Efficient Quadrotor Pathfinding and Control in Firefighting Applications with RRT* and LPV-MPC

Cedric Hollande
UPenn MEAM
Philadelphia, PA
cdrich@seas.upenn.edu

Nicholas Cannistraci
UPenn MEAM
Philadelphia, PA
ncanni@seas.upenn.edu

***Abstract*—The increasing deployment of unmanned aerial vehicles (UAVs), particularly quadrotors, in critical applications highlights their potential in addressing complex challenges in dynamic environments. This paper presents an approach to trajectory optimization and control for a quadcopter engaged in firefighting operations within a densely forested area. Utilizing the Rapidly-exploring Random Tree Star (RRT*) algorithm, we develop a path planning strategy that navigates around natural obstacles such as trees and branches to a designated fire zone. Coupled with Model Predictive Control (MPC), our method effectively manages the non-linear dynamics of the quadcopter, allowing the quadrotor to accurately follow the 3D path.**

***Index Terms*—quadrotor, RRT*, Model Predictive Control, path planning, trajectory optimization**

I. Introduction

In recent years, the application of quadrotors in emergency response scenarios has gained significant attention due to their agility and the precision with which they can be operated in complex environments. The problem of using MPC to control a quadrotor in 3D also ties into one of our senior design projects consisting of a drone application to assist emergency rescue situations in rural areas (navigating in a forest/in obstacle dense environments). In this project, we designed a controller for a quadcopter efficiently navigating through a 3D space cluttered with obstacles, extinguishing fires in forested terrains. Once the path planning and 3D control is obtained, we will attempt to manage the changing dynamics of the quadrotor due to decreasing water mass from the extinguishing process.

II. Environment Creation and Path Planning

A. Forest Generation

The forest is represented by cylindrical obstacles (trees) systematically placed across a defined 3D space. Each tree's position, radius, and height are randomly determined within specified ranges to ensure variability, while also maintaining a minimum distance from the designated start position to ensure accessibility for the initial deployment of the quadcopter. The fire zone is generated as a 2D bounding spline where all trees within the spline are designated as 'on fire'. This was necessary for determining the optimal path to extinguish the fire. Figure 1 is a matplotlib representation of this forest.

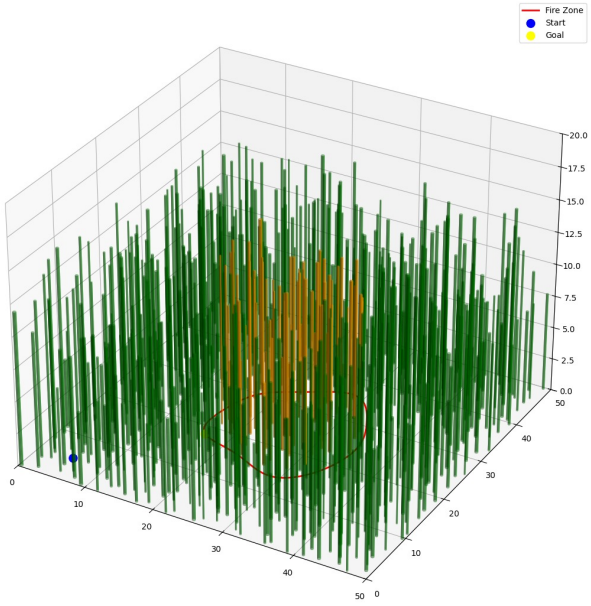

Fig. 1. Randomly Generated 3D Forest map.

B. RRT* Path Planning

We employ the Rapidly-exploring Random Tree Star (RRT*) algorithm, an advanced variant of the widely recognized Rapidly-exploring Random Tree (RRT), to navigate and create a path through the 3D forest we had generated. RRT* refines the approach of its predecessor by incorporating a rewiring step, which consistently seeks to minimize path costs, thus yielding more optimal routes over time. The algorithm operates by incrementally building a tree from the start position, extending towards randomly sampled points that bias toward the goal to effectively steer the exploration. For each new point, the nearest existing node is connected by a new node if it does not intersect with any obstacles. This new node potentially becomes a pivot for rewiring nearby nodes, ensuring that paths in the developing tree are the shortest possible. Each successful extension includes a check to determine if the new node enables a feasible, shorter path to the goal. After building the tree, the algorithm refines the resulting path by segmenting it and adjusting each segment

to maintain a safe distance from nearby obstacles, ensuring the path is not only direct but also traversable. A 2D birds-eye view of the results of using RRT* to create this path are shown in Figure 2.

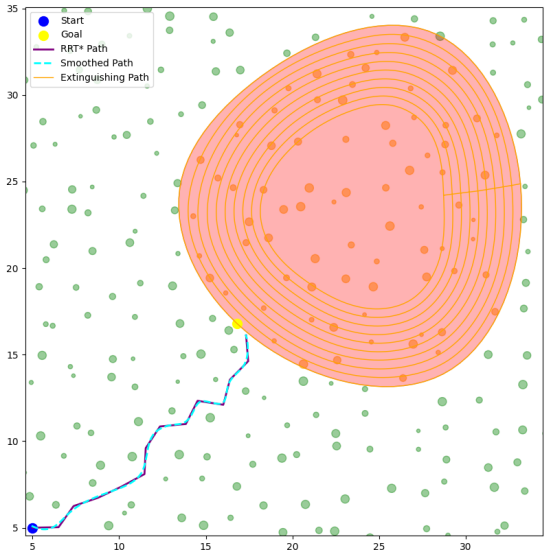

Fig. 2. Fast and optimal RRT* pathfinding in obstacle dense forest.

III. System Control

A. Dynamics Modeling

Referencing "Different linearization control techniques for a quadrotor system" [1] we modeled the full dynamics of the quadrotor using continuous time equations. Figure 4 shows the fully defined quadrotor system. In this model, we first calculate the state derivatives using the current states and control inputs shown below.

$$
\begin{cases}
\dot{x_1} = x_2 \\
\dot{x_2} = \frac{1}{m}\left(\cos\phi_1 \sin\theta_1 + \sin\phi_1 \sin\psi_1\right) U_1 \\
\dot{y_1} = y_2 \\
\dot{y_2} = \frac{1}{m}\left(\cos\phi_1 \sin\theta_1 \sin\psi_1 - \sin\phi_1 \cos\psi_1\right) U_1 \\
\dot{z_1} = z_2 \\
\dot{z_2} = -g + \frac{1}{m} cos\phi_1 cos\theta_1 U_1 \\
\dot{\phi_1} = \phi_2 \\
\dot{\phi_2} = \frac{I_y - I_z}{I_x}\theta_2\psi_2 + \frac{J_{TP}}{I_y}\theta_2\Omega + \frac{1}{I_x}U_2 \\
\dot{\theta_1} = \theta_2 \\
\dot{\theta_2} = \frac{I_z - I_x}{I_y}\phi_2\psi_2 + \frac{J_{TP}}{I_y}\phi_2\Omega + \frac{1}{I_y}U_3 \\
\dot{\psi_1} = \psi_2 \\
\dot{\psi_2} = \frac{I_x - I_y}{I_z}\phi_2\theta_2 + \frac{1}{I_z}U_4
\end{cases}
$$

Fig. 3. Quadrotor dynamics equations of motion.

Then, using the calculated velocities and angular velocities, transformed using a rotation and translation matrix, we compute the position derivatives and derivatives of the Euler angles (the nonlinear dynamics). Finally, we find the control forces and control torques which represent how changes in rotor speeds influence the thrust and torques, and are critical for controlling the drone's motion.

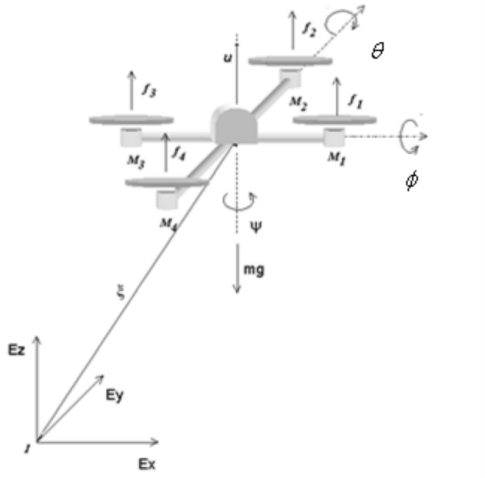

Fig. 4. Quadrotor dynamics force diagram.

B. MPC

After developing our model of the quadrotor systems dynamics, we implemented a Model Predictive Control (MPC) framework.

The core of the implementation resides within a defined class, which orchestrates the MPC process. The controller operates by first establishing a time discretization: generating a time vector that spans the simulation duration based on the sampling time and the number of inner loop iterations. This discretization is crucial for synchronizing the control updates with the system dynamics.

The MPC controller initiates by extracting the reference trajectories for the quadrotor's spatial coordinates (X, Y, Z) and orientation (yaw angle, psi) from the given trajectory function. The controller initializes the quadrotor's state, including velocities, angular rates, and rotor speeds, ensuring the system begins from a known configuration.

The control process is divided into two nested loops: an outer loop corresponding to the main control updates and an inner loop handling the finer dynamics. In each iteration of the outer loop, the controller employs feedback linearization to compute the desired roll and pitch angles, aligning the quadrotor's orientation with the reference trajectory. These reference angles are interpolated over the inner loop's horizon to provide a smooth trajectory for the quadrotor to follow.

Within the inner loop, the system dynamics are modeled using a Linear Parameter-Varying (LPV) approach, which

linearizes the nonlinear quadrotor dynamics around the current state. The LPV generates the necessary state-space matrices (Ad, Bd, Cd) for the current operating point. The MPC optimization problem is then formulated as a quadratic program (QP) using Drake's MathematicalProgram and solved with the pydrake QP solver. The objective is to minimize the deviation from the reference trajectory while adhering to the quadrotor's dynamic constraints and the following cost function [2] is used.

$$J'' = \frac{1}{2}\Delta u^T(\overline{\overline{C}}^T\overline{\overline{QC}} + \overline{\overline{R}})\Delta u + \begin{bmatrix}\tilde{x}_t^T & r^T\end{bmatrix}\begin{bmatrix}\widehat{A}^T\overline{\overline{QC}} \\ -\overline{\overline{TC}}\end{bmatrix}\Delta u =$$

$$= \frac{1}{2}\Delta u^T\overline{\overline{H}}\Delta u + \begin{bmatrix}\tilde{x}_t^T & r^T\end{bmatrix}\overline{\overline{F}}^T\Delta u = \mathbf{\frac{1}{2}\Delta u^T\overline{\overline{H}}\Delta u + f^T\Delta u}$$

Fig. 5. Final Costs used in the LPV-MPC.

Upon solving the QP, the optimal control increments (ΔU) are applied to update the quadrotor's control inputs (U2, U3, U4), corresponding to thrust and torque commands. These updated inputs are converted to rotor angular velocities. The system's state is subsequently integrated over the timestep, incorporating multiple sub-steps to enhance numerical stability and accuracy.

C. MPC Testing and Iteration

For our poster day, we managed to get the MPC somewhat working but it still exhibited significant inaccuracies despite positive results with synchronization of the control inputs. Figure 6 shows the initial attempts at following a 3D path using the aforementioned dynamics and MPC .

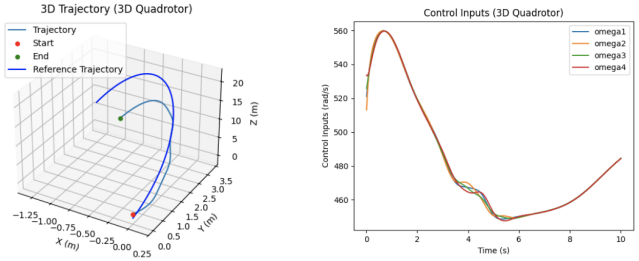

Fig. 6. First MPC implementation high error.

After making adjustments to our MPC and switching to an LPV linearization, we were able to get the path following to be much more accurate. From there, we tuned . We found that there was a sweet spot for accuracy in the amount of steps that we used in the integrator. Figures 7-8 show the inaccuracies at 30 integral steps and how it becomes way more accurate at 80 integral steps. We also noticed that it would start to deviate slightly when we increased the integral steps all the way to 100 integral steps.

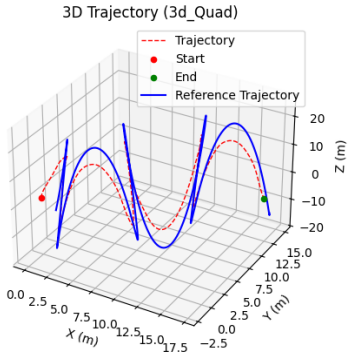

Fig. 7. LPV-MPC with 30 Integral steps - Low Precision

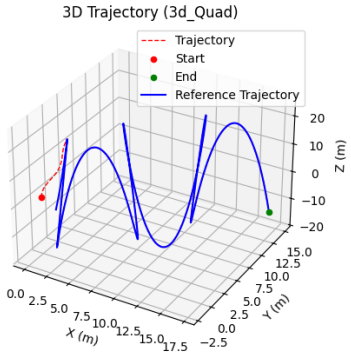

Fig. 8. LPV-MPC with 80 Integral steps - High Precision

IV. Final Results

Using the same MPC that worked accurately in Figure 8 we finally applied it to the path that we developed through the trees using RRT*. These results are displayed in Figure 9 and can be observed more clearly in the following video:(link). One thing to note about this video is that the reference path that the drone follows slightly deviates from the reference path generated from the RRT*. This was due to a slight issue in the discretization of the RRT* path and download time of the video prevented us from running the code again. However, we can still see that the drone follows the given reference path through the trees.

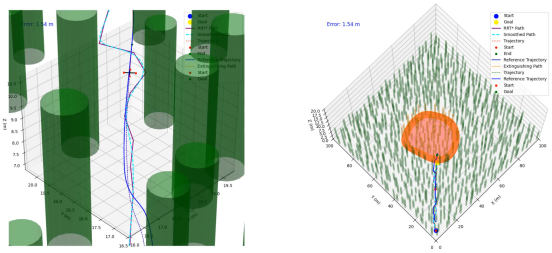

Fig. 9. Quadrotor navigating to the Fire Zone in a dense Forest, following a generated RRT* collision-free path, with a LPV-MPC / linear feedback controller

V. Conclusion and Future Works

We were successfully able to generate a complex 3D field of obstacles and control the movement of a quadrotor through this field using model predictive control. To make this slightly more applicable in the future, instead of using a randomly generated map of trees, we could use geothermal imaging of actual fires and apply our RRT* algorithm to real-life scenarios. Although this would be slightly more difficult from a planning perspective, the controls algorithm that we developed would be easily applicable.

Another aspect that we failed to account for was the varying mass of the quadrotor as it dispenses water into the fire. In order to account for this in the future, we would set a variable representing the flow rate of the water. Using this, we could easily track the mass of the quadrotor over time and adjust the dynamics accordingly.

References

[1] Belkheiri, M.; Rabhi, A.; Hajjaji, A.E.L.; Pégard, C.P. Different Linearization Control Techniques for a Quadrotor System. In Proceedings of the CCCA12, Marseilles, France, 6–8 December 2012; pp. 1–6.

[2] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.