{T} & \in [-g,\ g] N/kg \end{cases} \tag{4}$$

E. Subproblem 2 Planner

The kinodynamic planner selected was the OMPL controls RRT, which is designed to work for systems with differential constraints. To implement these constraints in the planner, an ODE method is built to interface with OMPL's `ODEBasicSolver` class. This method is named `KinematicUAVODE` and takes a state and control input and outputs the state derivative.

F. Subproblem 3 Planner

The optimizing planner chosen was OMPL's control Stable Sparse RRT(SST). SST is a sampling based anytime planner which achieves asymptotic near optimal solutions with respect to an objective [3]. By default, OMPL has several optimization objectives, non however are compatible with using control energy as a cost function. This required a custom optimization objective class to be created, `PathEnergyObjective`, which is derived from the `StateCostIntegralObjective` class. This class simply defines the cost of a state to be: $C_2 = (T_2 + T_1)/2 \times \Delta t + C_1$, where T_1 and C_1 is the thrust and cost respectively at the prior state, and Δt is the time between states.

G. Subproblem 4 Planner

For this subproblem there has been defined three separate region categories for the environment: gathering regions where the UAV must land at to gather data, transmission regions where the UAV will transmit this data, and refueling regions. These regions are displayed in figure 5

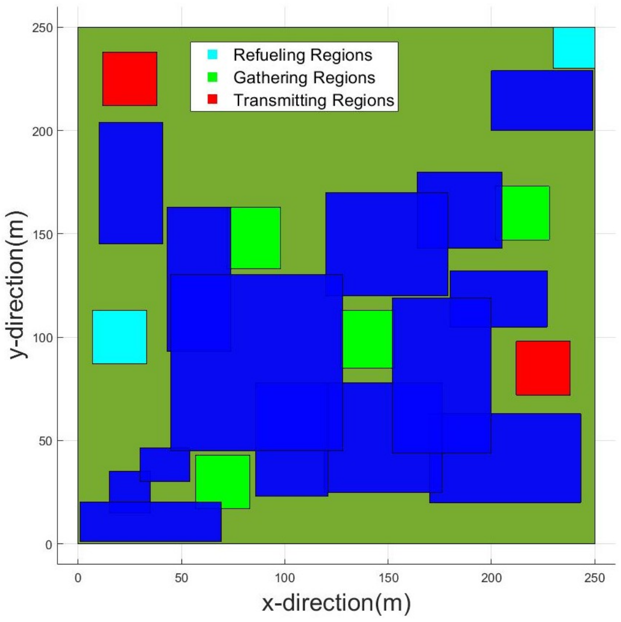


Fig. 5. Regions used for the purposes of high-level task planning.

It is assumed that the UAV has limited onboard memory, and must transmit region data before gathering more. Linear Temporal Logic(LTL) is used to formalize the required behavior for the UAV. The set of propositions for this system are as follows: p_G = Gather Data, p_T = Transmit Data, p_R = Recharge UAV.

$$\phi = GFp_R \wedge GFp_G \wedge G(p_G \rightarrow (\neg p_G\ U\ p_T)) \tag{5}$$

This LTL formula reads as: "The UAV will always eventually recharge, and always eventually gather data. Once data is gathered, the UAV must avoid gathering more data until the data has been transmitted." This formulation allows for the case where refuelling is required after data gathering to reach the transmission sight.

Implementation of this LTL formulation using OMPL was **not** achieved in this project.

IV. Results

A. Subproblem 1

The RRT planner implemented to solve the geometric start-goal path planning problem was very successful. An example of a successful path is shown in the figure 6.

Since this is a purely geometric planner and does not need to abide by differential constraints, the paths found are often

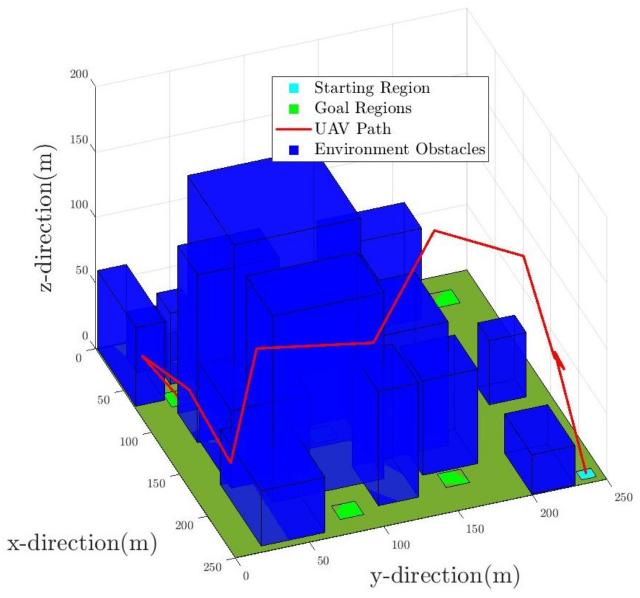


Fig. 6. Geometric path from starting pose to goal configuration, using RRT planner.

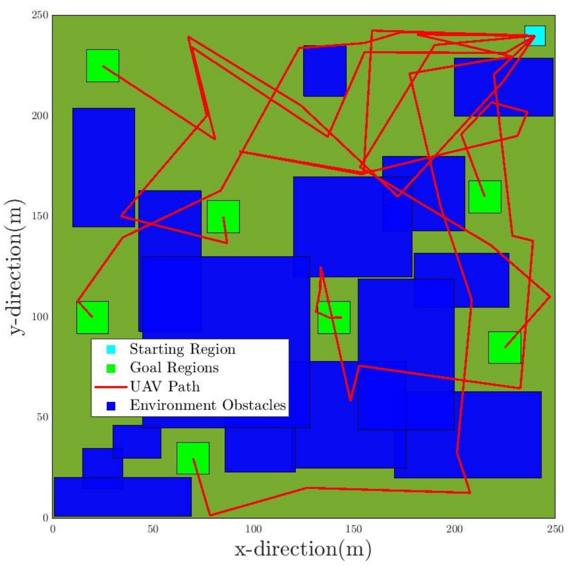


Fig. 7. Geometric paths from start to each goal region.

quite jagged. Figure 7 shows an overhead view of a valid plan to each of the goal regions.

The planner was able to successfully find a path in under 1 second, for all tests ran. This is expected as goal-bias RRT can quickly expand its tree to each goal region, and the simple AABB collision checking results in fast state validity checking. Since only geometric constraints are considered in this planner, it can very reliably find valid paths to the goal regions.

B. Subproblem 2

It was found that the planner often failed to solve for a single valid trajectory over this timespan. As a result, instead of expanding for a single goal during goal sampling, a random goal sample is selected from the set of regions. This results in much better reliability in producing results, albeit it is no longer solving for a specific goal. After these changes, the algorithm was still only able to find a valid solution 37.5% of runs for a planning time of 120 seconds. Longer planning times did not seem to have a significant effect on success rate.

An interesting resulting trajectory from the RRT planner is shown in the two figures 8 and 9, at different view angles.

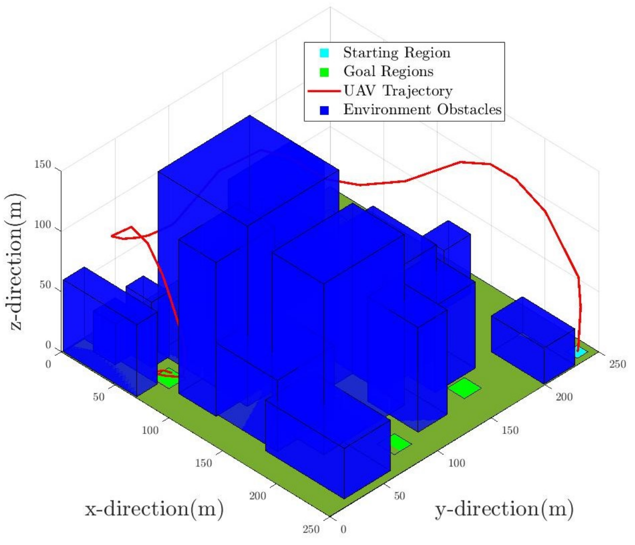


Fig. 8. First view of the selected kinodynamic trajectory.

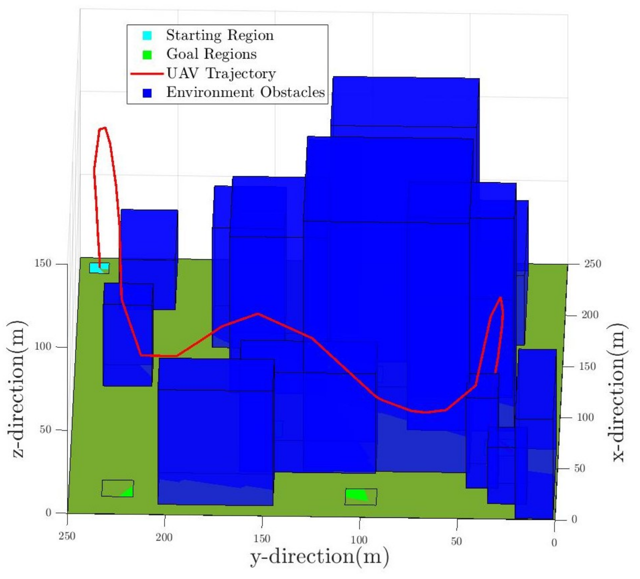


Fig. 9. Second view of the selected kinodynamic trajectory.

These results clearly show ‘smoother’ paths compared to the geometric planner, which is expected as the kinodynamic trajectories are generated from the UAV equations of motion. To verify this is the case, the solution controls were ran in a MATLAB simulation using ode45 which showed that these controls did indeed produce the paths output from the OMPL planner. Note that any jaggedness visible in the kinodynamic trajectories are a result of discretizing the paths.

While the kinodynamic planner was not able to produce reliable success, it was successful in generating trajectories for each goal region (see figure 10).

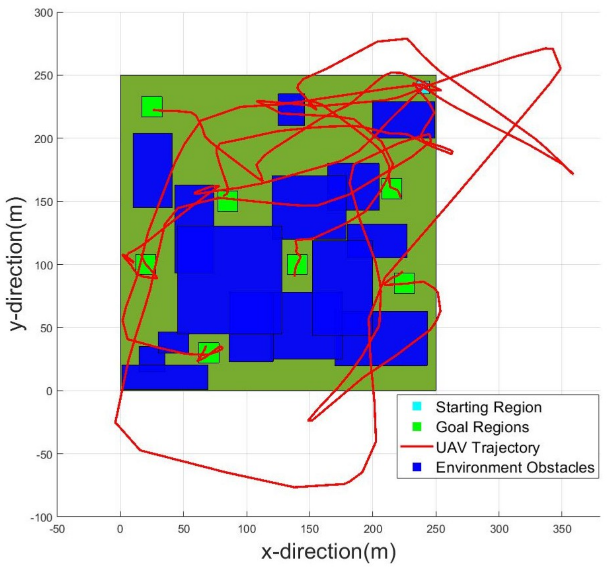


Fig. 10. Overhead view of kinodynamic trajectories reaching each goal region.

C. Subproblem 3

Two design parameters for the SST planner are pruning radius(δ_s), and neighborhood radius(δ_{BN}). Selecting a large δ_s will result in more pruning - a δ_s that is too large can inhibit the planner from discovering small passages [3]. Selecting a δ_{BN} that is too large can result in a slower expansion of the tree. With these considerations in mind, several different values of these parameters were tested. It was found that $\delta_s = 1$ and $\delta_{BN} = 3$ gave the best results although more testing would be required to find the optimal parameters for this system.

The SST planner was given 500 seconds to plan over. It was found that the planner often failed to solve for a single valid trajectory over this timespan. As a result, the goal region area was expanded by 10 meters in both the x and y directions, and the random goal biasing was kept from the RRT planner. These changes help with understanding the optimization performance of the planner.

The overhead view of the intermediate and final trajectories from a single SST run are shown in figure 11 and 12. It is interesting to see the visualization of how the 'optimal' trajectory jumps from goal to goal as the energy is driven down, with the final trajectory going to one of the closest goal regions. Each one of these trajectories shares the same initial path, but diverge as they spread out to different goals.

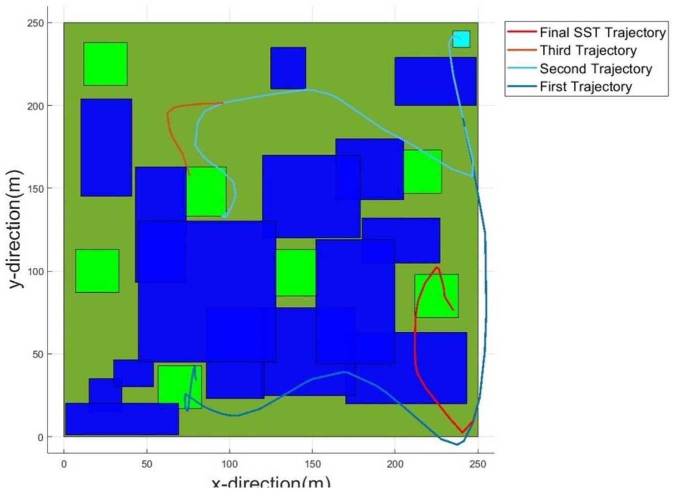


Fig. 11. Overhead view of intermediate and final trajectories generated from SST planner.

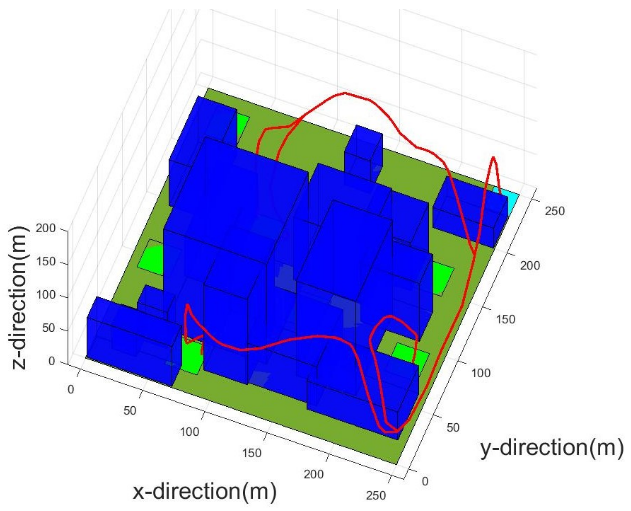


Fig. 12. Different view of the improving trajectories produced by a single SST run.

The thrust force and energy along these paths ares displayed in figure 13. The energy plot shows how the total path energy evolves for each trajectory, with the final expended energy upon landing marked by the red squares.

Figure 14 shows the path cost evolution versus SST solution iteration for many different SST runs. This plot clearly shows the path energy decreases with each iteration, although many of the SST runs did not achieve more than 3 valid solutions throughout its run-time. The big jumps in cost decrease for several of the runs can be attributed to the planner switching to a different goal trajectory.

The most energy conservative path found throughout all of testing is shown in figure 15. This path takes a very short

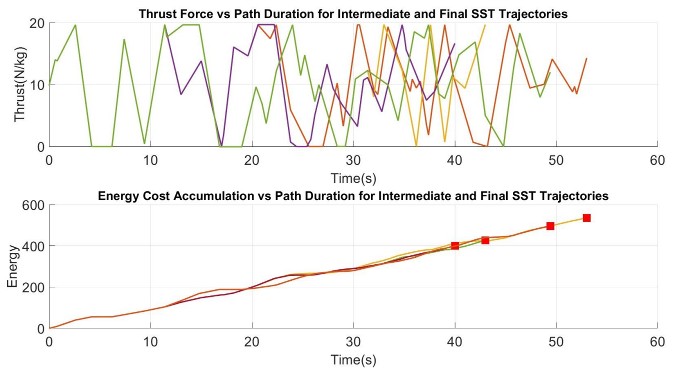


Fig. 13. Thrust and Energy versus time for SST intermediate and final solutions.

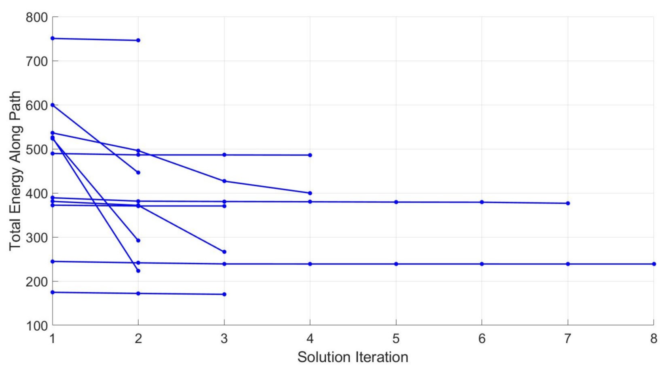


Fig. 14. Energy versus solution iteration for SST planner over 10 runs.

trajectory to the nearest goal, achieving a final objective value of 170.1.

Formal benchmarking was not performed for this planner as the very long execution time of 500 seconds restricted the testing ability. It would be recommended in the future to test this planner out on a scaled down environment first.

D. Subproblem 4

Subproblem 4 was not implemented in this project due to time constraints.

V. Discussion

Comparing the geometric to kinodynamic planning results, there is a clear jump in complexity that adding differential constraints brings to the table. The exponential relationship between space dimensionality and volume means the sampling based algorithm takes *much* longer to adequately cover the planning space.

Using a default sampling based planner, such as RRT or SST may not be the most efficient way of generating valid landing trajectories. Each take-off and landing maneuver share similar stages of movement. The take-off phase features a primarily vertical trajectory, the traversal phase translates the UAV to approximately overhead the landing zone, and the landing zone uses a gentle vertical descent to arrive at the

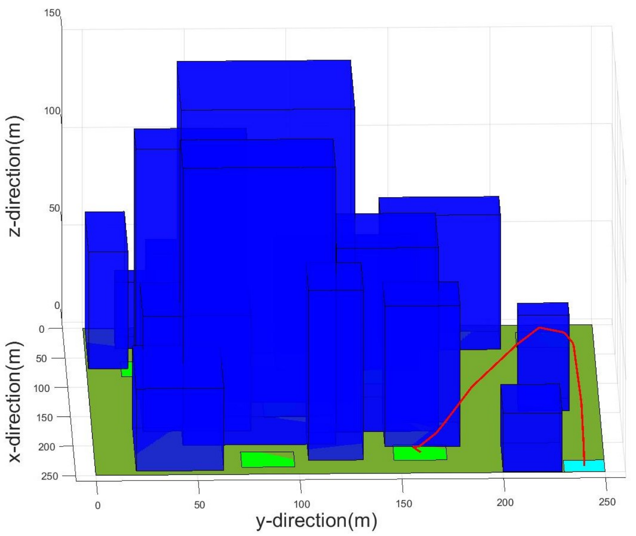


Fig. 15. Most energy optimal trajectory found in all of testing.

goal. Splitting the planning problem into these three phases could help inform the planner more useful control regions to sample. For instance, during the approximate take-off phase, the planning algorithm should primarily focus on sampling low $\dot{\rho}$ and $\dot{\eta}$ controls, and a higher $\dot{T}$. This would potentially result in much fewer 'dead-end' branches of the tree where trajectories have a low-likelihood of achieving a valid goal configuration. Adjustment of the control sampling region could occur directly after selection of the nearest node to a sampled node - based on the state of this node, the sampled extending control could take on a different distribution.

The SST planner showed the ability to optimize paths with respect to energy, however the difficulty in finding valid kinodynamic trajectories inhibited its ability to converge to the approximate optimal solution. The results of this planner were highly dependent on the first valid trajectory found - if this trajectory could easily be branched to more energy efficient goals, it often would. If it could not be branched to other goals, it typically only found 1-2 more valid solutions with minimal decrease in cost between them. It would have been wise to construct a second, scaled down planning environment to test this planner on. Faster solutions would be acquired and better benchmarking of results could be performed in order to find the best pruning and neighborhood radii. Attempting to characterize performance of this planner on the full scale environment was difficult, as the planning time required was upwards of 600 seconds.

Implementing a LTL planner for this environment would have been a very fun challenge, had time permitted.

VI. Conclusion

Several key motion planning challenges were addressed in this project - to varying success. The use of OMPL for this project was a powerful tool, and allows for focus to be spent on more creative aspects of motion planning. That being said, there is a learning curve associated with its use that had to be overcome first.

Acknowledgment

Thank you to Morteza Lahijanian and Justin Kottinger for their instruction throughout the semester.

References

[1] I. A. Sucan, M. Moll and L. E. Kavraki, ”The Open Motion Planning Library,” in IEEE Robotics and Automation Magazine, vol. 19, no. 4, pp. 72-82, Dec. 2012, doi: 10.1109/MRA.2012.2205651.

[2] 3D collision detection - game development: MDN. Game development — MDN. (n.d.). Retrieved December 14, 2021, from https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection.

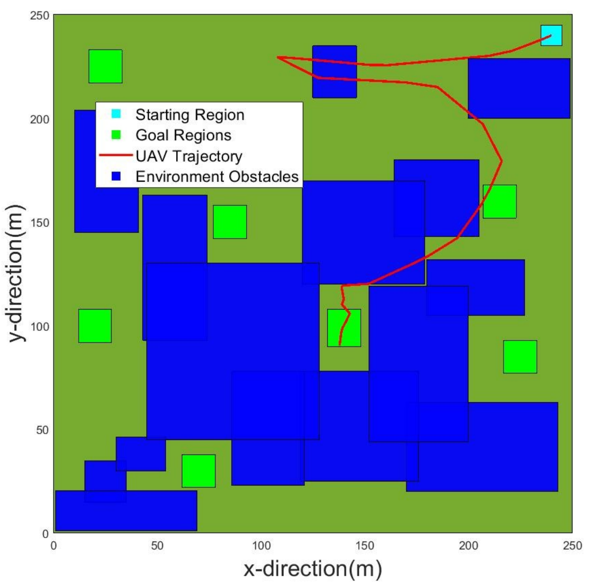
[3] Li, Yanbo, Zakary Littlefield, and Kostas E. Bekris. ”Asymptotically optimal sampling-based kinodynamic planning.” The International Journal of Robotics Research 35.5 (2016): 528-564.

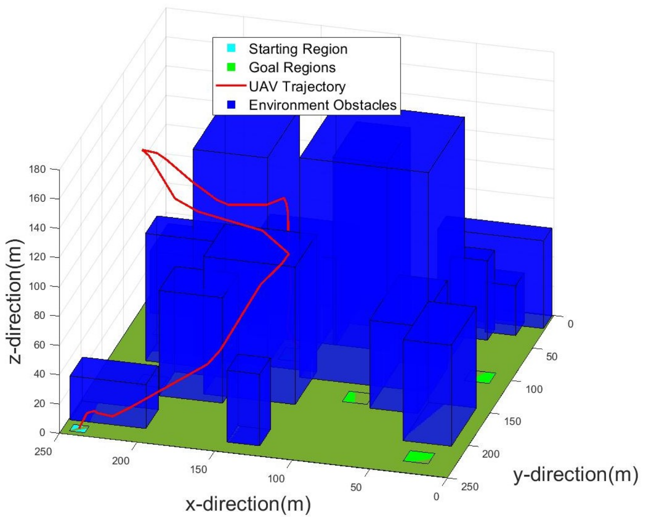
[4] Greicius, Tony. “NASA’s Ingenuity Mars Helicopter Reaches a Total of 30 Minutes Aloft.” NASA, NASA, 15 Dec. 2021, https://www.nasa.gov/feature/jpl/nasa-s-ingenuity-mars-helicopter-reaches-a-total-of-30-minutes-aloft.

Appendix

All code developed for this project can be found at the following Github address: https://github.com/griffjv/Motion_Planning_Final

A. Additional Kinodynamic Trajectories:


Starting Region
Goal Regions
UAV Trajectory
Environment Obstacles
y-direction(m)
x-direction(m)


Starting Region
Goal Regions
UAV Trajectory
Environment Obstacles
z-direction(m)
x-direction(m)
y-direction(m)