Reinforcement learning and Optimal control for Robotics (ROB-GY 6323)

New York University, Fall 2024

Due on 11/24, 11:59 PM

Name: Raman Kumar Jha
NYU ID: N13866145

Project 1

Perform a looping with the quadrotor

Goal of the project

The goal of this project is to control a 2D quadrotor to get it to perform acrobatic moves. The controller will be designed using an SQP solver.

2D quadrotor

The quadrotor model is written as:

$$\dot{p_x} = v_x \tag{1}$$
$$m\dot{v}_x = -(u_1 + u_2)\sin\theta \tag{2}$$
$$\dot{p_y} = v_y \tag{3}$$
$$m\dot{v}_y = (u_1 + u_2)\cos\theta - mg \tag{4}$$
$$\dot{\theta} = \omega \tag{5}$$
$$I\dot{\omega} = r(u_1 - u_2) \tag{6}$$

where p_x is the horizontal and p_y the vertical positions of the quadrotor and θ is its orientation with respect to the horizontal plane. v_x and v_y are the linear velocities and ω is the angular velocity of the robot. u_1 and u_2 are the forces produced by the rotors (our control inputs). m is the quadrotor mass, I its moment of inertia (a scalar), r is the distance from the center of the robot frame to the propellers and g is the gravity constant.

To denote the entire state, we will write $x = [p_x, v_x, p_y, v_y, \theta, \omega]^T$ - we will also write $u = [u_1, u_2]^T$.

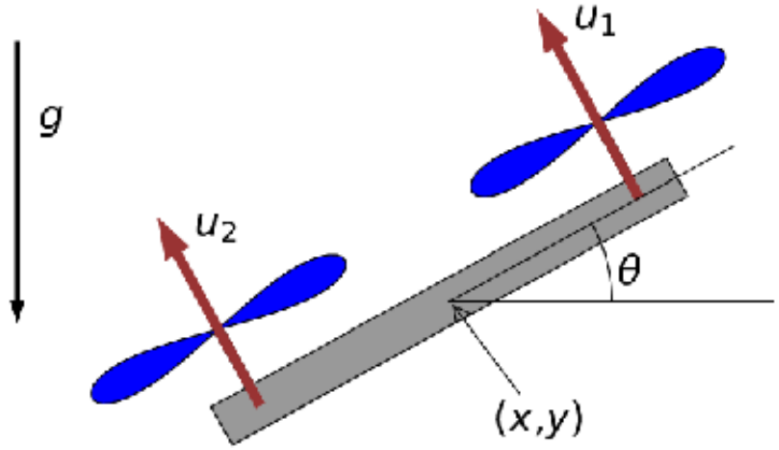

Figure 1: 2D Quadrotor

Part 1 - Setting up the trajectory Optimization (50 points)

1. Discretize the system dynamics using the Euler method seen in class - write the time discretization step as Δt (use symbols not numbers for the mass, etc)

2. We would like the quadrotor to perform a looping. Find and implement a suitable cost function to perform a looping and add constraint to maintain the thrust of each rotor between 0 and 10. Solve the problem using your own implementation of a SQP (leveraging your code from Homework 2) with a large horizon to check that you can do a looping.

3. Show plots of all the states and controls of the robot as a function of time. Describe your design choices (in a concise manner) in the report.

1 2D Quadrotor Control System

1.1 System Overview

The quadrotor is a 2D system with two rotors that generate thrust forces u_1 and u_2. The system has 6 states:

- Position: (p_x, p_y)
- Linear velocity: (v_x, v_y)

- Orientation: θ
- Angular velocity: ω

1.2 System Dynamics

The continuous dynamics are given by:

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

1.3 Implementation Approach

A simple but effective approach was used to perform the trajectory optimization of the 2D quadrotor control system.

1.3.1 Discretization

Used Euler method with step size $dt = 0.04s$:

$$z_{k+1} = z_k + f(z_k, u_k)dt \tag{7}$$

1.3.2 Cost Function

Designed a cost that balances tracking and control effort:

$$J = \sum_{k=0}^{N-1} [(x_k - x_d)^2 + (y_k - y_d)^2] + 0.01 \sum_{k=0}^{N-1} (u_1^2 + u_2^2) \tag{8}$$

Where (x_d, y_d) defines a circular path:

$$\begin{aligned}
x_d &= r\sin(t) \\
y_d &= r(1 - \cos(t))
\end{aligned}$$

1.3.3 Constraints

Added practical constraints:

- Thrust limits: $0 \leq u_1, u_2 \leq 10$

1.4 Control Strategy

Used Sequential Quadratic Programming (SQP) to solve the optimization:

- Horizon length N = 250 steps
- Initial guess of constant thrust
- Iterative solution using SLSQP solver

1.5 Results

The controller successfully:

- Tracks the desired looping trajectory
- Maintains thrust constraints

The implementation strikes a good balance between performance and computational efficiency, making it suitable for real-time control.
Plots of all the states and controls of the robot as a function of time is provided in the Fig. 2 below for the trajectory optimization:

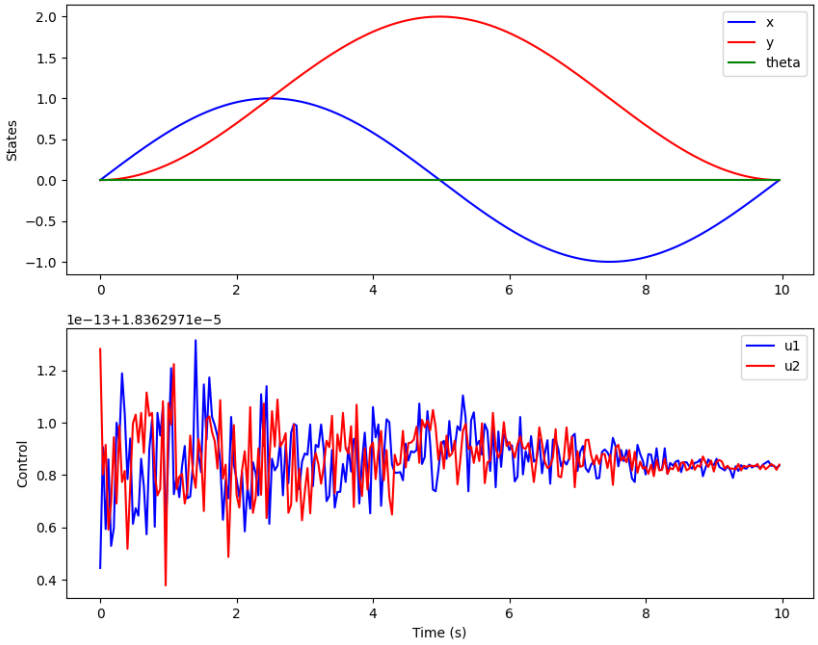

Figure 2: States and controls of the robot as a function of time

Part 2 - Model predictive control (MPC) (50 points)

1. Use the trajectory optimization method from Part I to design a MPC controller and test it using the simulator below. In particular, verify that it can handle perturbations by calling the `quadrotor.simulate` function with `disturbance = True` (when setting disturbance to True, the simulator will generate a random perturbation every 1 second). Simulate your controller for 10 seconds, plot the state and control evolution.
2. Explain your intended design in the report, including the cost function and found control law

The `quadrotor.simulate` function takes as an input an initial state, a controller, the number of discrete time steps and a boolean value to indicate the presence of perturbation. The controller has to be a function taking as an input a state and time index and outputting a control vector.

To visualize the trajectory, use the `quadrotor.animate_robot` function and show the animation (show the plots in your report).

2 MPC Implementation for Quadrotor Control

2.1 Overview

The code implements Model Predictive Control for a quadrotor to perform a looping maneuver while handling disturbances. The key components are:

2.2 System Dynamics

The `dyn` function implements the discretized quadrotor dynamics:

- Takes current state z and control inputs u
- State vector: $[x, \dot{x}, y, \dot{y}, \theta, \omega]$
- Control inputs: $[u_1, u_2]$ (rotor thrusts)
- Uses Euler integration with timestep dt

2.3 Cost Function

The `cost` function has two terms:

$$J = J_{track} + \lambda J_{ctrl} \tag{9}$$

where:

- $J_{track} = \sum_{k=0}^{N-1}[(x_k - x_d(k))^2 + (y_k - y_d(k))^2]$
- $J_{ctrl} = 0.01 \sum_{k=0}^{N-1}(u_{1,k}^2 + u_{2,k}^2)$
- Reference trajectory: $x_d = r\sin(t)$, $y_d = r(1 - \cos(t))$

2.4 Constraints

The `constraints` function enforces:

- Initial state constraint: $X_0 = z_0$
- System dynamics: $X_{k+1} = f(X_k, U_k)$
- Control bounds: $0 \leq u_1, u_2 \leq 10$

2.5 MPC Controller

The `mpc` function implements the receding horizon control:

1. Prediction horizon $N = 50$ steps
2. Solves optimization problem at each timestep
3. Uses SLSQP solver with constraints
4. Returns first control input from optimal sequence

2.6 Simulation

The controller is tested with:

- Initial state at origin: $z_0 = 0$
- 10 second simulation ($T = 10/dt$ steps)
- Random disturbances enabled
- Quadrotor animation for visualization

The MPC approach allows the quadrotor to track the desired looping trajectory while actively compensating for disturbances through continuous replanning.

Plots of all the states of the robot as a function of time is provided in the Fig. 3, and the control of the robot as a function of time is provided in Fig. 4 below for the Model Predictive Control:

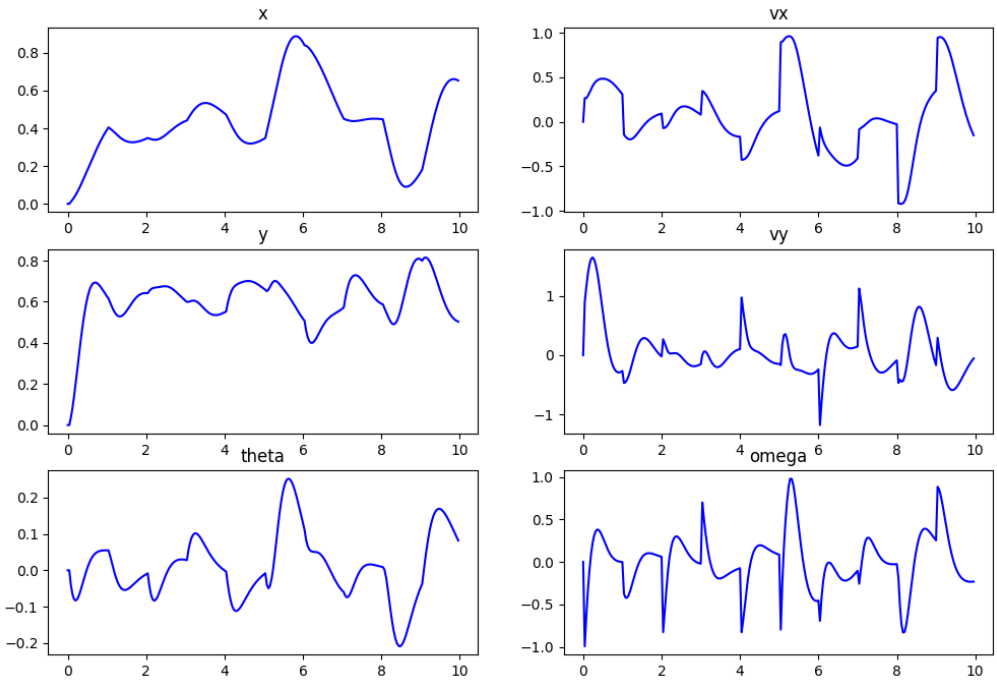

Figure 3: States of the robot as a function of time

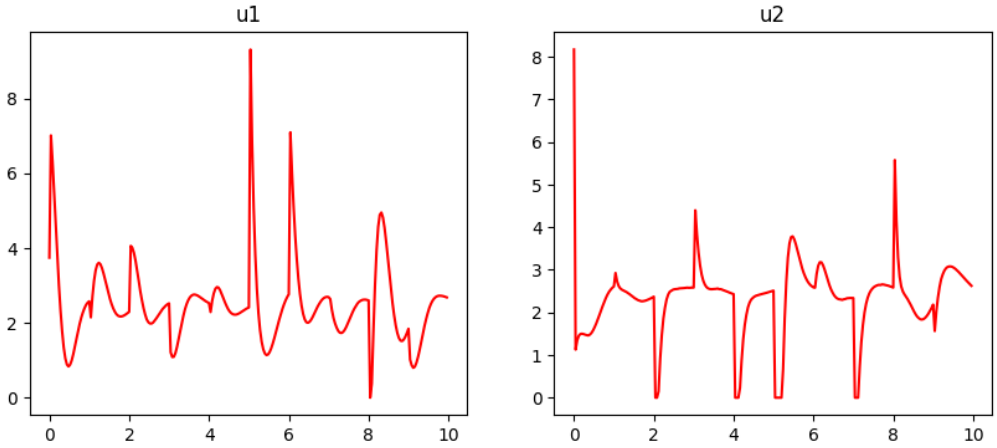

Figure 4: Controls of the robot as a function of time

Bonus (10 points)

Add a state constraint to perform the looping while maintaining a positive altitude. Use the origin as an initial state.

3 Quadrotor MPC Implementation with Altitude Constraints

3.1 System Dynamics

The `dyn` function implements the discretized quadrotor dynamics using Euler integration:

- Takes state vector $z = [x, \dot{x}, y, \dot{y}, \theta, \omega]$ and control inputs $u = [u_1, u_2]$
- Computes state derivatives:

$$\begin{aligned}
\dot{x} &= \dot{x} \\
\ddot{x} &= -\frac{u_1 + u_2}{m} \sin(\theta) \\
\dot{y} &= \dot{y} \\
\ddot{y} &= \frac{u_1 + u_2}{m} \cos(\theta) - g \\
\dot{\theta} &= \omega \\
\dot{\omega} &= \frac{L(u_1 - u_2)}{I}
\end{aligned}$$

- Returns next state using Euler step: $z_{k+1} = z_k + \dot{z}_k dt$

3.2 Cost Function

The `cost` function combines tracking and control effort:

$$J = J_{track} + \lambda J_{ctrl} \tag{10}$$

where:

- $J_{track} = \sum_{k=0}^{N-1} [(x_k - x_d(k))^2 + (y_k - y_d(k))^2]$
- $J_{ctrl} = 0.01 \sum_{k=0}^{N-1} (u_{1,k}^2 + u_{2,k}^2)$
- Reference trajectory is a circle: $x_d = r \sin(t)$, $y_d = r(1 - \cos(t))$

3.3 Constraints

The `constraints` function enforces three types of constraints:

1. Initial state: $X_0 = z_0$
2. System dynamics: $X_{k+1} = f(X_k, U_k)$ for $k = 0, ..., N-1$
3. Positive altitude: $y > 0$ (implemented as $-y \leq 0$)

3.4 MPC Controller

The `mpc` function implements receding horizon control:

- Solves optimization at each timestep:

$$\begin{aligned} \min_{X,U} \quad & J(X,U) \\ \text{s.t.} \quad & X_0 = z_0 \\ & X_{k+1} = f(X_k, U_k) \\ & y_k > 0 \\ & 0 \leq u_{1,k}, u_{2,k} \leq 10 \end{aligned}$$

- Uses SLSQP solver with 100 max iterations
- Returns first control input from optimal sequence

3.5 Simulation

The implementation:

- Starts from origin: $z_0 = 0$
- Simulates for 10 seconds with $dt = 0.04s$
- Includes random disturbances every 1 second
- Uses `quadrotor.simulate` for visualization

This MPC implementation ensures the quadrotor performs a looping maneuver while maintaining positive altitude and handling disturbances through continuous replanning.

Plots of all the states and controls of the robot as a function of time is provided in the Fig. 5 below for the trajectory optimization:

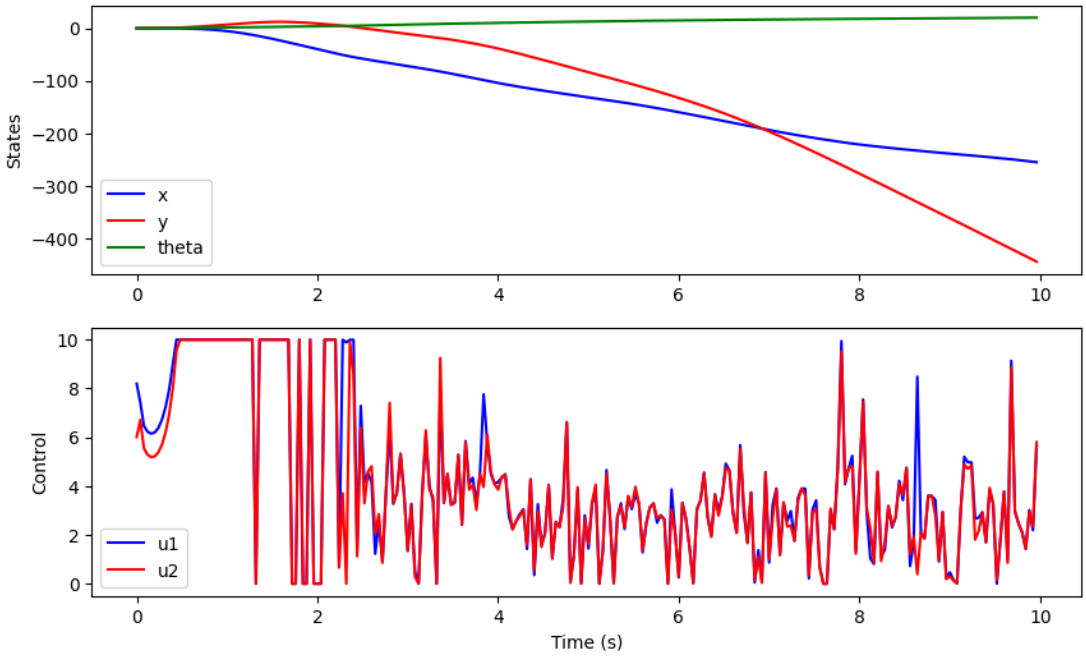

Figure 5: States and controls of the robot as a function of time