Project 1: Trajectory Optimization and MPC for 2D Quadrotor

Reinforcement Learning and Optimal Control for Robotics
ROB-GY 6323

Shantanu Ghodgaonkar
Univ ID: N11344563
Net ID: sng8399

Contents

***Abstract*—This project aims to develop a control framework for a 2D quadrotor to perform acrobatic maneuvers, specifically a looping trajectory. The control system is designed in two parts. First, a trajectory optimization problem is formulated and solved using Sequential Quadratic Programming (SQP), leveraging a cost function that minimizes deviations from the desired trajectory while respecting dynamic and actuator constraints. The quadrotor dynamics are discretized using the Euler method to enable numerical computation. In the second part, the trajectory optimization approach is extended into a Model Predictive Control (MPC) framework to handle real-time perturbations. The MPC is tested in a simulation environment, demonstrating the quadrotor's ability to adapt to disturbances and maintain stability over a 10-second horizon. Results are presented through state and control evolution plots, along with animations of the quadrotor's motion. The findings highlight the effectiveness of the proposed SQP-based trajectory optimization and MPC in achieving precise control of the quadrotor, with potential for extension to more complex scenarios.**

Link to Jupyter Notebook

1. Introduction

This project explores the development of a control system for a 2D quadrotor to perform a looping maneuver, showcasing the capabilities of advanced optimization-based control techniques. The system dynamics are defined by a set of nonlinear equations that model the quadrotor's position, velocity, and orientation. Control inputs are the thrust forces generated by the quadrotor's rotors, which must be regulated to achieve the desired trajectory while adhering to physical constraints.

The project is divided into two parts. In Part 1, a trajectory optimization problem is formulated and solved using Sequential Quadratic Programming (SQP). The optimization aims to minimize deviations from the desired looping trajectory while satisfying the quadrotor's dynamic constraints. The solution is implemented in Python, and simulations are conducted to verify the controller's ability to achieve the looping maneuver.

In Part 2, the trajectory optimization framework is extended into a Model Predictive Control (MPC) system. MPC enables real-time control by solving an optimization problem at each time step, making it suitable for handling disturbances and adapting to changes in the system. The MPC is tested in a simulated environment with disturbances to evaluate its robustness and real-time performance.

This report details the formulation of the optimization problems, the design and implementation of the control algorithms, and the results of the simulations. The findings highlight the potential of optimization-based methods for precise and adaptive control of quadrotors, with insights into their applicability in real-world scenarios.

2. SQP Formulation

2.1. System Analysis

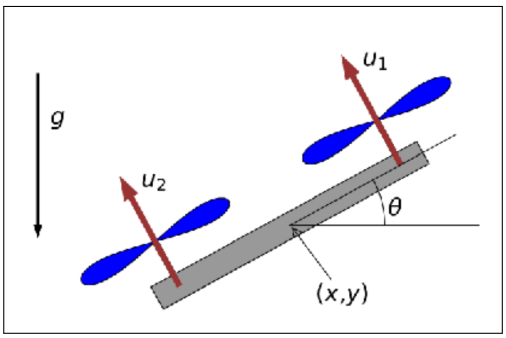

Figure 1. The given 2D Quadrotor

The 2D quadrotor is a simplified representation of a quadrotor's dynamics in a two-dimensional plane, capturing its essential physical properties and control challenges. This section provides a detailed analysis of the system's dynamics, control variables, and constraints to form the foundation for the control system design.

2.1.1. Dynamic Model. The quadrotor's motion is governed by nonlinear dynamics derived from Newtonian mechanics. The state of the system is represented as:

$$x = \begin{bmatrix} p_x \\ v_x \\ p_y \\ v_y \\ \theta \\ \omega \end{bmatrix},$$

where:

- p_x, p_y: Horizontal and vertical positions.
- v_x, v_y: Linear velocities in the horizontal and vertical directions.
- θ: Orientation (angle with respect to the horizontal plane).
- ω: Angular velocity.

The control inputs are the thrust forces generated by the rotors:

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix},$$

where u_1 and u_2 are the forces produced by the left and right rotors, respectively.

The equations of motion are given by:

$$\begin{aligned} \dot{p_x} &= v_x, \\ m\dot{v}_x &= -(u_1 + u_2)\sin\theta, \\ \dot{p_y} &= v_y, \\ m\dot{v}_y &= (u_1 + u_2)\cos\theta - mg, \\ \dot{\theta} &= \omega, \\ I\dot{\omega} &= r(u_1 - u_2). \end{aligned}$$

Here:

- m: Mass of the quadrotor.
- I: Moment of inertia.
- r: Distance from the center of mass to each rotor.
- g: Acceleration due to gravity.

2.1.2. Constraints. The quadrotor's control inputs and physical dynamics impose the following constraints:

1) **Control Limits**: Thrust forces are constrained to:
$$0 \leq u_1, u_2 \leq 10 \,(\text{units of force}).$$
2) **Dynamic Feasibility**: The state transitions must satisfy the discretized equations of motion.

2.1.3. Control Objectives. The primary objective of the control system is to drive the quadrotor to execute a looping maneuver. This requires:

- Accurate trajectory tracking.
- Smooth transitions in control inputs.
- Minimal deviation from the desired trajectory.

2.2. Discretization of System Dynamics

To discretize the system dynamics, we rearrange the given dynamic equations such that multiplication with Δt provides the necessary addition to the current

state, resulting in the next state. This approach ensures that the continuous-time dynamics are appropriately transformed into a discrete-time framework suitable for numerical computation. The system dynamics are discretized as shown below:

$$\begin{aligned}
\dot{p}_{x_n} &= v_{x_n}, \\
\dot{v}_{x_n} &= -\frac{(u_{1_n} + u_{2_n}) \sin\theta_n}{m}, \\
\dot{p}_{y_n} &= v_{y_n}, \\
\dot{v}_{y_n} &= \frac{(u_{1_n} + u_{2_n}) \cos\theta_n}{m} - g, \\
\dot{\theta}_n &= \omega_n, \\
\dot{\omega}_n &= \frac{L(u_{1_n} - u_{2_n})}{I}
\end{aligned}$$

From these, the discrete-time dynamics are derived as equality constraints for trajectory optimization:

$$\begin{aligned}
p_{x_{n+1}} &= p_{x_n} + \Delta t v_{x_n}, \\
v_{x_{n+1}} &= v_{x_n} - \Delta t \frac{(u_{1_n} + u_{2_n}) \sin\theta_n}{m}, \\
p_{y_{n+1}} &= p_{y_n} + \Delta t v_{y_n}, \\
v_{y_{n+1}} &= v_{y_n} + \Delta t \left(\frac{(u_{1_n} + u_{2_n}) \cos\theta_n}{m} - g \right), \\
\theta_{n+1} &= \theta_n + \Delta t \omega_n, \\
\omega_{n+1} &= \omega_n + \Delta t \frac{L(u_{1_n} - u_{2_n})}{I}.
\end{aligned}$$

This discretization ensures that the state evolution in the optimization problem adheres to the physical dynamics of the quadrotor. The time step Δt is a key parameter that determines the resolution of the discretization and impacts both the accuracy and computational cost of the numerical solution.

2.3. Linearization of Equality Constraints

As we can see, a couple of the constraints are nonlinear. To solve the optimization problem effectively, we must find a linear approximation of the constraints at a given guess $\bar{x}$. This can be expressed in the form:

$$A(\bar{x})\Delta x = b(\bar{x}),$$

where Δx represents a small variation around $\bar{x}$. To achieve this, we perform a Taylor Expansion of the constraints around $\bar{x}$, resulting in the following linearized equations:

$$\Delta p_{x_{n+1}} = \Delta p_{x_n} + \Delta t \Delta v_{x_n}, \tag{1}$$

$$\Delta v_{x_{n+1}} = \Delta v_{x_n} - \Delta t \Delta\theta_n \frac{(u_{1_n} + u_{2_n}) \cos\theta_n}{m} - \Delta t (\Delta u_{1_n} + \Delta u_{2_n}) \frac{\sin\theta_n}{m}, \tag{2}$$

$$\Delta p_{y_{n+1}} = \Delta p_{y_n} + \Delta t \Delta v_{y_n}, \tag{3}$$

$$\Delta v_{y_{n+1}} = \Delta v_{y_n} - \Delta t \Delta\theta_n \frac{(u_{1_n} + u_{2_n}) \sin\theta_n}{m} + \Delta t (\Delta u_{1_n} + \Delta u_{2_n}) \frac{\cos\theta_n}{m}, \tag{4}$$

$$\Delta\theta_{n+1} = \Delta\theta_n + \Delta t \Delta\omega_n, \tag{5}$$

$$\Delta\omega_{n+1} = \Delta\omega_n + \Delta t \frac{L(\Delta u_{1_n} - \Delta u_{2_n})}{I}. \tag{6}$$

Finally, these equations can be arranged in the matrix form:

$$A(y)\Delta y = b(y),$$

where A is the Jacobian of the constraints, Δy represents the small variation around the combined state and control vector, and b represents the constraints themselves.

$$A = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
1 & \Delta t & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & \cdots \\
0 & 1 & 0 & 0 & -\Delta t \frac{(u_{1_n}+u_{2_n})\cos\theta_n}{m} & 0 & -\Delta t \frac{\sin\theta_n}{m} & -\Delta t \frac{\sin\theta_n}{m} & 0 & -1 & 0 & 0 & 0 & \cdots \\
0 & 0 & 1 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & \cdots \\
0 & 0 & 0 & 1 & -\Delta t \frac{(u_{1_n}+u_{2_n})\sin\theta_n}{m} & 0 & \Delta t \frac{\cos\theta_n}{m} & \Delta t \frac{\cos\theta_n}{m} & 0 & 0 & 0 & -1 & 0 & \cdots \\
0 & 0 & 0 & 0 & 1 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & -1 & \cdots \\
0 & 0 & 0 & 0 & 0 & 1 & \Delta t \frac{I}{I} & -\Delta t \frac{I}{I} & 0 & 0 & 0 & 0 & 0 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{bmatrix}$$

$$\Delta y = \begin{bmatrix} \Delta X_0 \\ \Delta U_0 \\ \Delta X_1 \\ \Delta U_1 \\ \vdots \end{bmatrix}$$

$$b = \begin{bmatrix}
X_{init} - X_0 \\
p_{x_n} + \Delta t \cdot v_{x_n} - p_{x_{n+1}} \\
v_{x_n} - \Delta t \frac{(u_{1_n}+u_{2_n})\sin\theta_n}{m} - v_{x_{n+1}} \\
p_{y_n} + \Delta t \cdot v_{y_n} - p_{y_{n+1}} \\
v_{y_n} + \Delta t \left(\frac{(u_{1_n}+u_{2_n})\cos\theta_n}{m} - g \right) - v_{y_{n+1}} \\
\theta_n + \Delta t \cdot \omega_n - \theta_{n+1} \\
\omega_n + \Delta t \frac{L(u_{1_n}-u_{2_n})}{I} - \omega_{n+1} \\
\vdots
\end{bmatrix}.$$

Here, X_{init} is the initial state of the quadrotor. Since it is not explicitly defined in the constraints and may vary for MPC, we must ensure the system provides functionality to make it variable.

2.4. Linearization of Inequality Constraints

To incorporate inequality constraints into the optimization problem, we linearize them around a nominal guess $\bar{y}$. This section outlines the mathematical formulation of these constraints, their linearization, and their matrix representation.

It is given that the inequality constraints for the system are as follows:

1) The vertical position p_y must be non-negative:

$$p_y \geq 0$$

2) The control inputs u_1 and u_2 must remain within specified bounds:

$$0 \leq u_1 \leq 10, \quad 0 \leq u_2 \leq 10$$

These inequalities ensure the physical feasibility of the solution (e.g., the quadrotor does not penetrate the ground and control inputs remain within allowable limits).

To handle these nonlinear constraints within an optimization framework, we linearize them with respect to small variations Δy around the nominal state $\bar{y}$. Performing a Taylor expansion for the inequality constraints gives:

1) Linearized constraints for vertical position:

$$\Delta p_y \geq -\bar{p}_y$$

2) Linearized constraints for the control inputs:

$$\begin{aligned} \Delta u_1 &\leq 10 - \bar{u}_1, \\ \Delta u_1 &\geq -\bar{u}_1, \\ \Delta u_2 &\leq 10 - \bar{u}_2, \\ \Delta u_2 &\geq -\bar{u}_2. \end{aligned}$$

These linearized inequalities represent a first-order approximation of the original constraints around the nominal trajectory.

The linearized inequality constraints can be expressed in the matrix form:

$$G(y)\Delta y \leq h(y)$$

where:

- G is the Jacobian matrix of the inequality constraints.
- Δy represents variations around the combined state and control vector.
- h is the vector containing the right-hand side values of the inequalities.

The matrix G is constructed as a block diagonal matrix, where each block corresponds to the Jacobian of the constraints:

$$G = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

The vector h is formed as:

$$h = \begin{bmatrix} \bar{p}_{y_0} \\ 10 - \bar{u}_{1_0} \\ \bar{u}_{1_0} \\ 10 - \bar{u}_{2_0} \\ \bar{u}_{2_0} \\ \bar{p}_{y_1} \\ 10 - \bar{u}_{1_1} \\ \bar{u}_{1_1} \\ 10 - \bar{u}_{2_1} \\ \bar{u}_{2_1} \\ \vdots \end{bmatrix}$$

2.5. Trajectory Definition: Quadrotor Loop

To enable the quadrotor to perform a looping maneuver, a suitable trajectory must be defined. This section explains the mathematical representation of the trajectory, dividing it into distinct phases based on the looping path. The trajectory is designed in terms of the quadrotor's x- and y-positions (p_x, p_y) and its orientation θ.

2.5.1. Trajectory Design. The looping trajectory is a closed path designed to guide the quadrotor through a series of predefined positions and orientations. The trajectory is divided into five segments over one wavelength ($N_{\text{WAVE}} = 100$):

1) **Start Position:** The quadrotor begins at the origin $(p_x, p_y) = (0, 0)$ with an initial orientation $\theta = 0$.
2) **Ascent to First Corner:** The quadrotor moves to the top-right corner $(p_x, p_y) = (1, 3)$.
3) **Peak Position:** The quadrotor reaches the top of the loop $(p_x, p_y) = (0, 3)$.
4) **Descent to Final Corner:** The quadrotor moves to the top-left corner $(p_x, p_y) = (-1, 3)$.
5) **Return to Start:** The quadrotor completes the loop by returning to the origin.

The orientation θ evolves continuously over the trajectory, completing a full 2π rotation during the loop.

2.5.2. Parametric Representation. For each discrete step k in the trajectory, the position (p_x, p_y) and orientation θ_k are defined as follows:

$$\theta_k = \frac{k}{100} \cdot 2\pi,$$

$$(p_x, p_y) = \begin{cases} (0, 0), & \text{if } k < 20, \\ (1, 3), & \text{if } 20 \leq k < 40, \\ (0, 3), & \text{if } 40 \leq k < 60, \\ (-1, 3), & \text{if } 60 \leq k < 80, \\ (0, 0), & \text{if } 80 \leq k \leq 100. \end{cases}$$

2.6. Cost Function Formulation

The goal of the cost function is to penalize deviations of the quadrotor's state and control inputs

from their desired values. This section outlines the derivation of the cost function, its expanded form, and its final representation in matrix form, which is suitable for optimization.

2.6.1. Cost Function Derivation. Given the desired trajectory, the cost function can be formulated as:

$$\frac{1}{2}\left(\sum_{k=0}^{N-1}(X_k - X_{k_{\text{des}}})^T Q(X_k - X_{k_{\text{des}}}) + U_k^T R U_k + (X_N - X_{N_{\text{des}}})^T Q(X_N - X_{N_{\text{des}}})\right) \quad (7)$$

where:

- X_k: The state of the quadrotor at step k,
- $X_{k_{\text{des}}}$: The desired state at step k,
- U_k: The control input at step k,
- Q: A positive semi-definite matrix that penalizes deviations in the state,
- R: A positive definite matrix that penalizes control effort,
- N: The total number of steps.

This cost function penalizes:

1) Deviations of the quadrotor's state X_k from the desired state $X_{k_{\text{des}}}$,
2) The magnitude of control inputs U_k.

2.6.2. Expanded Cost Function. Expanding the cost function and ignoring constant terms (independent of X_k and U_k) gives:

$$\sum_{k=0}^{N-1}\left(\frac{1}{2}X_k^T Q X_k + \frac{1}{2}U_k^T R U_k - X_{k_{\text{des}}}^T Q X_k\right) + \frac{1}{2}X_N^T Q X_N - X_{N_{\text{des}}}^T Q X_N. \quad (8)$$

The goal is to minimize this cost function, expressed as:

$$\min_{X_k, U_k} \frac{1}{2}\left(\sum_{k=0}^{N-1} X_k^T Q X_k + U_k^T R U_k - 2X_{k_{\text{des}}}^T Q X_k\right) + \frac{1}{2}X_N^T Q X_N - X_{N_{\text{des}}}^T Q X_N. \quad (9)$$

2.6.3. Matrix Representation. To facilitate numerical optimization, the cost function is reformulated in matrix form as:

$$\min_y \frac{1}{2}y^T P y + q^T y,$$

where:

- y: The combined state and control vector,

$$y = \begin{bmatrix} X_0 \\ U_0 \\ X_1 \\ U_1 \\ \vdots \\ X_N \end{bmatrix}.$$

- P: The block diagonal matrix representing weights for states and controls,

$$P = \begin{bmatrix} Q & 0 & 0 & 0 & \cdots \\ 0 & R & 0 & 0 & \cdots \\ 0 & 0 & Q & 0 & \cdots \\ 0 & 0 & 0 & R & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

- q: The vector representing the linear terms in the cost function,

$$q^T = \begin{bmatrix} -X_{0_{\text{des}}}^T Q & 0 & -X_{1_{\text{des}}}^T Q & 0 & \cdots & -X_{N_{\text{des}}}^T Q \end{bmatrix}.$$

2.6.4. Penalizing State Variables. To focus the cost function on penalizing deviations in the quadrotor's position, we choose the diagonal elements of Q such that only the positional components of the state (p_x, p_y) are penalized. Similarly, R is tuned to penalize control inputs appropriately. Initial values for Q and R are chosen as:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

where:

- Q: Penalizes deviations in p_x and p_y, while ignoring other states such as velocities and orientation.
- R: Penalizes both control inputs u_1 and u_2 equally.

2.6.5. Gradient and Hessian of the Cost Function. Let us recall from Lecture 2 a shortcut method to compute the gradient of a quadratic cost function. If a function is of the form:

$$f(x) = x^T P x + q^T x,$$

then:

$$\nabla f(x) = 2Px + q, \quad \nabla^2 f(x) = 2P.$$

Applying this to our cost function:

- The gradient of the cost function is:

$$\nabla J(y) = Py + q.$$

- The Hessian of the cost function is:

$$\nabla^2 J(y) = P.$$

2.7. KKT Condition Formulation

The Karush-Kuhn-Tucker (KKT) conditions provide a necessary set of conditions for optimality in constrained optimization problems. With all the prerequisites in place, the KKT conditions for our optimization problem can be formulated as follows.

2.7.1. KKT Conditions. The optimization problem can be expressed as:

$$\begin{aligned} \min_{p} \quad & P^T \nabla^2_{yy} \mathcal{L}(y_k) P + q^T \nabla f(y_k), \\ \text{subject to} \quad & \nabla g(y_k)^T P + g(y_k) = 0, \\ & \nabla h(y_k)^T P + h(y_k) \leq 0, \end{aligned} \tag{10}$$

where:

- $\nabla^2_{yy} \mathcal{L}(y_k)$: Hessian of the Lagrangian $\mathcal{L}(y_k)$,
- $\nabla f(y_k)$: Gradient of the cost function,
- $\nabla g(y_k)$: Jacobian of equality constraints,
- $g(y_k)$: Equality constraint function,
- $\nabla h(y_k)$: Jacobian of inequality constraints,
- $h(y_k)$: Inequality constraint function.

2.7.2. Matrix Form of the KKT Conditions. The above conditions are equivalent to solving the following system of equations:

$$\begin{bmatrix} \nabla^2_{yy} \mathcal{L}(y_k) & \nabla g(y_k)^T & \nabla h(y_k)^T \\ \nabla g(y_k) & 0 & 0 \\ \nabla h(y_k) & 0 & 0 \end{bmatrix} \begin{pmatrix} p_k \\ \lambda_{k+1} \\ \mu_{k+1} \end{pmatrix} = \begin{pmatrix} -\nabla f(y_k) \\ -g(y_k) \\ -h(y_k) \end{pmatrix}$$

Here:

- p_k: Search direction for the optimization variables,
- λ_{k+1}: Lagrange multipliers for equality constraints,
- μ_{k+1}: Lagrange multipliers for inequality constraints.

To solve the KKT system efficiently, we use the `qpsolvers` library. Specifically, we refer to the example provided in the official documentation. This library is well-suited for quadratic programming problems and provides direct methods for computing the primal and dual solutions, including the Lagrange multipliers.

2.8. Computing the Total Constraint Violation

To evaluate whether a solution satisfies the constraints of the optimization problem, we calculate the *total constraint violation*, which consists of violations from both equality and inequality constraints. This ensures the solution adheres to the problem's physical and operational requirements.

2.8.1. Equality Constraint Violation. The equality constraints, derived from the system dynamics, are expressed as:

$$b(y) = 0,$$

where $b(y)$ is the vector of equality constraint residuals. Each element of $b(y)$ corresponds to the deviation from the expected system dynamics at a particular step.

The *equality constraint violation* is computed as:

$$\text{Equality Violation} = \sum_{i=1}^{N} |b_i|$$

where:

- b_i: Residual of the equality constraint at step i,
- N: Total number of steps.

This value quantifies the cumulative deviation of the solution from the system dynamics.

2.8.2. Inequality Constraint Violation. The inequality constraints impose bounds on the system's state and control variables. For the quadrotor, these are:

1) **Control Input Limits:**

$$0 \leq u_1 \leq 10, \quad 0 \leq u_2 \leq 10$$

where u_1 and u_2 are the thrust forces generated by the rotors.

2) **Non-Negative Vertical Position:**

$$p_y \geq 0$$

where p_y is the vertical position of the quadrotor.

The violation for each constraint is computed as the amount by which a variable exceeds or falls short of its allowable range. The *inequality constraint violation* is given by:

$$\begin{aligned} &\text{Inequality Violation} = \\ &\sum_{i=1}^{N} \Big(\max(0, u_1 - 10) + \max(0, -u_1) \\ &+ \max(0, u_2 - 10) + \max(0, -u_2) \\ &+ \max(0, -p_y) \Big) \end{aligned} \tag{11}$$

where:

- $\max(0, u_1 - 10)$: Amount by which u_1 exceeds its upper limit,
- $\max(0, -u_1)$: Amount by which u_1 falls below its lower limit,
- $\max(0, u_2 - 10)$ and $\max(0, -u_2)$: Similar terms for u_2,
- $\max(0, -p_y)$: Amount by which p_y falls below 0.

2.8.3. Total Constraint Violation. The *total constraint violation* combines the equality and inequality violations:

$$\text{Total Violation} = \text{Equality Violation} + \text{Inequality Violation}.$$

This provides a single scalar value quantifying the total deviation of the solution from the problem's constraints.

A smaller total constraint violation indicates better adherence to the system's physical and operational constraints, ensuring feasibility of the solution.

2.9. Filter Line Search

Finally we shall use the Filter Line Search algorithm to iteratively minimize the cost function while simultaneously reducing constraint violations. This algorithm adjusts a step direction and step size to ensure progress in both objectives—reducing cost and satisfying constraints.

Algorithm 1 Filter Line Search Algorithm

Initialize: Initial guess y_{guess}, initial step size $\alpha = 1.0$, reduction factor $\rho \in (0, 1)$, tolerance tol, maximum iterations MAX_ITER, and histories for cost, constraint violation, and step size.
Set $J_{\text{best}} = \infty$, $\text{Violation}_{\text{best}} = \infty$.
for $i = 1, 2, \ldots,$ MAX_ITER **do**
    Solve the KKT system to compute the search direction p_k.
    Set $y_{\text{new}} = y_{\text{guess}} + \alpha \cdot p_k$.
    Compute the cost $J(y_{\text{new}})$ and total constraint violation $\text{Violation}(y_{\text{new}})$.
    while $J(y_{\text{new}}) \geq J_{\text{best}}$ **and** $\text{Violation}(y_{\text{new}}) \geq \text{Violation}_{\text{best}}$ **do**
        Reduce step size: $\alpha \leftarrow \rho \cdot \alpha$.
        if $\alpha < 10^{-6}$ **then**
            Terminate the line search and exit.
        end if
        Update $y_{\text{new}} = y_{\text{guess}} + \alpha \cdot p_k$.
        Recompute $J(y_{\text{new}})$ and $\text{Violation}(y_{\text{new}})$.
    end while
    Update $J_{\text{best}} = J(y_{\text{new}})$ and $\text{Violation}_{\text{best}} = \text{Violation}(y_{\text{new}})$.
    Update $y_{\text{guess}} = y_{\text{new}}$.
    Record α, J_{best}, and $\text{Violation}_{\text{best}}$ in their respective histories.
    if $\text{Violation}_{\text{best}} < \text{tol}$ **then**
        Terminate the algorithm and exit.
    end if
end for
Output: Optimized states x and control inputs u.

2.10. Simulation

The simulation demonstrates the effectiveness of the formulated Sequential Quadratic Programming (SQP) approach to control the quadrotor in achieving a looping trajectory. The steps for the simulation are as follows:

2.10.1. Initialization. The quadrotor simulation is initialized with the following parameters:

- **Number of steps (N)**: 100, representing the discretized time steps for the trajectory.
- **Initial State (x_{init})**: The quadrotor starts from rest with all states initialized to zero:

$$x_{\text{init}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

- **Initial Guess (y_{guess})**: A flat guess for the combined state and control vector over all N steps:

$$y_{\text{guess}} = 0 \quad (\text{size: } (6+2) \times N)$$

2.10.2. Optimization. Using the **Filter Line Search Algorithm**, the control inputs and state trajectory are iteratively optimized to minimize the cost function while satisfying the system constraints:

$$x, u = \texttt{filter\_line\_search}(\ldots),$$

where:

- x: Optimized state trajectory
- u: Optimized control inputs

The tuned cost matrices used in the optimization are:

$$Q = \begin{bmatrix} 31.25 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.03125 & 0 & 0 & 0 & 0 \\ 0 & 0 & 31.25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.03125 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3.125 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.03125 \end{bmatrix}$$

$$R = \begin{bmatrix} 0.75 & 0 \\ 0 & 0.75 \end{bmatrix}$$

2.10.3. Visualization. The optimized trajectory and control inputs are animated using a simulation of the quadrotor:

$$\texttt{quadrotor.animate\_robot}(x, u)$$

This simulation visualizes:

1) **Quadrotor Path:** The trajectory $x = [p_x, p_y]$ in 2D space, showing the looping maneuver.
2) **Control Inputs:** The thrust forces $u = [u_1, u_2]$ applied to achieve the desired trajectory.

3. Model Predictive Control (MPC)

Model Predictive Control (MPC) is a powerful control strategy used to handle dynamic systems with constraints and disturbances. In this section, we simulate the quadrotor using MPC, demonstrating its capability to adapt to changing conditions and disturbances.

3.1. MPC Framework

MPC operates by solving an optimization problem at each time step to determine the optimal control actions over a finite prediction horizon. Only the first control action is implemented, and the process is repeated in a receding horizon fashion. The components of the MPC framework are:

1) **State Prediction:** Starting from the current state $z_0 = [p_x, v_x, p_y, v_y, \theta, \omega]$, the quadrotor's trajectory is predicted over a finite horizon ($N = 50$) using the system dynamics.
2) **Optimization Problem:** At each time step, the MPC solves a quadratic optimization problem to minimize the cost function while satisfying system constraints:

$$\min \frac{1}{2} \sum_{k=0}^{N-1} \left((X_k - X_{k_{\text{des}}})^T Q (X_k - X_{k_{\text{des}}}) + U_k^T R U_k \right).$$

Constraints include control limits, dynamic feasibility, and initial conditions.

3) **Receding Horizon Implementation:** The first control action u_0 is applied to the system, and the optimization problem is re-solved at the next time step with updated states.
4) **Disturbance Handling:** Random disturbances are introduced during the simulation to evaluate the robustness of the MPC controller.

3.2. Simulation Setup

The simulation is designed with the following components:

1) **Initial State:** The quadrotor starts from rest:
$$z_0 = [0, 0, 0, 0, 0, 0]^T.$$
2) **Prediction Horizon:** A prediction horizon of $N = 50$ steps is used to balance computational efficiency and accuracy.
3) **Controller:** The controller function:
   - Solves the optimization problem using the *Filter Line Search Algorithm*.
   - Outputs the first control action u_0 for implementation.
4) **Disturbances:** Random disturbances are applied every 25 steps to simulate external perturbations, modeled as:
$$\text{dist} \sim \mathcal{U}(-1, 1),$$
affecting the velocities v_x, v_y, ω.
5) **Simulation Horizon:** The simulation runs for 300 time steps ($h = 300$).

3.3. Control Logic

At each time step, the following steps are executed:

1) **Control Input Calculation:** The MPC controller predicts the optimal trajectory x and control sequence u using the current state z_k and desired trajectory $X_{k_{\text{des}}}$.
2) **State Propagation:** The system evolves to the next state using the dynamics:
$$z_{k+1} = \text{next\_state}(z_k, u_k),$$
where u_k is the control input applied at step k.
3) **Disturbance Injection (Optional):** If enabled, random disturbances are added to the state vector at periodic intervals.
4) **Time Update:** The simulation time is updated:
$$t_{k+1} = t_k + \Delta t.$$

3.4. Simulation

The simulation results are visualized using an animation of the quadrotor's trajectory and control inputs:

`quadrotor.animate_robot`(z, u).

4. Results

This section presents the results of the optimization and control strategies employed for the quadrotor. The outcomes are divided into two parts: the performance of the Sequential Quadratic Programming (SQP) approach, including the bonus objectives, and the robustness of the Model Predictive Control (MPC) under disturbances.

4.1. SQP and Bonus Objectives

4.1.1. Trajectory Optimization with SQP. The SQP algorithm successfully optimized the quadrotor's trajectory to perform a looping maneuver. Key observations include:

- **Trajectory Tracking:** The optimized trajectory closely follows the desired looping path, with minimal deviation.
- **Constraint Satisfaction:** The equality and inequality constraints were satisfied within the specified tolerance (10^{-4}), ensuring physical feasibility.
- **Control Effort:** The control inputs were fairly smooth and remained within the bounds:
$$0 \leq u_1, u_2 \leq 10.$$

4.1.2. Bonus Objective: Ensuring Positive Altitude. In addition to the control constraints, another bonus constraint was successfully observed:

$$p_y \geq 0$$

Without this constraint, the quadrotor would fall off with $p_y < 0$ at the end of the time horizon. This constraint prevents that from happening.

Thus, the performance of the SQP approach highlights its suitability for trajectory optimization in constrained and dynamic environments. The obtained plotted results are shown below:

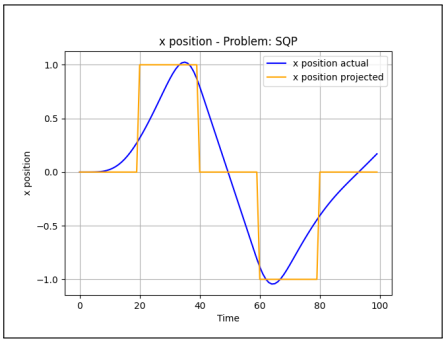

Figure 2. X Position SQP

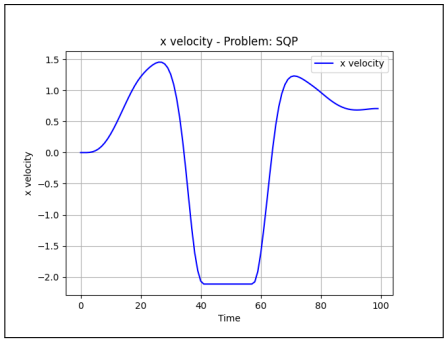

Figure 3. X Velocity SQP

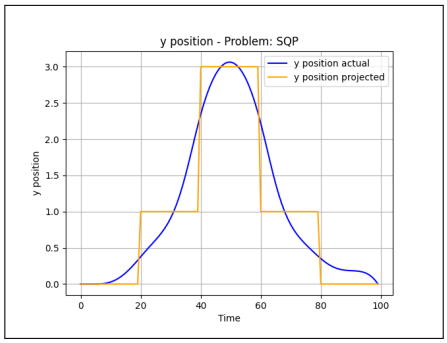

Figure 4. Y Position SQP

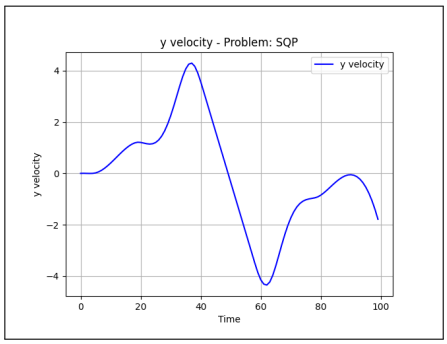

Figure 5. Y Velocity SQP

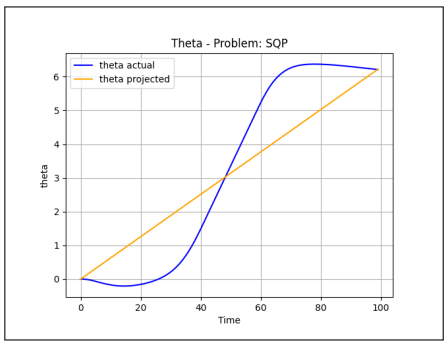

Figure 6. Theta SQP

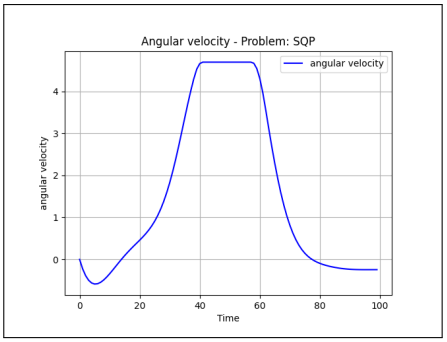

Figure 7. Angular Velocity SQP

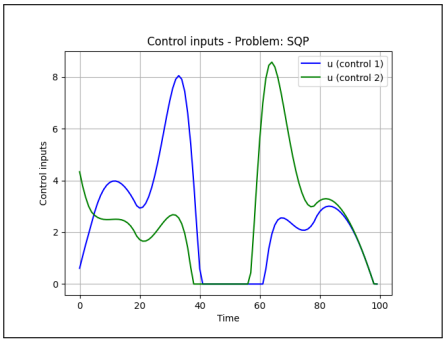

Figure 8. Control Inputs SQP

4.2. MPC Results

The MPC simulation was conducted to test the quadrotor's ability to track the looping trajectory in real-time under disturbances. Key results include:

4.2.1. Trajectory Tracking.

- **Desired Path:** The quadrotor successfully tracked the desired looping trajectory with minor deviations.
- **Dynamic Re-planning:** The receding horizon implementation of MPC allowed the quadrotor to continuously re-plan its trajectory, accounting for disturbances and ensuring accuracy.

4.2.2. Control Inputs.

- **Smooth Control:** The control inputs u_1, u_2 remained smooth and adhered to the physical constraints:

$$0 \leq u_1, u_2 \leq 10.$$

- **Robustness:** The control inputs adjusted dynamically to counteract disturbances, maintaining stability and feasibility.

4.2.3. Disturbance Handling.

- **Random Perturbations:** The quadrotor maintained trajectory tracking despite random disturbances applied every 25 steps.
- **Recovery Time:** The system quickly recovered from disturbances, with minimal impact on the trajectory.

4.2.4. Cost Function Analysis. The cost function values across the MPC simulation showed consistency, indicating effective optimization and adherence to the desired trajectory. The system's ability to adapt to external forces while maintaining low costs highlights the efficiency of the MPC approach.

4.2.5. Summary of MPC Results. The MPC approach demonstrated:

- **Robust Control:** Effective handling of random disturbances with fast recovery.
- **Trajectory Accuracy:** Successful tracking of the looping trajectory with minimal deviations.
- **Real-Time Feasibility:** Computational efficiency that supports real-time implementation.

The combination of predictive re-planning, robustness, and trajectory accuracy underscores the effectiveness of the MPC framework for controlling the quadrotor in dynamic environments. The resultant plots are shown below:

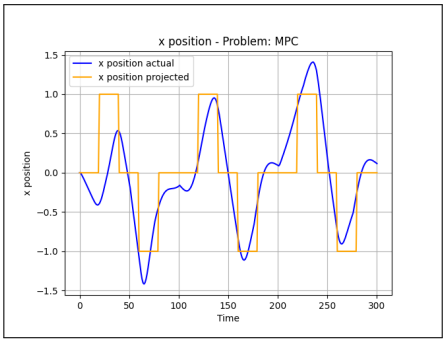

Figure 9. X Position MPC

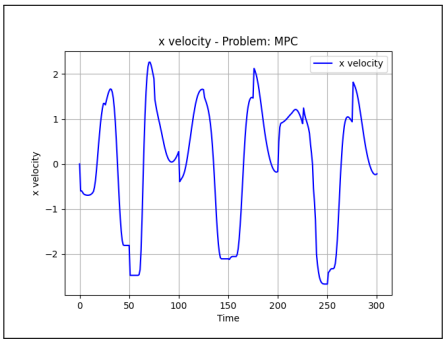

Figure 10. X Velocity MPC

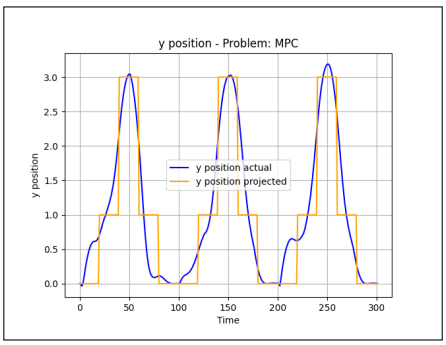

Figure 11. Y Position MPC

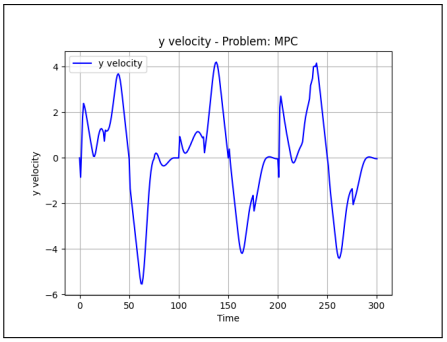

Figure 12. Y Velocity MPC

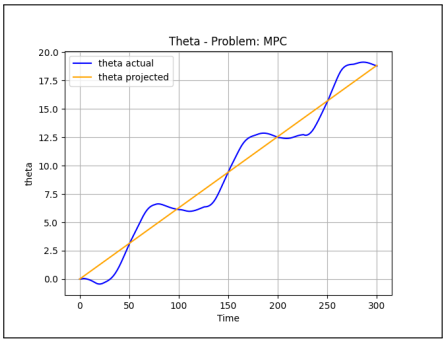

Figure 13. Theta MPC

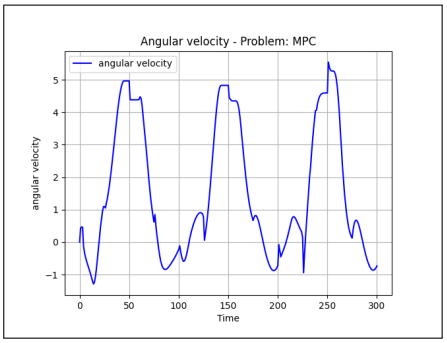

Figure 14. Angular Velocity MPC

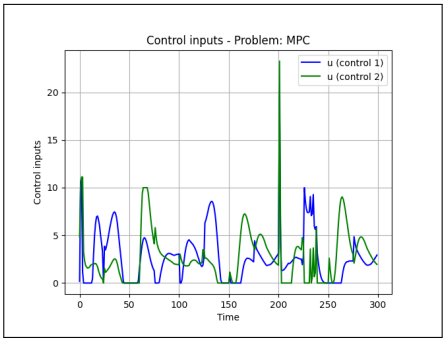

Figure 15. Control Inputs MPC