Reinforcement Learning and Optimal Control Project 1

Rishabh Verma

November 2024

1 Problem Statement

The goal of this project is to control a 2D quadrotor as shown in Fig. 1 to perform acrobatic moves. The controller will be designed using an SQP solver.

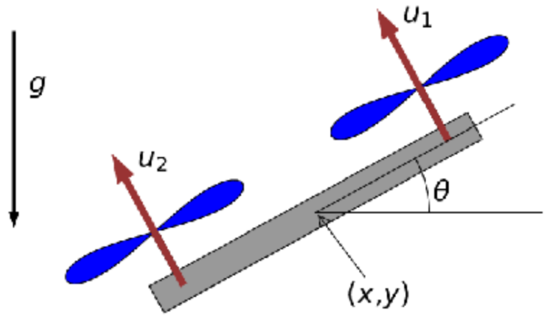


Figure 1: 2D Quadrotor Model

2 Model

2.1 Quadrotor Dynamics

The non-linear dynamics of the quadrotor are given by

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

First we discretize the dynamics over a time Δt

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

Next we need to do Taylor's Expansion about a small $\bar{x}$ such that we get equations in the form of $A(\bar{x})\Delta x = b(\bar{x})$ where Δx represents a small variation around $\bar{x}$. We would then stack the LHS and RHS into matrices A and b, and all the states x in a vector y that will collectively represent the dynamics of the quadrotor.

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

The above set of equations can be represented in the form of $A(y)\Delta y = b(y)$, where A holds the RHS of the equations and

$$y = \begin{bmatrix} p_{x_0} \\ v_{x_0} \\ p_{y_0} \\ y_{y_0} \\ \theta_0 \\ \omega_0 \\ u_{1_0} \\ u_{2_0} \\ p_{x_1} \\ v_{x_1} \\ p_{y_1} \\ \vdots \end{bmatrix}, b = \begin{bmatrix} O_{6x1} \\ p_{x_n} + \Delta t \cdot v_{x_n} - p_{x_{n+1}} \\ v_{x_n} - \Delta t\frac{(u_{1_n}+u_{2_n})\sin\theta_n}{m} - v_{x_{n+1}} \\ p_{y_n} + \Delta t \cdot v_{y_n} - p_{y_{n+1}} \\ v_{y_n} + \Delta t\left(\frac{(u_{1_n}+u_{2_n})\cos\theta_n}{m} - g\right) - v_{y_{n+1}} \\ \theta_n + \Delta t \cdot \omega_n - \theta_{n+1} \\ \omega_n + \Delta t\frac{L(u_{1_n}-u_{2_n})}{I} - \omega_{n+1} \\ \vdots \end{bmatrix}$$

Where O_{6x1} represents a 6x1 zero vector because our initial position is zero

2.2 Desired Trajectory

The desired trajectory is set as a form of an ellipse which repeats every 100 timesteps, where the drone will start and end at $(0, 0)$. The ellipse is defined using the parametric form $(a * cosine(t), b * sine(t)$ with the time period shifted to ensure that the starting position is at $(0, 0)$. The tilt θ of the drone is set as a linearly increasing where the orientation increases from 0 to 2π every 100 steps. The reason for having the theta not reset to zeros is because the solver took less time to compute on each iteration when the theta was not going to zero after every cycle and since every positive multiple of 2π would mean the quadrotor flips back in original position this works well.
Thus the trajectory obtained is

$$\begin{aligned}
x_{des} &= cos\left(-\tfrac{\pi}{2} + \tfrac{2\pi k}{N}\right), \\
y_{des} &= \frac{3}{2}\left(1 + sin\left(-\tfrac{\pi}{2} + \tfrac{2\pi k}{N}\right)\right), \\
\theta_{des} &= \frac{2\pi k}{100}
\end{aligned}$$

2.3 Cost Function

The motion of the quadrotor is governed by the trajectory that we need to follow so we can make our cost function [1] as

$$\frac{1}{2}\left(\sum_{k=0}^{N-1}(X_k - X_{k_des})^T Q(X_k - X_{k_des}) + U_n^T R U_n + (X_N - X_{N_des})^T Q(X_N - X_{N_des})\right)$$

Which penalizes deviation of the state from the desired state. This is equivalent to minimizing this cost (expanding and then ignoring constant terms, i.e. terms that do not depend on X_n)

$$\sum_{k=0}^{N-1}\left(\frac{1}{2}X_k^TQX_k+\frac{1}{2}U_n^TRU_n-X_{k_{des}}^TQX_k\right)+\frac{1}{2}X_k^TQX_k+X_{N_{des}}^TQX_N$$

The goal is to minimise this cost function, so we get,

$$\min_{X_k,U_k}\frac{1}{2}\left(\sum_{k=0}^{N-1}X_k^TQX_k+U_n^TRU_n-2X_{k_{des}}^TQX_k\right)+\frac{1}{2}X_k^TQX_k+X_{N_{des}}^TQX_N$$

The Cost Function then can be broken down in the form of

$$\min_y \frac{1}{2}y^TPy+q^Ty$$

where

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

$$q^T=\begin{bmatrix}-X_{0_{des}}^TQ&0&-X_{1_{des}}^TQ&0&\cdots\end{bmatrix}$$

This gives the gradient of cost function as

$$\nabla\mathcal{L}=Py+q$$

and the Hessian as

$$\nabla^2\mathcal{L}=P$$

2.4 Inequality Constraints

We have to apply the following constraints on the quadrotor

$$\begin{aligned}0\le u_1&\le 10\\0\le u_2&\le 10\\0&\le y\end{aligned}$$

This tentatively means [1]

$$\min_{X_k,U_k}\frac{1}{2}\left(\sum_{k=0}^{N-1}X_k^TQX_k+U_n^TRU_n-X_{k_{des}}^TQX_k\right)+\frac{1}{2}X_k^TQX_k+X_{N_{des}}^TQX_N$$

$$\begin{aligned}\text{Subject to}\quad X_{n+1}&=X_n+\Delta tf(X_n,U_n)\\X_0&=X_{init}\end{aligned}$$

For this I created a sparse matrix miniG where in each row corresponding to the position in the state vector y the value was 1 for upper limit, -1 for lower limit and 0 for no limit. This matrix is then propogated along the diagonal. Because this was a non-linear system we need to linearize it and we get h vector as a column vector with the limits stacked one on top of another.

$$\text{miniG}=\begin{bmatrix}0&0&0&0&0&0&0&0\\0&0&-1&0&0&0&0&0\\0&0&0&0&0&0&1&0\\0&0&0&0&0&0&-1&0\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&0&-1\end{bmatrix}\qquad h=\begin{bmatrix}0\\p_y\\10-u_1\\u_1\\10-u_2\\u_2\\\vdots\end{bmatrix}$$

3 Solving the KKT Problem and Filter Line Search

The MPC controller is implemented by simply calling the line_search function for the length of the horizon then recalculating the optimal trajectory based on the cost and violations. Rest of the process will remain same.

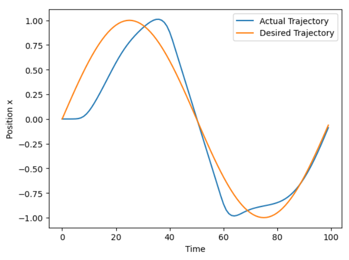
At each time step, the optimal control signal $\mathbf{u_i}_t$ is computed by solving a constrained optimization problem over a finite time horizon $N = 50$, aiming to minimize a cost function J that reflects the trajectory tracking error and control effort, while adhering to system dynamics and constraints.

The reason for chosing this timestep was that N = 50 was computationally the sweet spot for the balance between accuracy and computationtime. Value of N=100 gave each iteration 17 sec to calculate, which I deemed too big a tradeoff on time for accuracy.

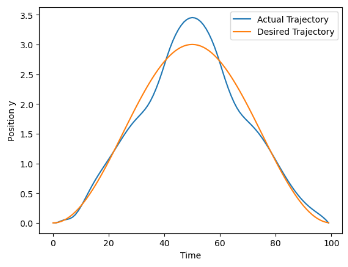
At every function call of controller the algorithm optimizes the trajectory for the next 50 steps and keeps sliding till we reach the end of horizon. Also this system has been tested in case of disturbances by setting the flag disturbance = True in the simulator.

4 Results

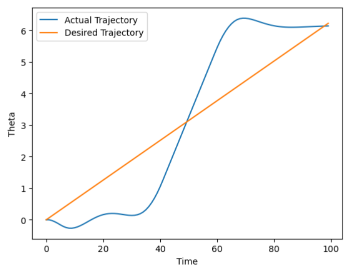
4.1 Trajectory Optimization



(a) X Tracking



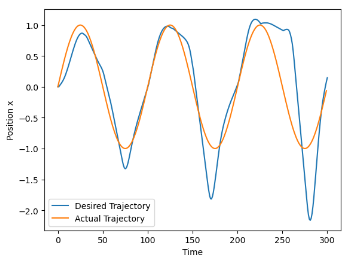
(b) Y Tracking



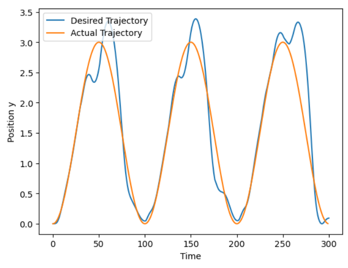
(c) θ Tracking

Figure 2: Results from Trajectory Optimization for X, Y, and θ.

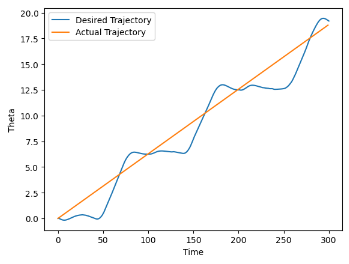
4.2 Model Predictive Control



(a) X Tracking



(b) Y Tracking



(c) θ Tracking

Figure 3: Results from MPC for X, Y, and θ.

References

[1] Ludovic Righetti and Contributors. *Lecture 2 - LQ problems with KKT and QP*. Accessed: 2024-11-24. 2024. URL: `https://github.com/righetti/optlearningcontrol/blob/main/Examples/Lecture%202%20-%20LQ%20problems%20with%20KKT%20and%20QP.ipynb`.