Iterative LQR to control a drone

jh7454

November 2022

1 Introduction

The goal of this project is to control a 2D quadrotor to perform acrobatic moves.

Firstly, Section 1.1 gives an introdution to the drone dynamics. Section 2 involves discretizing the system dynamics and making the drone hover at the origin. Section 3 is implementation of LQR to keep the drone stable even during disturbances. Section 4 is implementation of LQR to follow a desired trajectory. Lastly, Section 5 is the implementation of iLQR to make the drone perform some acrobatic moves.

The final Section 6 gives the conclusion and talk a bit about improvements.

1.1 Drone dynamics

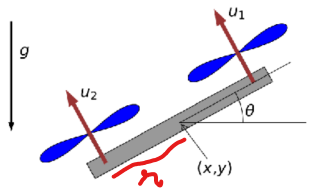


Figure 1: 2D quadrotor design

Figure 1 shows the 2D quadrotor, using this the dynamics can be written as:

$$\dot{x} = V_x, \tag{1a}$$
$$m\dot{V}_x = -(u_1 + u_2)\sin(\theta), \tag{1b}$$
$$\dot{y} = V_y, \tag{1c}$$
$$m\dot{V}_y = (u_1 + u_2)\cos(\theta) - mg, \tag{1d}$$
$$\dot{\theta} = \omega, \tag{1e}$$
$$I\dot{w} = r(u_1 - u_2) \tag{1f}$$

(1a), (1b), (1c), (1d), (1e) and (1f), can be used to find the state matrix and control matrix:

$$z = \begin{bmatrix} x & V_x & y & V_y & \theta & \omega \end{bmatrix}^\top, \tag{2a}$$

$$u = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^\top \tag{2b}$$

Constants m, g, I, r in (1b), (1d) and (1f), are $0.6, 9.81, 0.15, 0.2$, respectively. Tests are performed for

$$N = 1000 \tag{3}$$

Where 3 is the horizon length.

2 Part 1 - Setting up

Section 2.1 are the stages invloved in Discretizing the system dynamics, using the 4 equations u_1^* and u_2^* are computed in Section 2.2. Section 2.3 and Section 2.4 are attempts to control the drone to follow simple trajectories.

2.1 Discretizing the system dynamics

Using $\Delta t = 0.01$ we can discretize the system dynamics as:

$$x_{n+1} = x_n + \Delta t V_{x_n}, \tag{4a}$$

$$V_{x_{n+1}} = V_{x_n} + \Delta t \left(\frac{-(u_1 + u_2)\sin(\theta)}{m} \right), \tag{4b}$$

$$y_{n+1} = y_n + \Delta t V_{y_n}, \tag{4c}$$

$$V_{y_{n+1}} = V_{y_n} + \Delta t \left(\frac{(u_1 + u_2)\cos(\theta) - g}{m} \right), \tag{4d}$$

$$\theta_{n+1} = \theta_n + \Delta t \omega_n, \tag{4e}$$

$$\omega_{n+1} = \omega_n + \Delta t \left(\frac{r(u_1 + u_2)}{I} \right) \tag{4f}$$

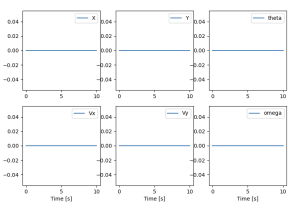
(4a), (4b), (4c), (4d), (4e), and (4f), represents each state after discretization.

2.2 Computing u_1^* and u_2^*

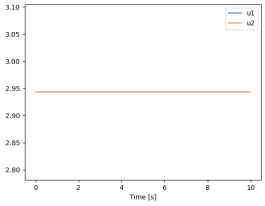
When (2a) is $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\top$ we can compute u_1^* and u_2^* using (4a), (4b), (4c), (4d), (4e), and (4f). By solving for u_1^* and u_2^* we get:

$$\begin{bmatrix} u_1^* & u_2^* \end{bmatrix}^\top = \begin{bmatrix} \frac{mg}{2} & \frac{mg}{2} \end{bmatrix}^\top \tag{5}$$

(5) will be used in the following stages, by simulating the drone, Figure 2 shows the drone perfectly at the origin.



(a) Drone states(2a)

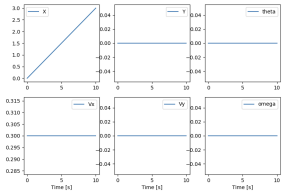


(b) Drone controls(2b)

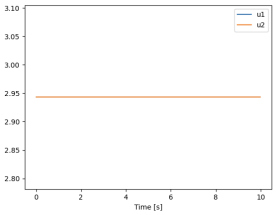
Figure 2: States and control for (3) using (5)

2.3 Moving in x direction while keeping $\theta = 0$

It is possible to move in x direction when $\theta = 0$, when the initial (2a) at stage 0 or z_0 is $\begin{bmatrix} 0 & 0.3 & 0 & 0 & 0 & 0 \end{bmatrix}^\top$ *i.e.*, $V_x = 0.3$. By simulating the drone, the drone moves in the x direction keeping $V_x = 0.3$ as shown in Figure 3.



(a) Drone states(2a)



(b) Drone controls(2b)

Figure 3: States and control for (3) using (5) and setting $V_x = 0.3$

2.4 Keeping drone at rest with $\theta = \frac{\pi}{2}$

The system becomes unstable at $\theta = \frac{\pi}{2}$ so it is not possible to keep at rest in this orientation.

3 Part 2 - LQR to stay in place

To control the drone even under disturbances, Section 3.1 linearizes the dynamics and Section 3.2 shows how the drone performs during disturbances.

3.1 Linearizing the dynamics

Using Taylor's series first order approximation we linearize the dynamics around an arbitrary point z^* and u^* to obtain using (4a), (4b), (4c), (4d), (4e), (4f):

$$A = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \frac{-(u_1+u_2)\Delta t\cos(\theta)}{m} & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 1 & \frac{-(u_1+u_2)\Delta t\sin(\theta)}{m} & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{6a}$$

$$B = \begin{bmatrix} 0 & 0 \\ \frac{-\sin(\theta)\Delta t}{m} & \frac{-\sin(\theta)\Delta t}{m} \\ 0 & 0 \\ \frac{\cos(\theta)\Delta t}{m} & \frac{\cos(\theta)\Delta t}{m} \\ 0 & 0 \\ \frac{r\Delta t}{I} & \frac{-r\Delta t}{I} \end{bmatrix}, \tag{6b}$$

$$\bar{z}_n = z_n - z^*, \tag{6c}$$

$$\bar{u}_n = u_n - u^* \tag{6d}$$

The linear system is:

$$\bar{z}_{n+1} = A\bar{z}_n + B\bar{u}_n \tag{7}$$

The A and B from (6a) and (6b) respectively can be used the solve the backward Riccati equation around (6c) and (6d) for the system (7):

$$\bar{u}_n = K_n\bar{z}_n, \tag{8a}$$

$$u_n = K_n(z_n - z^*) + u^* \tag{8b}$$

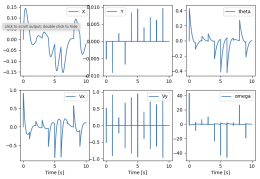
8a is the solution, by rearranging a few variables the original systems solution is obtained 8b. K_n and u^* represent the gain and 5 respectively in 8b.

3.2 Testing hovering at origin during disturbances

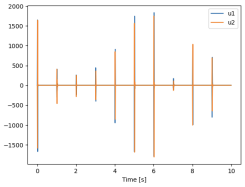
Using the control 8b the drone is able to stay very close to the origin even during disturbances. Figure 4 shows the graph 4a having x, y always very close to the origin even when disturbances are large. Graph 4b shows large control values produced to stabilize during disturbances.

4 Part 3 - Following a trajectory using linearized dynamics

The drone needs to follow a circular trajectory, Section 4.1 shows how to generate the desired trajectory. Section 4.2 describes how the designed controller performs with and without disturbances. The Figures 5 and 6 are analyzed in Section 4.3. Section 4.4 explains tracking under a certain condition.



(a) Drone states(2a)



(b) Drone controls(2b)

Figure 4: States and control for (3) using (8b) with disturbances

4.1 Trajectory generation

To make the drone follow circular trajectory having a radius of 1 centered at $(0, 0)$, the circle must be in its parametric form.

$$x = r\cos(\theta), \tag{9a}$$
$$y = r\sin(\theta) \tag{9b}$$

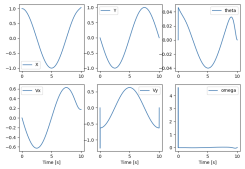
Using the parametric equation of a circle 9a, 9b, where $r = 1$ the (x, y) coordinates for 3 can obtained, this is the desired trajectory to follow.

4.2 Designing controller and tracking the trajectory

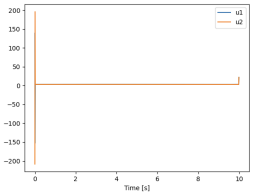
Using 6a and 6b we can solve the LQR trajectory for the system 7:

$$\bar{u}_n = K_n\bar{z}_n + k_n, \tag{10a}$$
$$u_n = K_n(z_n - z^*) + k_n + u^* \tag{10b}$$

Similar to 8b there is an additional term k_n in 10b which denotes the feed forward term. Simulating this controller, we obtain Figure 5 when simulated without disturbances and Figure 6 during disturbances.

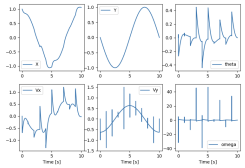


(a) Drone states(2a)

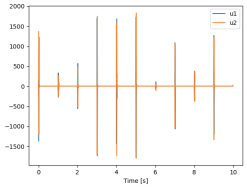


(b) Drone controls(2b)

Figure 5: States and control for (3) using (10b) without disturbances



(a) Drone states(2a)



(b) Drone controls(2b)

Figure 6: States and control for (3) using (10b) with disturbances

4.3 Analyzing the results

The graph 5a and 5b in Figure 5 shows the state sequence 2a and control sequence 2b respectively without disturbances, the designed controller is performing well as it is able track (x, y, θ) very closely. It is not possible to keep θ at 0 as then the drone will not be able to move, hence there is small change in θ.

Similarly, the graph 6a and 6b in Figure 6 shows the state sequence 2a and control sequence 2b respectively during disturbances. The controller does not have trouble tracking under disturbances and quick to go back to desired trajectory, by setting a high state penalty(Q matrix) and quick response (R matrix) the desired results are obtained.

4.4 Tracking with $\theta = \frac{\pi}{4}$

It is not possible to track when $\theta = \frac{\pi}{4}$ as the drone needs to keep varying θ at each quadrant to move in a complete circle.

5 Part 4 - iterative LQR

This section is divided into two parts where Section 5.1 is making the drone reach a vertical orientation and Section 5.2 is performing a full flip. Section 5.3 lists the pro and cons of iLQR.

5.1 Task 1 - reaching a vertical orientation

The steps to reach vertical orientation is, firstly, finding a time-varying cost function Section 5.1.1, then computing the quadratic approximation of the cost in Section 5.1.2. Section 5.1.3 shows the implementation of iLQR and Section 5.1.4 shows the line search algorithm. The results of the task are shown in Section 5.1.5.

5.1.1 Time-varying cost function

By taking a simple quadratic cost in the form:

$$J = \sum_{n=0}^{N} \bar{z}_n^\top Q_n \bar{z}_n + \bar{u}_n^\top R_n \bar{u}_n \tag{11}$$

5.1.2 Quadratic approximation of cost

By approximating the cost using second order Taylor's series second order approximation for 11.

$$J = \sum_{n=0}^{N} q_n^\top \bar{z}_n + \frac{1}{2} \bar{z}_n^\top Q_n \bar{z}_n + r_n^\top \bar{u}_n + \frac{1}{2} \bar{u}_n^\top R_n \bar{u}_n \tag{12}$$

Where q_n and r_n are the gradient vectors and Q_n and R_n are the Hessian's in 12. There was no terminal cost taken, $\bar{z}_n$ and $\bar{u}_n$ are given in 6c and 6d.

$$\bar{z}_n = z_n - z^*, \tag{6c}$$
$$\bar{u}_n = u_n - u^* \tag{6d}$$

5.1.3 iLQR algorithim

Algorithm 1 iLQR

1: $u_0^*, \ldots, u_{N-1}^* \leftarrow$ `initial guesses`
2: $x_0^*, \ldots, x_{N-1}^* \leftarrow$ `integrate with control guesses`
3: Linearize dynamics and get quadratic approximation of cost, 6 12
4: $u_n = K_n(z_n - z^*) + k_n + u^* \leftarrow$ `solving backward Riccati equation`, 10b, ▷ Line Search(5.1.4)
5: back to (2) until convergence

1 gives an overview of the algorithm, it can be used to solve an optimization problem typically the non-linear least squares problem.

5.1.4 Line search

There is a possibility that the new controller does not improve the cost, as often the case in optimization problems. To overcome this limitation a line search was performed to minimize the cost further.

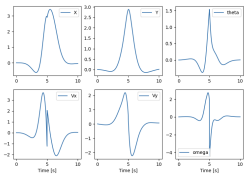
Algorithm 2 Line search

1: Start with $\alpha = 1$
2: $u_n = K_n(z_n - z^*) + \alpha k_n + u^* \leftarrow$ `obtain control sequences`
3: Integrate with nonlinear dynamics
4: If cost decreases stop, else decrease α
5: Repeat

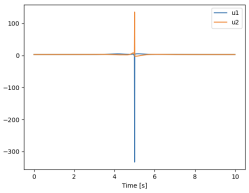
5.1.5 Reaching vertical orientation

Using both the algorithims 1 and 2 the drone was able to reach the orientation at $t = 5$, the state penalty (Q_i) and response (R_i) had to be changed at certain stages, in turn changing the gradient vectors and Hessian's 12 to obtain the desired trajectory.

The desired trajectory was obtained as shown in Figure 7's graph 7a. Analysing



(a) Drone states(2a)



(b) Drone controls(2b)

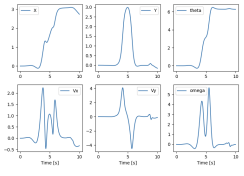
Figure 7: States and control for (3) using Algorithm 1 and Algorithm 2

the Figure 7 carefully the drone reaches the desired x position but slightly undershoots y position at $t = 5$. The $L1$ error is less than 0.02 so it is acceptable, it has no problem reaching $\theta = \frac{\pi}{2}$ while staying close to the origin for the rest of the duration's.

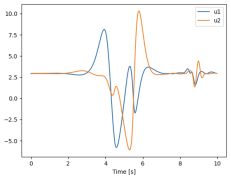
The time complexity can be reduced if the approximation is close to the original value, though difficult to find, if done properly can improve both the accuracy and time complexity of the entire algorithm.

5.2 Task 2 - doing a full flip

A new time varying cost function was created using 3 desired points. The state penalty (Q_i) and response(R_i) had to be tuned in (11) and (12) perform a complex operation such as flipping.



(a) Drone states(2a)



(b) Drone controls(2b)

Figure 8: States and control for (3) using Algorithm 1 and Algorithm 2

The Figure 8 has the graph 8a and graph 8b which represent the state sequences and control sequences. It can clearly be seen that the drone is able to go the desired points at $t = 5$ and $t = 10$.

5.3 Pros and cons

The iLQR Algorithm 1 is computationally very expensive, and when coupled with line search Algorithm 2 becomes more expensive. This is to be expected as most non-linear optimization problems are computationally expensive. If it has to be used in a real robot the time complexity of the algorithm has to taken into account as a slower algorithm can make the robot not perform efficiently.

6 Conclusion

All the tasks were completed successfully, all the tasks had a major factor common no matter what algorithm was used to compute the optimal controls, that common factor was tuning the state penalty (Q_n) and response (R_n).

Perhaps neural networks can be used to tune the these matrix to make the process autonomous. A very basic method to implement a network is to make fully connected layers and have targets to the desired trajectory. Another minimization can be performed at each time step in the horizon using a L2 loss and a backprogation. This type of network can be used to find the perfect values for Q_n and R_n.