Analysis of Non-Convex Trajectory Optimization Techniques for Quadrotor in Forest Environments

Frank Gonzalez
Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, MA
frankgon@mit.edu

Shreya Gupta
Computer Science and Engineering
Massachusetts Institute of Technology
Cambridge, MA
shreyag@mit.edu

Kaitlin Zareno
Computation and Cognition
Massachusetts Institute of Technology
Cambridge, MA
kzareno@mit.edu

***Abstract*—Certain environments can make traversing by land vehicle extremely challenging, so it is important to move towards navigation of these dense environments via unmanned aerial vehicles (UAVs). When moving through these dense environments, the conservation of resources and determining the optimal motion of the UAV are essential to efficiently completing the task at hand, for example flying from an initial location to a final location. Thus, this paper presents a comparative analysis of three non-convex trajectory optimization techniques: direct transcription, direct collocation, and direct shooting for a quadrotor navigating through a forest environment. As a system with non-linear dynamics, certain optimization techniques, such as the Linear Quadratic Regulator (LQR) can fail in cases in which linearization is not a sufficient approximation of the non-linear dynamics. Thus, we need to examine a class of techniques that can accommodate for this non-linearity in order to properly obtain an optimal control solution.**

The question remains: what are the relative performances of direct transcription, direct shooting, and direct collocation in generating a trajectory?

In our analysis of these trajectory optimization techniques, we consider both the infinite and finite lookahead cases. For both the infinite and finite lookahead case, we implement three trajectory optimization techniques that are tested in forest environments of different densities. The infinite lookahead case gives us insight into forest navigation as an offline planning problem, while the finite lookahead case frames it as an online planning problem. We drive this shift from offline planning to online planning by limiting the field of view of the quadrotor. We implement each of these optimization techniques using the Drake package in our simulated forest environment.

We determined that direct collocation is the best-performing algorithm for our particular use case, followed by direct transcription, and finally direct shooting. In all cases, the finite lookahead formulation remained both more robust and ran faster than their infinite lookahead counterparts.

***Index Terms*—trajectory optimization, quadrotor, planning, infinite lookahead, finite lookahead**

I. Introduction

Natural environments that contain dense, hard-to-navigate terrain can make traversal by land vehicle extremely challenging. In order to side-step these challenges, we move towards autonomous navigation of these forested environments via unmanned aerial vehicles (UAVs). Autonomous drones are currently used to solve many problems such as mapping and delivery, and there are many applications and advantages of UAV solutions in natural environments. The three of us are fascinated by drones and the ongoing challenges of navigating large, unknown environments.

After discussing the three non-convex trajectory optimization techniques in class, we wanted to better understand these differences in practice. The optimization techniques were framed as different representations of the same problem; we wanted to see how these changes in representation would affect outcomes.

Although the infinite lookahead case gives a good understanding of trajectories that could be planned offline, we wanted to gain a better understanding of the online planning problem. We formulate the online planning problem by limiting the lookahead of the trajectory optimization algorithms in order to mimic a quadrotor with a limited field of view. In this case, the quadrotor is only able to plan in accordance with what it can see and thus must make an optimal decision based on the partial information. We wanted to better understand how the finite lookahead would change the generated optimal trajectory and if the different optimization methods would provide different results in this scenario.

II. Related Work

As mentioned in the *Introduction*, navigating a UAV through cluttered terrain is an active area of research. Our methodology for navigation is based on nonconvex trajectory optimization, but the ways in which others have solved this problem are varied. Methods are typically clustered into two types of formulations: keeping the UAV away from obstacles and keeping the UAV within free space.

One method of keeping the UAV away from obstacles involves generating trajectories that explicitly avoid occupied space. For example, [1] proposes a method to optimize over several piecewise polynomials to define a trajectory. They

approach this by first generating a path using a state space search algorithm such as RRT* or A*. Then, a minimal set of waypoints is used to construct a piecewise polynomial that connects the waypoints. In the event that a generated piecewise function is not collision free, a new waypoint is added in the middle of the polynomial, and the entire trajectory is reoptimized. The specific formulation of the polynomials ensures that the newly added waypoint is indeed collision-free.

The Graphs of Convex Sets (GCS) method, initially introduced in [2], provides an example of a problem formulation focused on keeping the UAV within unobstructed areas. The GCS approach involves dividing the obstacle-free regions into convex, "safe" regions. To traverse between these sets, the robot must follow a trajectory that remains entirely within the safe regions. The proposed solution was tested by weaving a quadrotor through a randomly generated building, which is a task quite similar to the one tackled in this paper.

We consider our work to be in the "keeping the UAV away from obstacles" category and will discuss exactly how we implemented it in the rest of this paper.

III. Environment

For our project, we implement a 2D environment of a forest consisting of a quadrotor instance. The forest is a collection of *Tree* instances that the quadrotor must avoid, as can be seen in Fig. 1.

A. Final Environment Representation

Each *Tree* instance is a randomly initialized stationary point mass with a *tree_radius* randomly chosen from the range of [0, 4]m. To ensure reproducibility and comparability of results, our program offers the option to set a fixed seed. Fixing the seed ensures that the trees in the lower density environments also appear in the higher density environments, as demonstrated in Fig. 2. Additionally, despite visualizing a 2D environment, we treat the trees as if they extend infinitely in the z direction, forming cylinders in a 3D space. This approach prevents the quadrotor from flying over a tree, necessitating the planning of optimal trajectories that circumvent them. Furthermore, as trajectory optimization algorithms can get trapped in local minima with a non-convex environmental representation, we constrain our forest to a convex representation. We accomplish this by defining a minimum distance between trees of 1m, otherwise known as the *tree_buffer*. To reiterate, the minimum distance between two adjacent trees must be greater than the *tree_buffer*, guaranteeing that each tree retains a distinct convex shape in our space and prevents the occurrence of locally optimal solution trajectories.

The *Quadrotor* instance is a *MultibodyPlant* instance that is contains propellor inputs and Roll, Pitch, Yaw dynamics. This mirrors the quadrotor setup defined in the Chapter 3 Deepnote Workspace [2]. In order to integrate the Quadrotor with the trajectory optimization techniques and support the operations of a Mathematical Program, we create an *AutoDiff* instance of our *Quadrotor*.

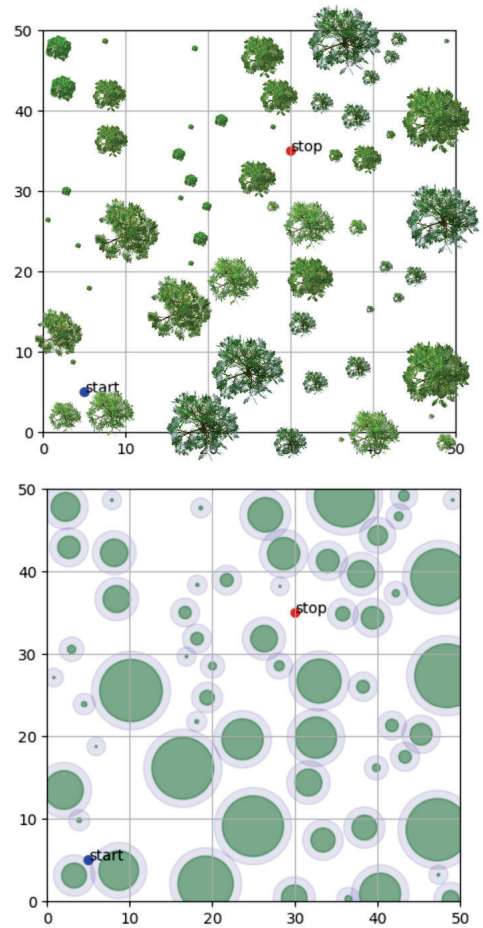


Fig. 1. 2D Visual Representation of Convex Forest Environment

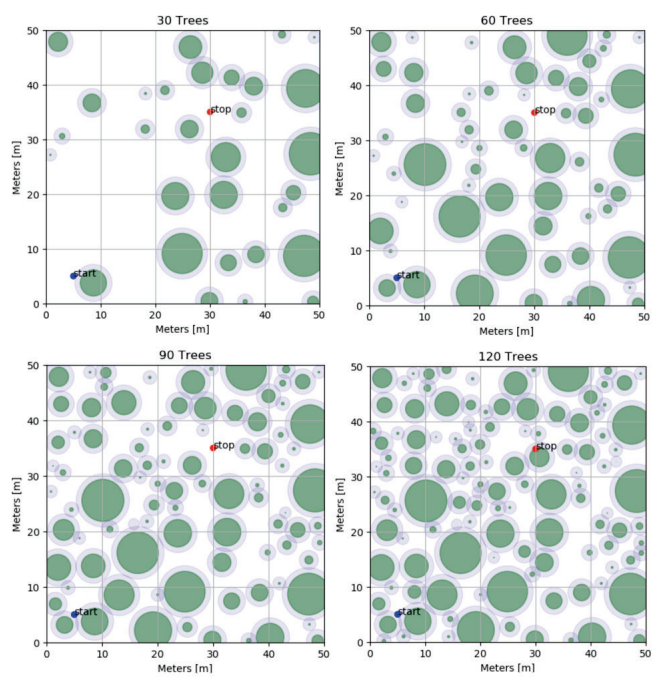


Fig. 2. Seeded Forests of Increasing Density: 30, 60, 90, and 120 trees

IV. Problem Formulation

In order to formulate the problem, we first define the variables which we are manipulating and considering.
State Vector: $s = [x, y, z, \psi, \theta, \phi, \dot{x}, \dot{y}, \dot{z}, \dot{\psi}, \dot{\theta}, \dot{\phi}]$
Input Vector: $u = [\omega_0, \omega_1, \omega_2, \omega_3]$
Input Vector direct shooting: ${u_s}^1 = [Q_0, Q_1, Q_2, Q_3, Q_4, Q_5]$
Initial state: s_0
Goal state = s_f
Horizon radius = r
Trajectory: $s(t)$
Input Trajectory: $u(t)$
Input Trajectory direct shooting: $u_s(t)$
Forest: F
Tree: T

V. Approach

To generate trajectories for the quadrotor, we look at a suite of trajectory optimization algorithms. Each optimization algorithm requires us to discretize the quadrotor dynamics. For our analysis, we chose to discretize each optimization into $N = 50$ timestamps with a timestep of $ts = 0.2s$. A mathematical description of each algorithm is provided below:

Direct Transcription: The discretized state and input trajectories are decision variables subject to costs and constraints specific to the problem formulation.

$$\begin{aligned} \min_{s[\cdot],u[\cdot]} \quad & \ell_f(s[N]) + \sum_{n=0}^{N-1} \ell(s[n], u[n]) \\ \text{s.t.} \quad & s[n+1] = f(s[n], u[n]), \forall n \in [0, N-1] \\ & s[0] = s_0 \\ & \text{+ Additional Constraints} \end{aligned}$$

Direct Collocation: The state and input trajectories are re-expressed as piecewise polynomials and sampled for "breakpoints" that become the decision variables. We additionally impose dynamics constraints by calculating intermediate "collocation points".

$$\begin{aligned} \min_{s[\cdot],u[\cdot]} \quad & \ell_f(s[N]) + \sum_{n=0}^{N-1} \ell(s[n], u[n]) \\ \text{s.t.} \quad & \dot{s}(t_{c,n}) = f(s(t_{c,n}), u[n]), \forall n \in [0, N-1] \\ & s[0] = s_0 \\ & \text{+ Additional Constraints} \end{aligned}$$

Direct Shooting: We express the state trajectory as a function of the input trajectory, massively reducing the number of decision variables in the optimization problem.

$$\begin{aligned} \min_{u_s[\cdot]} \quad & \ell_f(s[N]) + \sum_{n=0}^{N-1} \ell(s[n], u_s[n]) \\ & s[n+1] = f(s[n], u_s[n]), \forall n \in [0, N-1]^2 \end{aligned}$$

[1]This control input is specifically for direct shooting. It uses generalized forces rather than propellor inputs. The reasoning for this difference is explained in the *Iterations: Quadrotor Representation* section.

The additional constraints and costs vary between optimization algorithms and infinite lookahead vs finite lookahead formulations.

A. Infinite Lookahead: Direct Transcription and Direct Collocation

In order for the proposed solution from direct transcription and direct collocation to be satisfactory, it must adhere to the following requirements.

1) Constraint Formulation:

a) $s[N] = s_f$
A final state constraint to enforce that the final state of the trajectory matches the goal state.

b) $\forall n, s[n][2] >= 0$
A height constraint to keep the UAV at or above ground level.

c) $\forall n, \forall T \in F, ||s[n][0:2] - T.position|| >= T.radius$
A tree constraint to ensure the UAV does not choose a trajectory that goes through a tree.

d) An equal time step constraint. This constraint is only for direct collocation.

2) Cost Formulation: The cost function contains only the running actuation cost.

$$L_{total} = L_{actuation}$$

Actuation cost is defined by the cost of the squared control inputs.

$$L_{actuation} = \sum_{i=0}^{3} \omega_i^2$$

B. Infinite Lookahead: Direct Shooting

1) Cost Formulation: The cost function is a summation of the running costs for actuation, distance to the goal, and tree breaching.

$$L_{total} = L_{actuation} + L_{goal} + L_{tree}$$

Actuation cost is defined by the cost of the squared control inputs.

$$L_{actuation} = \sum_{i=0}^{5} Q_i^2$$

The distance to the goal cost is defined as the scaled cost of the squared x and y deviation. In our results, c was set to 10.

$$dist = s[0:2] - s_g[n][0:2]$$

$$L_{goal} = c \sum_{i=0}^{1} dist[i]^2$$

The tree breaching cost is defined as a shifted inverse tangent function that imposes a high penalty for being inside the tree radius and a negligible cost for being outside the tree.

[2]This is not a constraint for the optimization, but rather enforced via a rollout.

The exact function used is below, with tunable parameters c and k representing the maximum cost of being inside the tree and the steepness of the cost decay at the tree boundary. In our results, c and k were both set to 2000.

$$dist(T) = ||s[n][0:2] - T.position||$$

$$L_{tree} = \sum_{T \in F} c(-\arctan k(dist(T) - T.radius) + \frac{\pi}{2})$$

C. Additional Constraints and Costs for Finite Lookahead

In the case of finite lookahead, we add additional constraints to ensure that our program works properly. More specifically, we run an optimization to determine an x and y location for the subgoal. Again, this optimization only determines an x and y, so the other values in the state vector are set to 0. For simplicity, let $s_{g,p}$ represent the first two state values being optimized.

1) Subgoal Constraints:

a) $\sum(s_{g,p} - T.position)^2 \geq T.radius^2$
This constraint ensures that the subgoal location does not exist within a tree.

b) $\sum(s_{g,p} - current_state[0:2])^2 \leq r^2$
This constraint requires that the subgoal location exists within the horizon.

2) Subgoal Cost: We impose a single cost for the subgoal calculation: distance from the global goal.

$$deviation = end[0:2] - s_{g,p}$$

$$\min_{s_{g,p}} \quad \sum_{i=0}^{1} deviation[i]^2$$

D. Iterations: Quadrotor Representation

We iterated over a few representations when choosing how to model the quadrotor. Each representation came with its own set of nuances that either made us switch to another representation or make alterations to our problem formulation.

1) QuadrotorPlant: The first representation was the QuadrotorPlant class from the Drake package [3]. This representation used Euler angles, which led to gimbal lock within the dynamics engine, causing the optimizers to fail. We attempted to constrain the state vector to avoid gimbal lock, but it still occurred throughout the optimization. We give two reasons why this could have happened in the *Discussion* section. To avoid this error, we switched to a MultibodyPlant representation that uses quaternions instead of Euler angles in the dynamics function.

2) MultibodyPlant Quadrotor: Drake's MultibodyPlant Quadrotor [3] representation has the flexibility of using either quaternions or Euler angles for a floating base. When first implemented this, we assumed it automatically used quaternions, but the code snippet we inherited still used Euler angles. Despite this, the optimizers no longer experienced gimbal lock. When used in direct shooting, however, several states experienced overflows, causing the optimizer to fail. We then switched to the quaternion based MultibodyPlant, but this did not resolve the overflows. Moreover, our implementations for direct transcription and direct collocation stopped working. Due to these complications, we decided to revert back to the Euler based MultibodyPlant. To fix direct shooting, we resorted to changing the input to the optimizer. Instead of optimizing over propeller inputs, we optimized over generalized forces.

E. Iterations: Constraint and Cost Engineering

The specific constraints and costs we added for each optimization also changed throughout the course of the project. Most of these changes came from learning about better ways to formulate constraints and choosing costs with behaviors more suited for our environment.

1) Constraints: One of the initial issues we ran into with constraints resulted from taking norms of constraints. For example, in choosing the subgoal for finite lookahead, the horizon radius constraint took the form of $||x|| \leq c$, but this led to slight numerical errors resulting in taking square roots of negative numbers. Instead, the constraint was reformulated as $x^T x \leq c^2$, which avoids the square root entirely.

2) Costs: The main cost that underwent significant changes was the tree cost for direct shooting. Our original cost function consisted of a cubic cost of the following form:

$$dist = ||end[0:2] - T.position||$$

$$L_{tree} = \sum_{T \in F} -(dist - T.radius)^3$$

This cost function had the unintended consequence of creating an unbounded cost that can grow towards negative infinity by having the quadrotor fly away from the forest, ignoring the goal entirely. A better cost would be one that incurs a high penalty inside the tree and zero cost outside of the tree. A steep sigmoid function captures this perfectly, leading to the following cost function:

$$dist = end[0:2] - s_{g,p}$$

$$L_{tree} = \sum_{T \in F} \frac{c}{k + e^{dist}}$$

The equation for a sigmoid function involves exponentiating the distance. Unfortunately, Drake does not allow AutoDiff types as the exponent, so this cost did not work. Another function with similar behavior is a steep arctangent function. After tuning the steepness and cost of crossing a tree boundary, we converged on a cost of the following form:

$$dist(T) = ||s[n][0:2] - T.position||$$

$$L_{tree} = \sum_{T \in F} c(-\arctan k(dist(T) - T.radius) + \frac{\pi}{2})$$

VI. Results

Fig. 3 shows a sample set of results for the 6 algorithms we considered. For consistency, these results were generated by setting a random seed (np.random.seed(2)) so the forests were the same for all runs. The algorithms were tested on forests with increasing tree density: 30, 60, 90, and 120 trees (going left to right in the figure). Figs. 4 through 8 are representations of the z position of the quadrotor flying through the 90 tree example using infinite and finite lookahead methods.

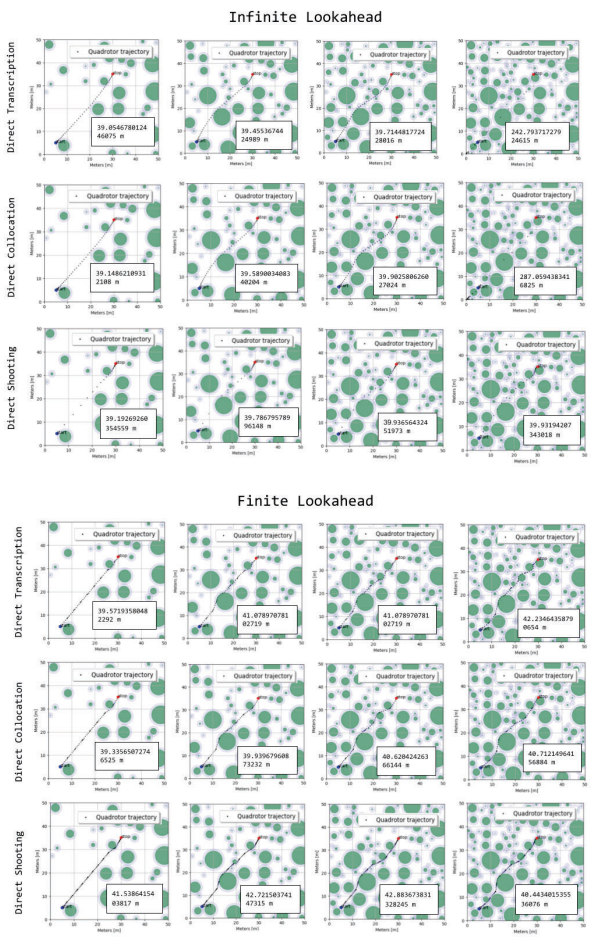


Fig. 3. 6 optimization algorithms tested on randomly generated forests with 30,60,90,and 120 trees (left to right)

A. Infinite vs Finite Lookahead

When comparing finite and infinite lookahead we considered 2 metrics: completion and path length. Completion is dependent on the solver's ability to produce a viable trajectory through the environment from start to end location. In addition, we also considered path length as a comparison metric, with the prediction that the infinite lookahead case would result in lower path lengths since it is assumed that if the solver is given more information, it is able to obtain a more optimal solution.

That said, due to the coastline paradox, the path lengths of the infinite and finite lookahead cases may not be comparable due to the number and spacing of time steps taken – the finite lookahead cases have more total time stamps (more "dots" in Fig. 3) than their infinite lookahead counterparts. According to the coastline paradox, increasing the resolution of path length measurements leads to a peculiar phenomenon where paths appear to grow longer, much like a fractal. Thus, we consider path lengths as a comparison metric only within each lookahead case, i.e. we won't compare the numerical path lengths of infinite lookahead direct transcription and finite lookahead direct transcription, but we will consider path length when comparing direct transcription, direct collocation, and direct shooting in the infinite case and similarly comparing the three methods in the finite case.

We see that algorithms that use finite lookahead are more robust than the algorithms that use infinite lookahead. For the example in Fig. 3, we see the algorithms that use infinite lookahead fail to produce a viable trajectory once the forest reaches 120 trees, while the algorithms that use finite lookahead are able to produce viable trajectories. Since finite lookahead only allows the optimization to consider what is within a certain radius of the quadrotor, the paths are less smooth and are marginally larger across the board, with the largest difference being just over 2 meters – this is only 5 percent longer than the shortest path produced in the infinite case. Regarding path length, we mentioned earlier that the path length comparison might not be the most robust tool in analyzing optimization methods across lookahead distance, so it is much more important to consider the production of a viable trajectory when considering the robustness of the finite vs infinite lookahead optimization.

If we look at the z positions of the finite and infinite lookahead methods, we see a very intuitive pattern. In the finite lookahead case, the goal point of the quadrotor is defined in x and y, with a z of 0. So every time the quadrotor reaches the goal point, it must land in order to re-plan and then take flight again. This results in the "bumps" seen in the trajectories in the finite lookahead case. Meanwhile, the infinite lookahead case only plans once.

B. Direct Shooting

When looking at the trajectories in Fig. 3, it seems that direct shooting is successfully getting to the goal. Unfortunately, this is not exactly the case. Looking at the final state of the finite lookahead direct shooting algorithm, we see that the final x, y, and z positions are 29.9912555, 34.9840112, and -0.0785253152 respectively. When implementing our algorithm, we deemed this error to be within an acceptable delta from the goal. This differs from direct transcription and direct collocation where the algorithms were able to reach the goal perfectly. This is likely due to the fact that for direct shooting, the implementation did not have an end goal constraint, but rather a cost.

Additionally, the graph in Fig. 7 demonstrates a competing effect between the various elements in the cost function.

Namely, in order to minimize the goal cost and avoid trees, the quadrotor needs to increase its actuation cost. But because the actuation cost is penalized, with inadequate cost balancing, the quadrotor constantly hovers under the target height and sometimes clips trees. Moreover, the same cost function was not robust across varying forest densities: the relative balance between the actuation cost and the tree cost needed to be re-tuned at higher forest densities to ensure the quadrotor did not clip through trees in order to minimize its position cost. In the other optimization algorithms, this was not an issue because the trees were formulated as constraints, not costs. When constraints become costs, they are effectively "soft constraints" and the optimization is no longer guaranteed to avoid them.

Overall, direct shooting is not comparable to direct collocation and direct transcription. As mentioned earlier in the paper, the actuation inputs for direct shooting are generalized forces rather than propellers, which makes any comparisons between direct shooting and direct collocation and direct transcription confusing due to two different methodologies. The same can be said for overall trajectory paths. Therefore, the rest of the results section will do a deeper dive into direct transcription and direct collocation.

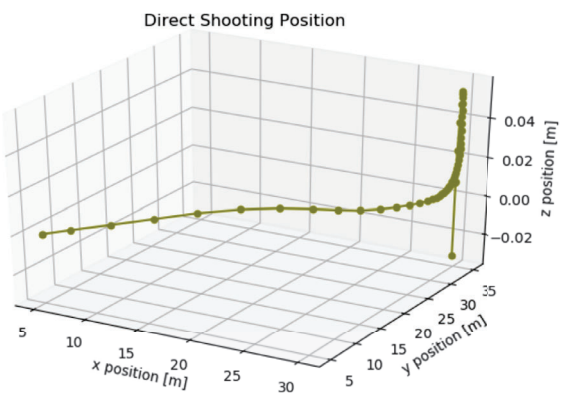


Fig. 4. z position of the quadrotor flying through the 60 tree example above using infinite lookahead direct shooting

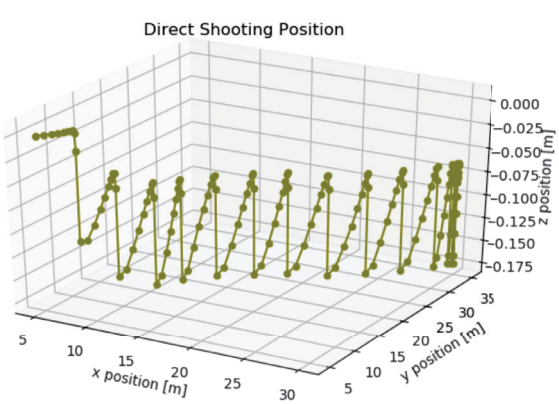


Fig. 5. z position of the quadrotor flying through the 60 tree example above using finite lookahead direct shooting

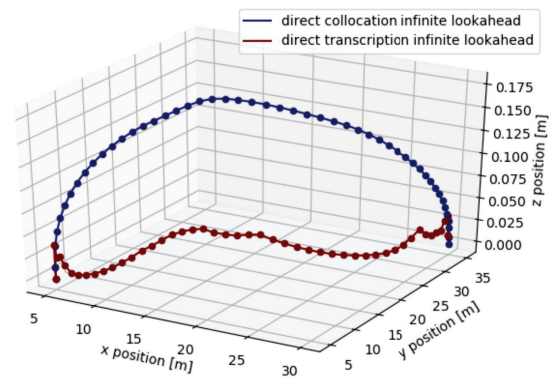


Fig. 6. z position of the quadrotor flying through the 90 tree example above using infinite lookahead methods

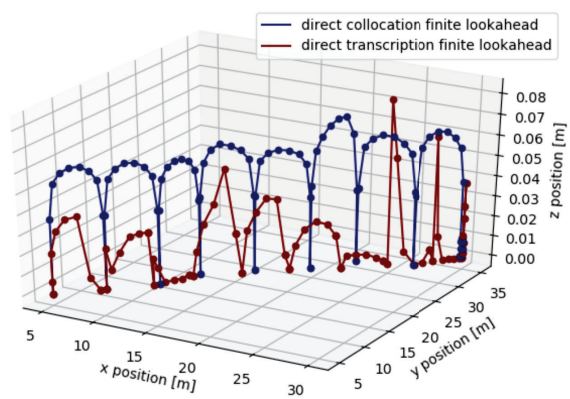


Fig. 7. z position of the quadrotor flying through the 90 tree example above using finite lookahead methods

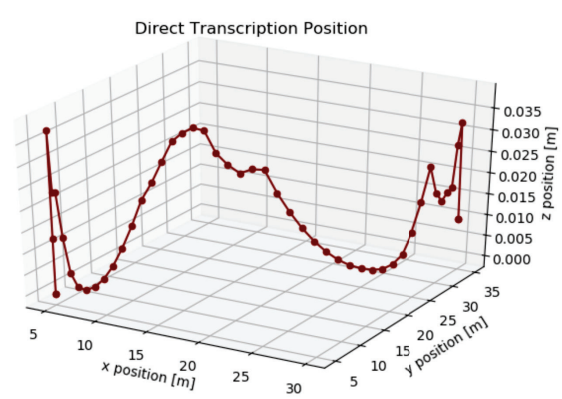


Fig. 8. z position of the quadrotor flying through the 90 tree example above using infinite lookahead direct transciption

C. Direct Transcription and Direct Collocation

Both direct transcription and direct collocation did well until 120 trees, where the infinite look-ahead cases failed. Between direct transcription and direct collocation, we noted direct collocation usually ran a little faster, however, we have not yet taken quantitative measurements. In all finite lookahead cases, direct transcription produced a slightly longer path length than direct collocation; this contrasts the infinite lookahead case, in which direct collocation generates longer path lengths than direct transcription.

An interesting thing to note is the z position of the quadrotor in the infinite lookahead cases. We see that direct transcription remains a lot closer to 0 in the z position, but also has a lot more fluctuations as can be seen in Fig. 8, whereas direct collocation is significantly smoother. The cost function for direct collocation and direct transcription does not cause the program to accrue cost for deviation from the goal at each time step. The only cost is the actuation cost. This may explain the variability in z positions between the two methods, as the actuation cost may have been spent on maneuvers in other axes.

1) Actuation Inputs: We see differences in trajectory patterns reflected in the command inputs for the respective method. In Fig. 9, we see that the command inputs seem to have a steady state value of about 2 and occasionally drop to -2 for brief intervals. Although it is difficult to draw an exact one-to-one correspondence between these spikes and the position graphs seen in Fig. 3 and Fig. 8, the number of spikes roughly corresponds to the number of times the z position deviates from a smooth parabolic curve.

Meanwhile, we see that infinite lookahead direct collocation has a significantly smoother actuation input graph. Furthermore, there is never negative actuation, meaning direct collocation largely relies on the quadrotor's natural dynamics for the latter half of its parabolic arc. We can see that there are significant spikes in actuation at both the beginning and end of the trajectory. The spikes at the very beginning are likely from the quadrotor taking off and setting the initial direction it is flying in. Similarly, the spikes at the end are likely to ensure the quadrotor perfectly arrives at the goal position. Finally, the overall parabolic shape of the actuation input graph mirrors that of the position graph we can see in Fig. 6; as the change in the z position of the trajectory gets less steep, we are using less actuation. The two bumps in the middle of the command inputs are likely due to the particularly tight maneuvers the quadrotor conducts as it hugs two trees along its trajectory. This hypothesis is supported by looking at the path in Fig. 3 and noting that there are two inflection points in the middle of the trajectory for infinite lookahead direct collocation. Furthermore, when flying through a forest with no trees, we do not see these inflection points as demonstrated in Fig. 11. Both these observations imply that the two "bumps" in the actuation input come from obstacle avoidance. Even so, direct collocation still manages to maintain a smoother trajectory than direct transcription.

Looking at the actuation inputs for the finite lookahead cases, we see an interesting difference between the direct transcription and direct collocation cases. In the direct collocation case, the actuation inputs between each subgoal are relatively consistent. This lines up with the consistency of the segments of the trajectory in Fig. 7. Furthermore, looking at the trajectory qualitatively in Fig. 3 (third column), we see the direct collocation only starts "hugging" trees after approximately a quarter of the way through. We see the actuation inputs increase in magnitude and also seemingly in complexity as time goes on, which lines up with our observations. In Fig. 3, we also see that direct transcription "hugs" the contours of the trees longer and the trajectory has more "corners" than the trajectory for direct collocation. This may be part of the reason why the trajectory and command inputs are significantly less consistent from subgoal segment to subgoal segment.

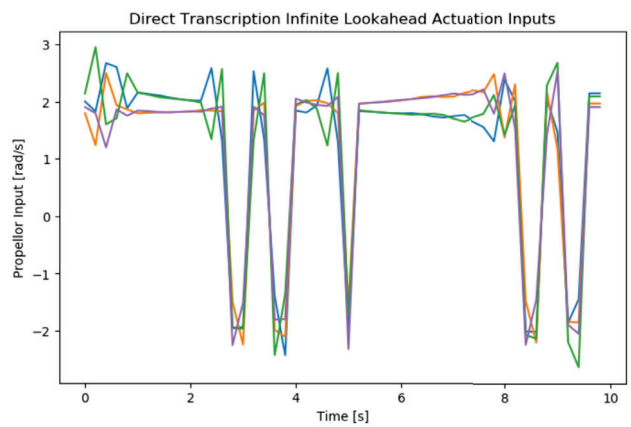


Fig. 9. command inputs for navigating through example forest of 90 trees using infinite lookahead direct transcription

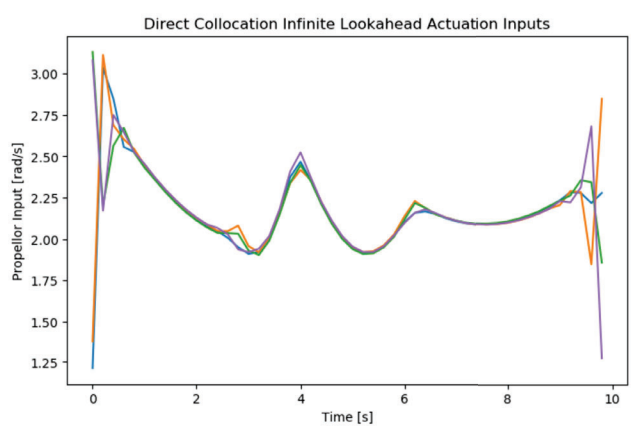


Fig. 10. command inputs for navigating through example forest of 90 trees using infinite lookahead direct collocation

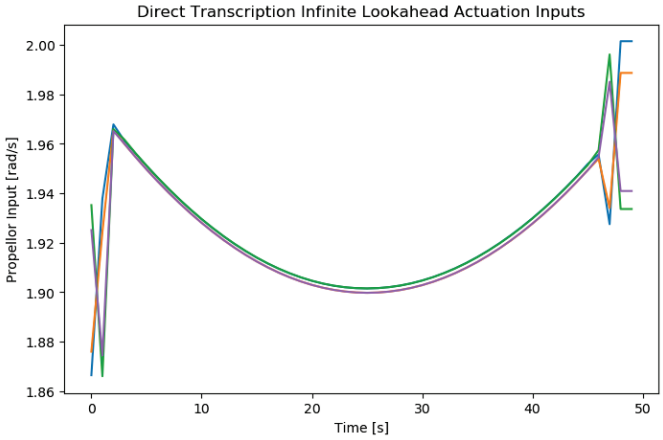


Fig. 11. z position of the quadrotor flying through a forest with no trees using finite lookahead direct collocation

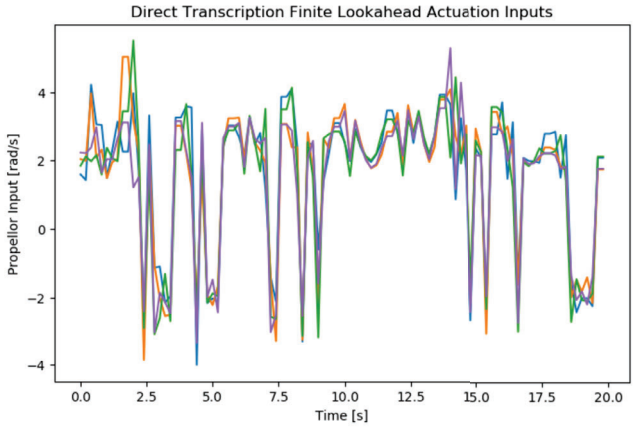


Fig. 12. command inputs for navigating through example forest of 90 trees using finite lookahead direct transcription

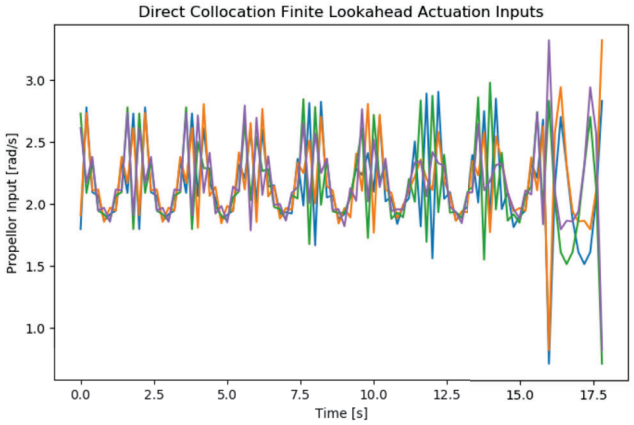


Fig. 13. command inputs for navigating through example forest of 90 trees using finite lookahead direct collocation

VII. Discussion

We determined that direct collocation is the best-performing algorithm for our particular use case, followed by direct transcription, and finally direct shooting. In all cases, the finite look-ahead formulation remained both more robust and ran faster than their infinite lookahead counterparts. We see that direct collocation has smoother command inputs, while still providing trajectories with low path lengths and successfully avoiding trees. We attribute the success of direct collocation to the mathematics behind the optimization. Since direct collocation represents inputs as piece-wise polynomial functions, it is less likely to result in trajectories with irregular shapes such as those from direct transcription. We were successfully able to compare the three optimization methods we implemented to determine the one that performed best in navigating a forest. We were also able to draw relations between command inputs and trajectories, although we believe that a finer grain analysis could provide more insight.

One important point to touch on is the lack of generalizability of our solution. Collocation worked well for our problem as most of the trajectories were able to be represented as splines. For example, there were no ”tight corners” that the quadrotor needed to turn in order to navigate the forest we generated. We speculate that we would arrive at significantly different results if our task was navigating a dense city or a maze. In that case, perhaps the sharp actuation of direct transcription would allow for more precise navigation through geometries that are difficult to capture using polynomial representations.

Most of these algorithms are incredibly expensive, which raises the question of how feasible it is to run the algorithms on hardware. The infinite lookahead case has the benefit of knowing the entire environment, allowing the trajectory to be pre-computed. Conversely, the finite lookahead case needs a horizon small enough such that an onboard computer can quickly generate partial trajectories. The quadrotor also gets very close to some of the trees along the generated trajectory. In real life, we'd want to enforce a safety margin. This can be accomplished by adding a *buffer_radius* to keep the quadrotor a minimum distance from each tree.

VIII. Next Steps

A. Non-convex Forests

The environment representation allows us to find solution trajectories that are not subject to local minima, however, it does not fully encapsulate a realistic forest environment. For example, in a real forest environment, tree canopies may overlap. We would represent this in simulation by allowing trees to overlap such as in Fig. 14. However, this would create non-convex obstacles. The non-convexities of these obstacles would cause the trajectory optimization methods we are using to potentially fail due to local minima. In these cases, the solver would get stuck in the concavity between two objects. The repeated failure of the solver in certain simulated environments is what ultimately led us to switch to the convex forest representation.

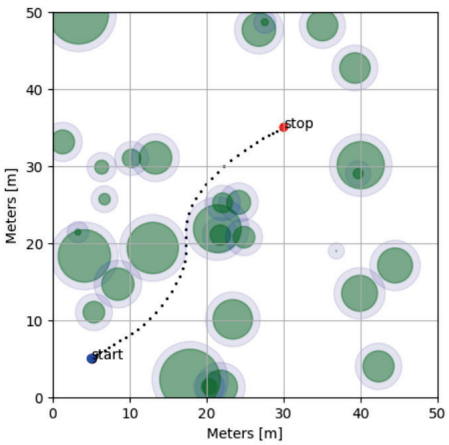


Fig. 14. Non-Convex Forest Environment

B. 3D Forest Representation

A real forest environment contains trees that are not uniform in 3D space; in our current representation, trees exist as infinite cylinders centered around a randomly initialized location with radius *tree_radius*. We initially experimented with a 3D environment provided in a GCS Deepnote notebook from the underactuated textbook [2], seen in Fig. 15; however, this representation did not easily lend itself towards integration with the optimization methods we wanted to compare, so we switched to the forest representation explained in the *Environment* section. In the future, we can look into a more complex tree representation that could allow the quadrotor to fly under a tree's foliage instead of avoiding the tree altogether.

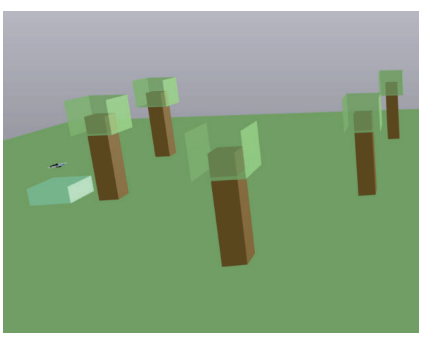


Fig. 15. 3D Sparse Forest Environment

C. Quadrotor Representation

As stated in *Approach* section, limitations regarding the Quadrotor Representation are as follows. We attempted to use quaternions for our floating base in an attempt to prevent gimbal lock. We believe this did not work for one of two reasons. First, the constraints we imposed only ensured that the returned solution was within specified parameters, but the optimizer still explored violating solutions while performing gradient descent. Therefore, despite enforcing these constraints, gimbal lock still occurred. Second, the state at each time stamp obeyed the constraints, but in intermediate states, the dynamics would result in a gimbal lock.

D. Using Quaternions with Different Solvers

In general, quaternions are a more robust way to express the dynamics of a system since it avoids singularities. Solvers like SNOPT however, tend to do very poorly with unit quaternion constraints, as mentioned by the course staff. A potential solution would be to try different solvers and compare their performance in handling quaternions and unit quaternion constraints.

E. Midair Planning for Finite Lookahead

As mentioned in the *Infinite vs Finite Lookahead* section of this paper, the done lands at every subgoal, creating a "bouncing" motion as the quadrotor is on its way to the location of the goal. In the future, we can look into planning midair such that the quadrotor does not have to land in between planning sessions. This may be as simple as constraining the state vector of the start position of the next subgoal trajectory optimization to match only x and y of the current state vector of the quadrotor (the state vector of the quadrotor after reaching a subgoal).

F. Direct Shooting

In this paper, we implemented direct shooting using generalized forces rather than propeller inputs. In future iterations, we would like to determine why the overflow errors occurred and how to properly interface direct shooting with propeller inputs. This will allow for a more proper comparison between direct shooting and the other optimization algorithms.

G. GCS

As discussed in the *Related Works* section, we could look into using GCS as a trajectory optimization method. GCS would lend itself nicely to this environment, and we see it being particularly successful compared to the other trajectory optimization methods, especially when considering a more realistic non-convex forest environment. GCS is able to handle non-convex objects because the solution trajectory considers continuous decision variables constrained in convex free sets. Since the resulting search space is convex, a solution will be found.

H. Parameter Tuning

We recognize that we were running our experiments on a very limited set of parameters. For example, all infinite lookahead experiments were run on $N = 50$ time stamps with a time step of 0.2 seconds. Both these parameters could be tuned to arrive at more optimal trajectories. We were limited by factors such as increasing N leading to an increase in runtime, along with the fact that tuning parameters by hand is a time consuming process. In order to address this, we could include these parameters as decision variables in our optimization methods. This would require significant refactoring of our code but would yield better and more generalizable results.

IX. Contributions

A. Frank Gonzalez

Implemented the framework for running multiple optimizations to emulate a finite lookahead optimization and the mathematical program for selecting subgoals.

B. Shreya Gupta

Implemented the initial mathematical program with constraints and costs for navigation around trees for direct transcription and direct collocation, collected data, and generated plots.

C. Kaitlin Zareno

Implemented the initial framework for the direct shooting formulation, organized and facilitated group meetings, and significantly influenced the direction and structure of the final paper.

D. All

Ideated and conceptualized the project; built the environment and set up the quadrotor dynamics; implemented and debugged direct transcription, direct shooting, and direct collocation; and created the presentation and paper.

Described metaphorically, we all contributed ”seeds” to the project that eventually were contributed to by everyone. Looking at our code and the rest of our work, it is best described as ”one giant shared custody”.

Acknowledgment

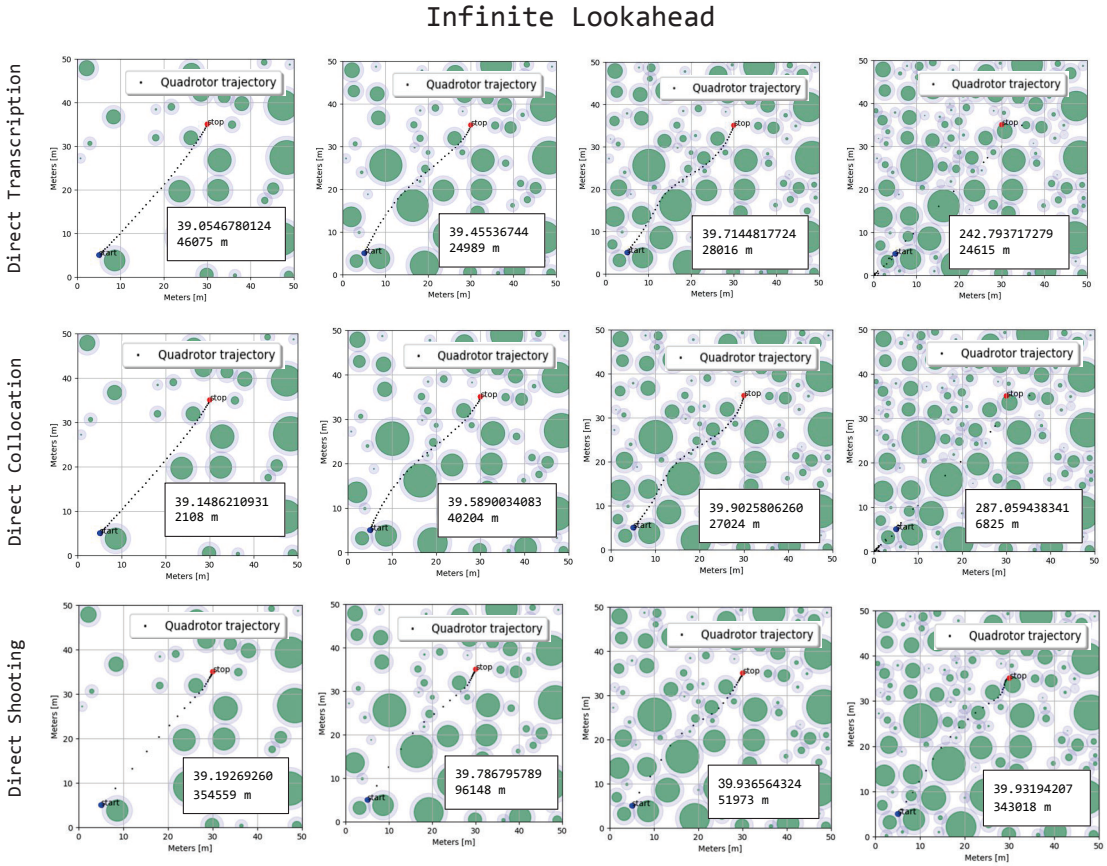
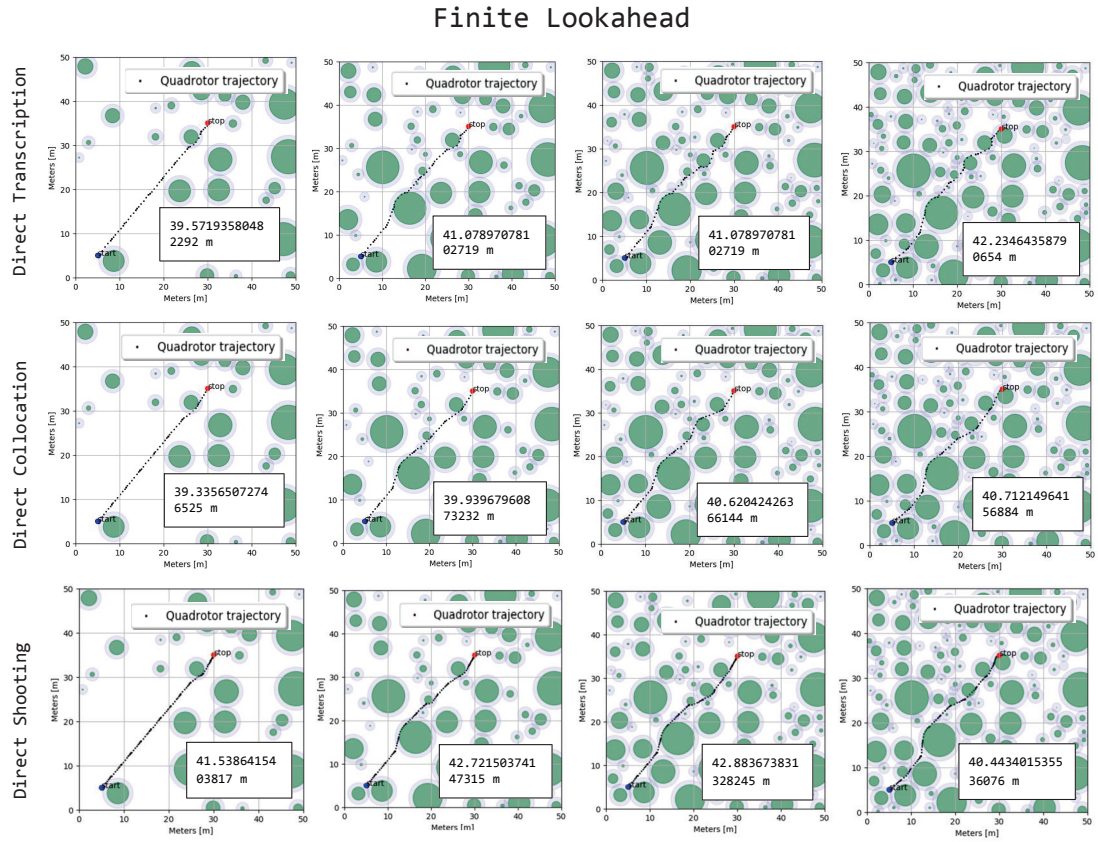
We’d like to thank the course staff and everyone who has put in time into making Drake what it is today. More specifically, we would like to thank Professor Tedrake and Sunbochen Tang for helping us through this project via many (MANY) piazza posts, emails, and in-person discussions.

References

[1] Bry, A., Richter, C., Bachrach, A. & Roy, N. Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *The International Journal Of Robotics Research*. **34**, 969-1002 (2015)

[2] Tedrake, R. Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation. (Course Notes for MIT 6.832,2023), https://underactuated.csail.mit.edu

[3] Tedrake, R. & Drake Development Team Drake: Model-based design and verification for robotics. (2019), https://drake.mit.edu

Appendix

A. Performance of Various Optimization Algorithms on Forests of 30, 60, 90, and 120 Trees