Model Predictive Control-based Trajectory Planning for Quadrotors*

Mintae Kim[1], Qiayuan Liao[1], Faris Tulbah[1], Ruiting Wang[1]

***Abstract*— In this paper, we aim to address the challenge of trajectory planning for autonomous quadrotors. This challenge emerges due to the highly nonlinear dynamics of the system and the complicated environment in which it is operated. To tackle this control problem, we will explore two vastly different approaches - one utilizing traditional trajectory optimization tools and the other utilizing a deep learning framework. Three transcription methods are implemented for trajectory optimization, the MPC performance of which is evaluated under different receding horizons. Additionally, the study introduces a duality-based collision avoidance strategy using polytope representation for better trajectory planning. We have also implemented a neural network-based controller. This model operates without an explicit representation of the system's dynamics. Instead, it learns to generate control inputs based on observed data, offering a flexible and potentially more generalizable approach to quadrotor control. By learning directly from the system's performance, such a framework can adapt to a variety of conditions and scenarios, even those not explicitly modeled in traditional simulations. This approach highlights the power of deep learning in handling complex control tasks where explicit dynamic modeling is challenging or impractical. Findings suggest that MPC in both types of problems can control the system, either with obstacles or perturbations in the given scenario. Further research and development are required to fully realize and validate the model behavior in real-world scenarios.**

I. Introduction

Trajectory planning for autonomous quadrotors has been studied for a long time, but real-time spatial and temporal optimal trajectory planning in applications remains a challenging task. This challenge arises from the highly nonlinear dynamics of the system, the complicated environment in which it operates, and the limited information we can measure. In this paper, we seek to tackle some challenges associated with trajectory planning for autonomous quadrotors with Model Predictive Control (MPC).

In this study, we first exploit the dynamics of the system and model the problem as a traditional trajectory optimization, to discretize the model, three transcription methods are used - direct transcription, single-shooting, and direct collocation. The study introduces a duality-based collision avoidance strategy using polytope representation to mimic the complicated scenarios the quadrotor may face. We study its performance under different MPC horizons.

However, such a framework depends on the availability of an accurate dynamics model for the underlying system, a condition further exacerbated by stringent real-time constraints. In the second part of the study, we present Neural MPC, a trajectory planning framework for quadrotors that incorporates deep learning principles within the MPC paradigm. By learning directly from the system's performance, such a framework can adapt to a variety of conditions and scenarios, even those not explicitly modeled in traditional simulations. This approach highlights the power of deep learning in handling complex control tasks where explicit dynamic modeling is challenging or impractical.

The main contributions of the work is listed as follows:

1) A classic MPC framework is implemented and evaluated for autonomous quadrotor trajectory planning problems with obstacle avoidance.
2) We proposed a novel neural MPC framework to solve the same problem without explicit system dynamics information while considering perturbations.

II. Problem Formation

A. Quadrotors Dynamics

The system dynamics can be written as follows,

$$\begin{aligned}\ddot{\mathbf{p}} &= \left(\mathbf{R}[0,0,k_F\textstyle\sum_{i=0}^{3}\omega_i^2] - [0,0,mg]\right) m^{-1} \\ \ddot{\boldsymbol{\psi}} &= \mathbf{J}^{-1}\left(\tau(l,k_F,k_T,[\omega_0^2,\omega_1^2,\omega_2^2,\omega_3^2]) - \dot{\boldsymbol{\psi}} \times \left(\mathbf{J}\dot{\boldsymbol{\psi}}\right)\right)\end{aligned} \tag{1}$$

where $\mathbf{p} = [x, y, z]$ corresponds to the drone positions and $\boldsymbol{\psi} = [\phi, \theta, \psi]$ to its angular positions; $\mathbf{R}$ and $\mathbf{J}$ are its rotation and inertial matrices respectively, $\tau(\cdot)$ is a function calculating the torques induced by the motor speeds ω_i, while arm length l, mass m, gravity acceleration constant g along with k_F and k_T are scalar variables describing the quadrotors' physical properties. [4]

The dynamics of the Quadcopter model are central to its functionality and realism. This section is meticulously designed to simulate the complex interactions of physical forces and movements that a quadcopter experiences during flight. Key aspects of the dynamics function in the Quadcopter class include:

1) Force and Torque Calculations: Our model calculates the quadcopter's dynamics by considering forces like gravity, thrust, and aerodynamics, and torques from motors. It computes each motor's thrust and transforms these forces into the world frame based on the quadcopter's orientation, essential for simulating various flight conditions.

2) Orientation Calculation: Euler Angles, Rotation Matrices, and Quaternions: The model accurately represents the quadcopter's orientation, converting between Euler angles, rotation matrices, and quaternions. Euler angles offer intuitive understanding, while matrices and quaternions avoid

*This is a final project paper for MEC231A, Fall 2023 at University of California, Berkeley. Please find the presentation link here.

[1]Mintae Kim, Faris Tulbah, Qiayuan Liao, and Ruiting Wang are with University of California, Berkeley, {mintae.kim, faris.tulbah, qiayuanl, rtwang}@berkeley.edu

gimbal lock and enhance stability, crucial for precise orientation tracking and stabilization.

3) Simulation of External Factors: External factors, like wind gusts, are simulated by introducing random velocity perturbations, adding realism and complexity. This enhances the development of robust control algorithms for realistic flight conditions.

4) Torque Calculation: The dynamic behavior of the quadcopter is governed by several key equations:

The torques can be calculated as follows:

$$\tau_x = (F_1 + F_2 - F_3 - F_4)\frac{L}{\sqrt{2}} \tag{2}$$

$$\tau_y = (-F_1 + F_2 + F_3 - F_4)\frac{L}{\sqrt{2}} \tag{3}$$

$$\tau_z = (-T_1 + T_2 - T_3 + T_4) \tag{4}$$

where F_i are the perturbed forces from each motor.

B. Optimal Control Problem

Generally, an optimal control problem can be formulated as the following optimization problem,

$$\min_{\mathbf{x}(t),\mathbf{u}(t)} \; p(\mathbf{x}(t_0+t_h)) + \int_{t_0}^{t_0+t_h} q(\mathbf{x}(t),\mathbf{u}(t),t)\,dt, \tag{5a}$$

$$\text{s.t. } \mathbf{x}(t_0) = \hat{\mathbf{x}}, \tag{5b}$$

$$\dot{\mathbf{x}}(t) = \mathbf{f}_c(\mathbf{x}(t),\mathbf{u}(t),t), \tag{5c}$$

$$\mathbf{g}(\mathbf{x}(t),\mathbf{u}(t),t) = \mathbf{0}, \tag{5d}$$

$$\mathbf{h}(\mathbf{x}(t),\mathbf{u}(t),t) \geq \mathbf{0}, \tag{5e}$$

where p and q are the terminal cost and stage cost respectively, (5b) is the initial state constraint, (5c) is system dynamics, (5d) is general equality constraints, (5e) is the general inequality constraints.

III. Optimization Implementation and Result

In this chapter, we aim to implement MPC for the trajectory optimization problem, with the system dynamics introduced in the previous chapter.

We first introduce the three transcription methods we used for this problem and the correlated formulation of the problem. Then, we introduce the modeling method of duality-based collision avoidance and its performance. Finally, we show the performance of MPC under different planning horizons and receding horizons. The results show that for MPC, direct collocation has the best performance, so the required prediction horizon can be shorter than the other two transcription methods.

A. Trajectory Optimization

In this section, we assessed three transcription techniques aimed at optimizing trajectories. We then analyzed their performance with the MPC model for varying receding horizons [1].

1) Direct transcription: Direct transcription, as its name suggests, is a method that uses the integral form of the dynamics constraint directly. The control inputs are discrete and are represented as piecewise constant, while the states are also discretized using the zero-order hold method.

2) Single-Shooting: In the single-shooting method, we approximate the trajectory using a single simulation. That is to say, the entire trajectory is modeled from the initial states, and the system states are no longer decision variables in the trajectory optimization problem. The simulation saves the state trajectory. Errors in the simulation grow larger as the state moves further away from its initial point. However, in MPC, these discrepancies in simulation can be eliminated by iteratively solving the problem.

3) Direct collocation: By using first-order and cubic spline to interpolate the $\mathbf{u}(t)$ and $\mathbf{x}(t)$ respectively, we can enforce the dynamics constraints at the so-called collocation point [2]

$$\dot{\mathbf{x}}_{c,k} = f(\mathbf{x}_{c,k}, \mathbf{u}_{c,k}), \tag{6}$$

where $\mathbf{x}_{t_{c,n}}$ is the sample time corresponding to the collocation point, and the collocation points are defined by

$$\mathbf{u}_{c,k} = \frac{1}{2}\left(\mathbf{u}_k + \mathbf{u}_{k+1}\right), \tag{7}$$

$$\mathbf{x}_{c,k} = \frac{1}{2}\left(\mathbf{x}_k + \mathbf{x}_{k+1}\right) + \frac{h}{8}\left(\dot{\mathbf{x}}_k - \dot{\mathbf{x}}_{k+1}\right), \tag{8}$$

$$\dot{\mathbf{x}}_{c,k} = -\frac{3}{2h}\left(\mathbf{x}_k - \mathbf{x}_{k+1}\right) - \frac{1}{4}\left(\dot{\mathbf{x}}_k + \dot{\mathbf{x}}_{k+1}\right). \tag{9}$$

B. Duality-based Collision Avoidance

1) Polytope representation: We can describe the shape of the dynamic robot and the i-th static obstacle as convex polytopes in a l-dimensional space, which are defined using inequality constraints, respectively, as

$$\mathcal{R}(\mathbf{x}) \coloneqq \{\mathbf{y} \in \mathbb{R}^l : A_{\mathcal{R}}(\mathbf{x})\mathbf{y} \leq \mathbf{b}_{\mathcal{R}}(\mathbf{x})\}, \tag{10a}$$

$$\mathcal{O}_i \coloneqq \{\mathbf{y} \in \mathbb{R}^l : A_{\mathcal{O}_i}\mathbf{y} \leq \mathbf{b}_{\mathcal{O}_i}\}, \tag{10b}$$

where $\mathcal{R}$ and $\mathcal{O}_i$ represent the robot and the i-th obstacle respectively, $A_{\mathcal{R}}(\mathbf{x}) \in \mathbb{R}^{s^{\mathcal{R}} \times l}, \mathbf{b}_{\mathcal{R}}(\mathbf{x}) \in \mathbb{R}^{s^{\mathcal{R}}}$ define the robot, and $A_{\mathcal{O}_i} \in \mathbb{R}^{s^{\mathcal{O}_i} \times l}$ and $\mathbf{b}_{\mathcal{O}_i} \in \mathbb{R}^{s^{\mathcal{O}_i}}$ define the i-th obstacle. The symbols $s^{\mathcal{R}}$ and $s^{\mathcal{O}_i}$ represent the number of facets of the polytopic sets for the robot and i-th obstacle, respectively. Note that the pose of the robot polytope $\mathcal{R}$ depends on the system state.

2) Minimum Distance Primal Problem: The square of the minimum distance between $\mathcal{R}(\mathbf{x})$ and $\mathcal{O}_i$, denoted by $d_i^*(\mathbf{x})$, can be computed using a QP as follows:

$$d_i^*(\mathbf{x}) = \min_{\mathbf{y}^{\mathcal{R}}, \mathbf{y}^{\mathcal{O}_i}} \left\|\mathbf{y}^{\mathcal{R}} - \mathbf{y}^{\mathcal{O}_i}\right\|^2, \tag{11a}$$

$$\text{s.t.} \quad A_{\mathcal{R}}(\mathbf{x})\mathbf{y}^{\mathcal{R}} \leq \mathbf{b}_{\mathcal{R}}(\mathbf{x}),\ A_{\mathcal{O}_i}\mathbf{y}^{\mathcal{O}_i} \leq \mathbf{b}_{\mathcal{O}_i}. \tag{11b}$$

We directly want to use this as an inequality constraint, *i.e.*, enforce $d^*(\mathbf{x}) \geq 0$. However, the above optimization problem can only be solved numerically and not analytically. This results in the difficulty of adding the minimum distance function to other optimization problems because it is implicitly defined by (11) and the whole problem will become a bi-level optimization problem.

3) Minimum Distance Duality Problem: We can convert the minimization problem (11) into a maximization problem using the principle of duality [3].

The dual problem of (11) is given by

$$d^*(\mathbf{x}) = \max_{\boldsymbol{\lambda}^{\mathcal{R}}, \boldsymbol{\lambda}^{\mathcal{O}_i}} -(\boldsymbol{\lambda}^{\mathcal{R}})^T \mathbf{b}_{\mathcal{R}}(\mathbf{x}) - (\boldsymbol{\lambda}^{\mathcal{O}_i})^T \mathbf{b}_{\mathcal{O}_i} \tag{12a}$$

$$\text{s.t.} \quad A_{\mathcal{R}}^T(\mathbf{x})\boldsymbol{\lambda}^{\mathcal{R}} + A_{\mathcal{O}_i}^T \boldsymbol{\lambda}^{\mathcal{O}_i} = 0, \tag{12b}$$

$$\left\| A_{\mathcal{O}_i}^T \boldsymbol{\lambda}^{\mathcal{O}_i} \right\|_2 \leq 1, \ \boldsymbol{\lambda}^{\mathcal{R}} \geq 0, \ \boldsymbol{\lambda}^{\mathcal{O}_i} \geq 0, \tag{12c}$$

where $\boldsymbol{\lambda}^{\mathcal{R}}$ and $\boldsymbol{\lambda}^{\mathcal{O}_i}$ are the dual variables corresponding to the primal problem.

4) Constraints Formulation: Once we have the maximum problem, we can enforce $d(\mathbf{x}) \geq \alpha$, where α is the minimum distance margin between the polytopes of the robot and obstacles, and $d(\mathbf{x})$ is any feasible objective value of (12), into any optimal control problem. As a result, we have:

$$d^*(\mathbf{x}) \geq d(\mathbf{x}) \geq \alpha, \tag{13}$$

which means the robot never collides with the obstacle. A full problem formulation example can be:

$$\begin{aligned}
\min_{\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\lambda}_k^{\mathcal{R}}, \boldsymbol{\lambda}_k^{\mathcal{O}_i}} \quad & q(\mathbf{x}_N) + \sum_{k=0}^{N-1} p(\mathbf{x}_k, \mathbf{u}_k) \\
\text{s.t.} \quad & \mathbf{x}_{k+1} = \mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k), \quad \forall k \in [0, N-1], \\
& \mathbf{x}_0 = \mathbf{x}(0), \\
& -(\boldsymbol{\lambda}_k^{\mathcal{R}})^T \mathbf{b}_{\mathcal{R}}(\mathbf{x}) - (\boldsymbol{\lambda}_k^{\mathcal{O}_i})^T \mathbf{b}_{\mathcal{O}_i} \geq \alpha, \\
& A_{\mathcal{R}}^T(\mathbf{x})\boldsymbol{\lambda}_k^{\mathcal{R}} + A_{\mathcal{O}_i}^T \boldsymbol{\lambda}_k^{\mathcal{O}_i} = 0, \\
& \left\| A_{\mathcal{O}_i}^T \boldsymbol{\lambda}_k^{\mathcal{O}_i} \right\|_2 \leq 1, \\
& \boldsymbol{\lambda}_k^{\mathcal{R}} \geq 0, \ \boldsymbol{\lambda}_k^{\mathcal{O}_i} \geq 0, \\
& + \text{additional constraints.}
\end{aligned} \tag{14}$$

5) Result: We formulate a trajectory optimization problem using (14) and the dynamics mentioned above, with the initial state $\mathbf{x}_0 = [0,0,0,0,0,0,0,0,0,0,0,0,0]^T$, $\mathbf{x}_N = [1.0,0,0,0,0,0,0,0,0,0,0,0,0]^T$, and the total time interval is 2 s. The result is shown in Fig. 2. By considering the obstacle's shape by duality formulation the robot can avoid the obstacle.

C. Terminal cost and terminal condition design

The goal of the problem is for the quadrotor to move to a given position and remain in that position. In receding horizon control problem, in order to keep the problem feasible, instead of using a terminal constraint such that,

$$\mathbf{x}_T = \mathbf{x}_{\text{f}} \tag{15}$$

we designed the cost function in such a way that penalizes unwanted behavior,

$$J = \alpha_1 J_{\text{input}} + \alpha_2 J_{\text{state}} + \alpha_3 J_{\text{final}} \tag{16}$$

where,

$$J_{\text{input}} = \|\mathbf{u}\|_2, J_{\text{state}} = \sum_{k=0}^{n-1} \|\mathbf{x}_k - \mathbf{x}_{\text{f}}\|_2, J_{\text{final}} = \|\mathbf{x}_N - \mathbf{x}_{\text{f}}\|_2 \tag{17}$$

By tuning the parameters $(\alpha_1, \alpha_2, \alpha_3)$, we adjust the trade-off between them. Since our goal is to make sure the quadrotor moves to a given position, we set $\alpha_1 = \alpha_2 = 1, \alpha_3 = 100$.

D. Receding Horizon Control

Finally, we created diverse short horizons N for the related MPC problems of the trajectory optimization problems with varying transcription procedures.

Figure 1(a) shows the direct transcription results. With $N = 2$, the model does not converge; with $N = 4$, the quadrotor would overshoot and need more steps to return to its target; with $N = 6$, the model can smoothly control the quadrotor to reach the target point.

Similarly, as shown in Figure 1(b), the single-shooting method requires a horizon of $N = 6$ to effectively guide the quadrotor to its destination. In contrast, with direct collocation 1(c), a horizon of $N = 2$ will cause the quadrotor to overshoot its target before returning, while a horizon of $N = 4$ is sufficient for optimal control.

In conclusion, various trajectory optimization transcription approaches necessitate different minimum MPC receding horizons for the issue to converge, and the direct collocation method has superior controller performance.

IV. Bringing Neural MPC to Trajectory Planning

We have incorporated deep learning, particularly neural network-based controllers, into quadcopter control systems. Deep learning is adept at discerning complex patterns and making informed decisions from extensive data sets. Our study focuses on two PyTorch-based neural network controllers, `BoxConstrainedController` and `RandConstController`, each tailored for specific control system needs, harnessing neural networks' flexibility and computational efficiency.

1) Integration with PyTorch for Batch Processing: Our dynamics function is integrated with PyTorch, enabling efficient batch processing and GPU acceleration for scalable and rapid simulations. PyTorch's tensor operations aid in managing complex calculations and parallel processing, crucial for real-time quadcopter dynamics simulation.

2) Handling of Real-world Perturbations: The quadcopter model is distinctive for its incorporation of real-world perturbations, enhancing practical applicability. It simulates variable payloads and structural changes through mass perturbation factors and accounts for motor performance variations, adding realism. These features equip the model for robust and versatile real-world scenario simulations.

A. Simulation of Perturbations in System Dynamics

1) Overview: Our control model introduces three key perturbations—velocity, mass, and motor efficiency—to test system robustness in real-world-like conditions.

2) Velocity Perturbations: We simulate environmental effects, such as wind, by adding small, random, normally distributed noise to the system's velocity, representing wind's unpredictability.

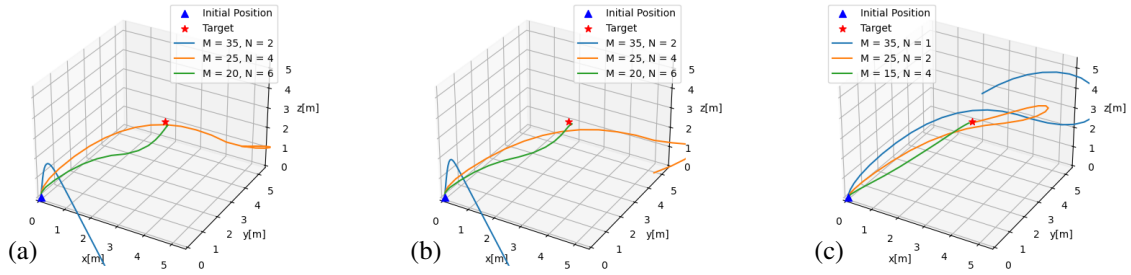

Fig. 1. Trajectory for the quadrotor under different receding horizons using (a) direct transcription, (b) single shooting, (c) direct collocation.

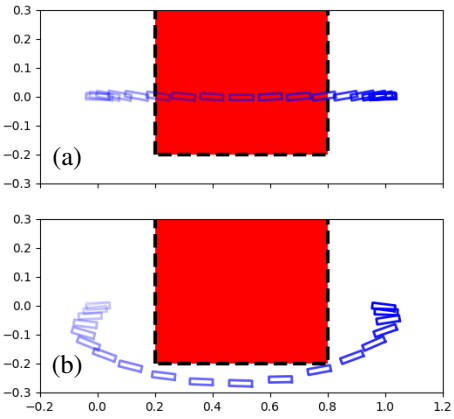

Fig. 2. Snapshots for trajectory optimization (a) without and(b) with Duality-based collision avoidance constraints. The red box is the obstacle, and the blue is the robot. Lighter snapshots are earlier in time.

3) Mass Perturbations: Mass perturbations account for variable system weight due to factors like fuel use or payload changes, adding complexity to the dynamics.

4) Motor Efficiency Fluctuations: Motor efficiency variations, influenced by wear and environmental conditions, are modeled as realistic fluctuations affecting thrust and performance.

5) Combined Impact on the System: Incorporating these perturbations creates a realistic simulation, challenging our control model to maintain stability and revealing the interplay and impact of various factors on system performance.

B. BoxConstrainedController

`BoxConstrainedController` is a neural network tailored for control tasks that require bounded control inputs. Its architecture and functionality are as follows:

1) Network Architecture: The controller's architecture is composed of several key elements:

- **Input Layer:** The network begins with an input layer of dimension `in_dim`.
- **Hidden Layers:** It includes multiple hidden layers, each of dimension `h_dim`. The total number of hidden layers is defined by `num_layers`.
- **Activation Functions:** Intermediate layers use `Softplus` activation functions, while the final layer employs `Tanh` to regulate the output within a specific range.
- **Output Layer:** The output layer has a dimension of `out_dim`.

Optional zero initialization for the last layer can be enabled with `zero_init`.

2) Scaling and Constraints:

- **Input Scaling:** The controller allows for input scaling, adjustable via `input_scaling`.
- **Output Scaling and Constraints:** When the `constrained` flag is set, the outputs are initially confined to [-1, 1] and then rescaled based on `output_scaling`.

C. RandConstController

`RandConstController` functions as a constant controller, generating a uniform random control signal. It is particularly suitable for applications that require a steady control input.

1) Functionality: The controller initializes with a control signal `u0`, constant during the forward pass and defined by its `shape` and values uniformly distributed between `u_min` and `u_max`. `BoxConstrainedController` and `RandConstController` merge machine learning with control theory, providing versatile, efficient solutions. `BoxConstrainedController` is suited for dynamic input tasks within set constraints, while `RandConstController` excels in simpler, constant input scenarios, demonstrating neural networks' utility in control systems.

D. Cost Function Implementation in Control Systems

In this project, we present various cost functions implemented in PyTorch: `IntegralCost`, `circle_loss`, and `circus_loss`. These functions are designed to evaluate and guide the performance of control systems under various operational scenarios.

1) IntegralCost: The `IntegralCost` class, inheriting from PyTorch's `nn.Module`, is developed to compute an integral cost function, which is vital in control tasks.

a) Initialization: The constructor of `IntegralCost` is defined with several key parameters:

- **x_star**: A torch.tensor representing the target position.
- **u_star**: Control input with zero associated cost, defined as a torch.tensor or float.
- **P**: Terminal cost weights, either a scalar or tensor.

- **Q**: State weights, quantifying the cost of deviating from the target state `x_star`.
- **R**: Controller regulator weights to penalize control effort deviation from `u_star`.

b) Forward Method: The forward method calculates the cost for given trajectories (`x`) and control inputs (`u`):

$$\text{Cost} = \|\text{P}\cdot(x_{\text{last}} - x_{\text{star}})\|_2 + \|\text{Q}\cdot(x - x_{\text{star}})\|_2 + \|\text{R}\cdot(u - u_{\text{star}})\|_2 \tag{18}$$

Here, x_{last} represents the final state in the given trajectories.

2) Circle Loss Function: The `circle_loss` function is tailored to enforce a system's trajectory around a circular path.

a) Functionality: This function computes the mean absolute deviation from a circle with radius `a`:

$$\text{circle\_loss} = \text{mean}(|x^2 + y^2 - a|) \tag{19}$$

Here, `z`, representing the state, is split into its x and y components.

3) Circus Loss Function: `circus_loss` is designed to guide the system to follow an elongated, circus-like shape.

a) Functionality: It calculates the mean absolute deviation from a predefined geometric shape:

$$\text{circus\_loss} = \text{mean}(|\sqrt{(x+a)^2 + y^2} \cdot \sqrt{(x-a)^2 + y^2} - k|) \tag{20}$$

Parameters `a` and `k` define the curve's amplitude and the shape's length, respectively.

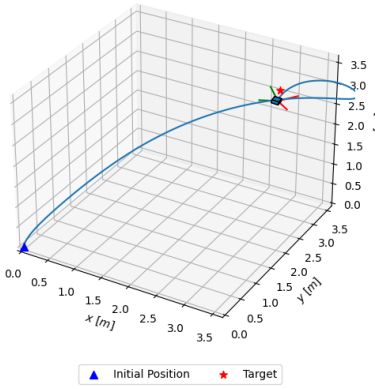

Fig. 3. 3D trajectory of the quadcopter with initial and target positions indicated, illustrating the effects of the implemented perturbations.

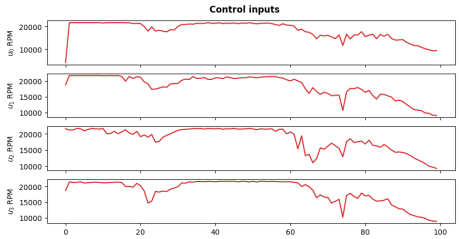

Fig. 4. Control inputs over time for the quadcopter, showing the adaptive response of the system to the applied perturbations.

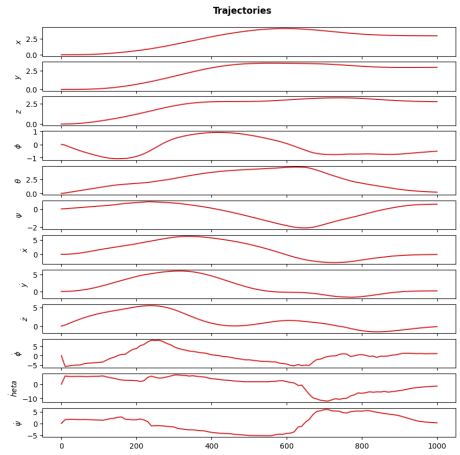

Fig. 5. State trajectories of the quadcopter, highlighting the system's dynamic behavior under the influence of perturbations.

E. Simulation

The implementation of velocity, mass, and motor efficiency perturbations significantly impacts the quadcopter's performance. Figure 3 shows the 3D trajectory of the quadcopter, marking the initial and target positions. Figures 4 and 5 detail the control inputs and state trajectories over time, respectively.

As shown in Figure 3, the quadcopter's trajectory undergoes noticeable deviations from the desired path due to the introduced perturbations. Figure 4 shows the control inputs, which reflect the controller's efforts to compensate for the perturbations, as indicated by the variability in motor RPMs. The state trajectories, depicted in Figure 5, demonstrate the temporal evolution of the quadcopter's position and orientation, affirming the complex nature of the control task when subjected to perturbations.

V. CONCLUSIONS

This study offers a thorough examination of trajectory optimization in autonomous quadrotors, emphasizing the development and evaluation of MPC transcription methods. A notable advancement is the duality-based collision avoidance strategy using polytope representation, enhancing autonomous systems' safety.

Our project enhances MPC effectiveness in real-time contexts by integrating a Neural MPC framework with deep learning. We focused on accurately modeling quadrotor dynamics, including precise force and torque calculations, efficient batch processing via PyTorch, and simulating real-world perturbations, demonstrating their impact on quadcopter performance in dynamic simulations.

This research contributes to the field of autonomous robotics by integrating Model Predictive Control (MPC) with deep learning, offering potential benefits for fields like search-and-rescue and logistics. Future efforts should aim to validate and refine this framework in real-world scenarios, bridging the gap between simulations and the diverse challenges of practical robotic applications.

APPENDIX

Rotation Matrix from Euler Angles

The rotation matrix from Euler angles is:

$$R = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \quad (21)$$

Euler to Quaternion Transformation

The transformation from Euler angles to quaternion is expressed as:

$$q_0 = \cos(\frac{\psi}{2})\cos(\frac{\theta}{2})\cos(\frac{\phi}{2}) + \sin(\frac{\psi}{2})\sin(\frac{\theta}{2})\sin(\frac{\phi}{2}) \quad (22)$$

$$q_1 = \cos(\frac{\psi}{2})\cos(\frac{\theta}{2})\sin(\frac{\phi}{2}) - \sin(\frac{\psi}{2})\sin(\frac{\theta}{2})\cos(\frac{\phi}{2}) \quad (23)$$

$$q_2 = \cos(\frac{\psi}{2})\sin(\frac{\theta}{2})\cos(\frac{\phi}{2}) + \sin(\frac{\psi}{2})\cos(\frac{\theta}{2})\sin(\frac{\phi}{2}) \quad (24)$$

$$q_3 = \sin(\frac{\psi}{2})\cos(\frac{\theta}{2})\cos(\frac{\phi}{2}) - \cos(\frac{\psi}{2})\sin(\frac{\theta}{2})\sin(\frac{\phi}{2}) \quad (25)$$

where ψ, θ, ϕ represents the yaw, pitch, and roll angles, respectively.

TABLE I

SYSTEM CONSTANTS AND PARAMETERS

Parameter	**Symbol**	**Value**
Gravity	G	$9.81\,\text{m/s}^2$
Radians to Degrees Conversion	RAD2DEG	$180/\pi$
Degrees to Radians Conversion	DEG2RAD	$\pi/180$
Mass of the System	M	$0.027\,\text{kg}$
Length Parameter	L	$0.0397\,\text{m}$
Thrust-to-Weight Ratio	THRUST2WEIGHT_RATIO	2.25
Inertia Tensor	J	$\text{diag}(1.4\times10^{-5}, 1.4\times10^{-5}, 2.17\times10^{-5})\,\text{kg}\cdot\text{m}^2$
Inverse of Inertia Tensor	J_{inv}	$\text{linalg.inv}(J)$
Thrust Coefficient	KF	3.16×10^{-10}
Motor Constant	KM	7.94×10^{-12}
Gravity Force	GRAVITY	$G\times M$
Hover RPM	HOVER_RPM	$\sqrt{\text{GRAVITY}/(4\times KF)}$
Maximum RPM	MAX_RPM	$\sqrt{(\text{THRUST2WEIGHT\_RATIO}\times\text{GRAVITY})/(4\times KF)}$
Maximum Thrust	MAX_THRUST	$4\times KF\times\text{MAX\_RPM}^2$
Maximum XY Torque	MAX_XY_TORQUE	$(2\times L\times KF\times\text{MAX\_RPM}^2)/\sqrt{2}$
Maximum Z Torque	MAX_Z_TORQUE	$2\times KM\times\text{MAX\_RPM}^2$

REFERENCES

[1] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017. [Online]. Available: https://doi.org/10.1137/16M1062569

[2] C. R. Hargraves and S. W. Paris, "Direct trajectory optimization using nonlinear programming and collocation," *Journal of guidance, control, and dynamics*, vol. 10, no. 4, pp. 338–342, 1987.

[3] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *Trans. Control Syst. Tech.*, 2021.