Iterative LQR to control a drone

-by Pavan Chowdary Cherukuri pc3088

Part 1: Setting Up

Qsn 1: Initially the system dynamics should be discretized. It is considered that each second is discretized into 1000 steps. By using the system dynamics, discretization can be performed. For any variable to be discretized, the value of its derivative at that particular point is required. Discretization of the system dynamics is performed so that the value of any variable at time step n+1 can be found from the value of that variable at time step n.

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

The discretized equations are:

$$\begin{aligned}
x_{n+1} &= x_n + v_x * \Delta t \\
v_{x\,n+1} &= v_{x\,n} - (u1 + u2) * sin(\theta_n) * \Delta t/m \\
y_{n+1} &= y_n + v_y * \Delta t \\
v_{y\,n+1} &= v_{y\,n} + (((u1 + u2) * sin(\theta_n)) - (m * g)) * \Delta t/m \\
\theta_{n+1} &= \theta_n + \omega * \Delta t \\
\omega_{n+1} &= \omega_n + (r * (u1 - u2) * \Delta t)/I
\end{aligned}$$

Qsn 2: In this Part, it is given that the required state of the quadrotor is $[0, 0, 0, 0, 0, 0]T$.
It is required to compute u1* and u2* such that the quadrotor stays in this position forever. Since it needs to be at rest, as per equation (6) in the system dynamics angular acceleration will be equal to zero making u1=u2. Since θ = 0, as per equation (4) u1+u2 will be equal to MASS*GRAVITY. By this, u1* = u2* = (MASS*GRAVITY)/2 was obtained for the quadrotor to stay in the required state, which is rest.

Qsn 3: Analyzing the system dynamics, it is not possible to move in the x direction while keeping θ = 0. If we take a look at system dynamics.

In equation 2, the acceleration in the x direction is defined as -(u1+u2)*sin(θ). If θ = 0, then the acceleration in x, will always be zero, which implies irrespective of the u1 and u2, the system cannot move in the x-direction.

Qsn 4: If θ = π/2, from equation 4, acceleration in y direction is going to be "-g" irrespective of the values of u1 and u2. So the acceleration in y direction is inevitable, hence it is impossible to stay in rest when θ = π/2.

Part 2: LQR to stay in place

Qsn 1: For this part of the project, the goal is to design an LQR controller that keeps the robot at a predefined state. Here, we are considering that state to be at the origin, for which the optimal control u* was found in Part 1 of the project. So for that the dynamics of the system should be linearized at every time step. Mathematically, to linearize around the point to stabilize x* with control u*, the equations can be given as shown:

$$x_{t+1} \simeq \underbrace{f(x^*, u^*)}_{x^*} + \underbrace{\left[\frac{\partial f}{\partial x}\Big|_{x_t=x^*, u_t=u^*}\right]}_{A} \underbrace{(x_t - x^*)}_{\bar{x}_t} + \underbrace{\left[\frac{\partial f}{\partial u}\Big|_{x_t=x^*, u_t=u^*}\right]}_{B} \underbrace{(u_t - u^*)}_{\bar{u}_t}$$

So as per this, the Jacobians of the function at $x_t = x^*$ and $u_t = u^*$ should be found. For this "*sympy*" python library has been used. The "*jacobian()*" function has been used to calculate the Jacobians. This could also be done by hand. At a particular time step $x_t = x^*$ and $u_t = u^*$, let, $\theta = x^*[4]$ (fifth element in the array); m = MASS; dt = DELTA_T(single time step), $u1 = u^*[0]$ (first element in the array) and $u2 = u^*[1]$ (second element in the array).

The resulting Jacobian matrices are given by equations:

$A = \partial f/\partial x$ = $[[1, dt, 0, 0, 0, 0], [0, 1, 0, 0, -(dt * (u1 + u2) * cos\theta)/m, 0], [0, 0, 1, dt, 0, 0],$
$[0, 0, 0, 1, -(dt * (u1 + u2) * sin\theta)/m, 0], [0, 0, 0, 0, 1, dt], [0, 0, 0, 0, 0, 1]]$

$B = \partial f/\partial u$ = $[[0, 0], [(-dt * sin\theta)/m, (-dt * sin\theta)/m], [0, 0],$
$[(dt * cos\theta)/m, (dt * cos\theta)/m], [0, 0], [(dt * r/I), -(dt * r/I)]]$

Through this process the values of matrices A and B can be obtained.

Qsn2: All the above mentioned processes are being performed in the "*get_linearization(z,u)"* function. It takes state 'z' and controls 'u' as inputs at a particular time step and returns the Jacobian matrices 'A' and 'B'.

Qsn 3.4.5: In these parts an infinite horizon LQR that stabilizes the origin needs to be designed and tested with and without perturbations. In the design part of the infinite horizon LQR, the cost function that needs to be minimized is given as:

$$\min_{\bar{u}_0,\cdots,\bar{u}_{N-1}} \left(\sum_{k=0}^{N-1} \bar{x}_k^T Q \bar{x}_k + \bar{u}_k^T R \bar{u}_k \right) + \bar{x}_N^T Q \bar{x}_N$$

Where,

$$\bar{x}_n = x_n - x^*$$

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

Subject to the dynamics:

$$\bar{x}_{n+1} = A\bar{x}_n + B\bar{u}_n$$

As discussed above, the dynamics were linearized at every time step and the 'A' and 'B' matrices represent the Jacobians. The LQR solution/ optimal policy is be given by:

$$\bar{u}_n = K_n \bar{x}_n$$

$$u_n = K_n(x_n - x^*) + u^*$$

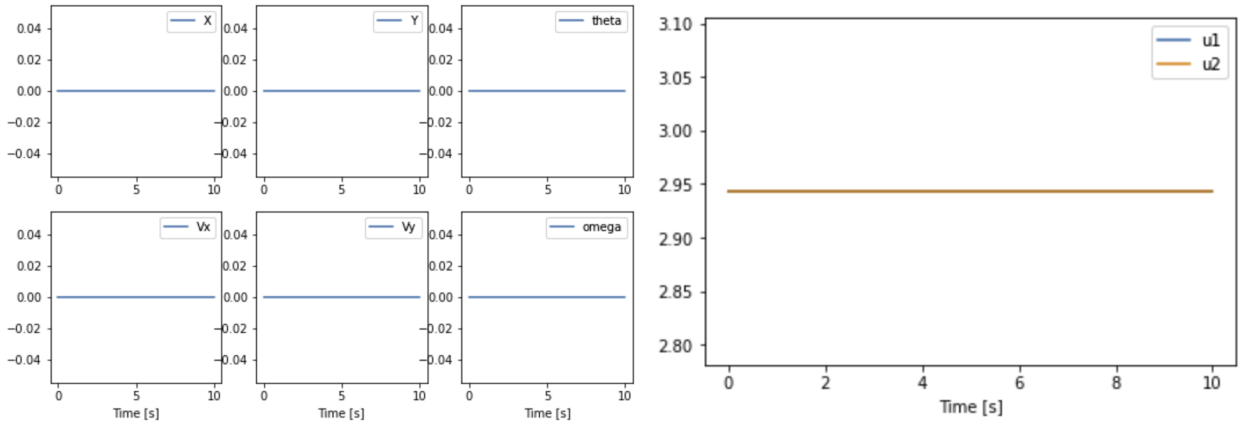
Where '*Kn*' is the gain. This can be solved by backward recursion (from N-1 to 0) of the Discrete-time Riccati equations:

$$P_N = Q_N$$

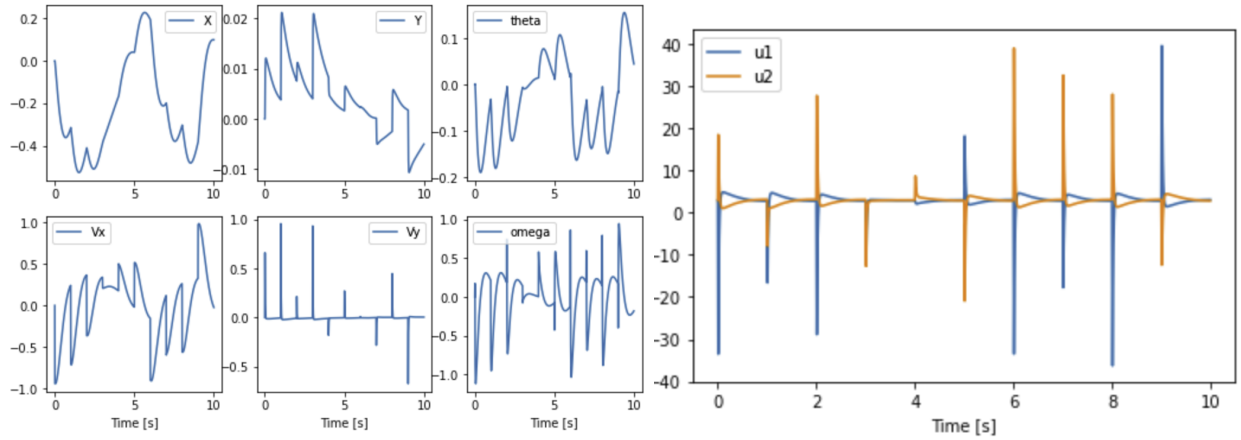
$$K_n = -(B_n^T P_{n+1} B_n + R_n)^{-1} B_n^T P_{n+1} A_n$$

$$P_n = Q_n + A_n^T P_{n+1} A_n + A_n^T P_{n+1} B_n K_n$$

The quadrotor is simulated with and without disturbances. The plots when simulated without disturbances is are as follows:



The plots when simulated with disturbances is are as follows:



Part 3: Following a trajectory using linearized dynamics

Qsn 1.2: In this part of the project, it is required to follow a given trajectory leveraging a linearized version of the dynamics to design LQ controllers. For this if we have a desired trajectory, then we can find the control trajectory.
Assuming if we have a desired trajectory sequence: $x_1^*, x_2^*, \ldots . \ x_N^*$. Corresponding control sequence can be found and let it be: $u_1^*, u_2^*, \ldots . \ u_N^*$.
For this problem, the cost function that is needed to be minimized is given by:

$$\min_{u_0,\cdots,u_{N-1}} \left(\sum_{k=0}^{N-1} (x_k - x_k^*)^T Q (x_k - x_k^*) + (u_k - u_k^*)^T R (u_k - u_k^*) \right) + (x_N - x_N^*)^T Q (x_N - x_N^*)$$

Now, at each n, the dynamics are being linearized and the values of 'A' and 'B' matrices are found at each time step. The note to be taken here is that the *x** and *u** are also varying with time steps unlike in the previous case. So, the A and B matrices will not be the same in this case as like in the previous parts. The Linear quadratic solutions for the tracking problem can be given by:

$$\bar{u}_t = K_t \bar{x}_t + k_t$$

$$u_t = K_t(x_t - x_t^*) + k_t + u_t^*$$

Where, Kt and kt are the controller gains and feed forward terms. This can be solved by backward Riccati equations:

$$P_N = Q_N \qquad p_N = q_N$$

$$K_n = -(R_n + B_n^T P_{n+1} B_n)^{-1} B_n^T P_{n+1} A_n$$
$$P_n = Q_n + A_n^T P_{n+1} A_n + A_n^T P_{n+1} B_n K_n$$
$$k_n = -(R_n + B_n^T P_{n+1} B_n)^{-1} B_n^T p_{n+1}$$
$$p_n = q_n + A_n^T p_{n+1} + A_n^T P_{n+1} B_n k_n$$

In Qsn 1 it is asked that the trajectory required to be followed is going to be a circle with a center at origin and a radius 1.

The equation of the circle is given by: $x^2 + y^2 = r^2$

In the parametric form the state can be given by: $[r * cos\alpha,\ r * sin\alpha]$

The velocities in the 'x' and 'y' axes can be given by: $[r * -sin\alpha * \partial\alpha/\partial t,\ r * cos\alpha * \partial\alpha/\partial t]$

Since the state is discretized (in our case for a 1000 steps), $\alpha_{t+1} = \alpha_t + 2 * \Pi/1000$

Hence, $\partial\alpha/\partial t = 2 * \Pi/1000$

Hence, this makes our state vector (z*) at time 't' as:

$[r * cos\alpha,\ r * -sin\alpha * 2 * \Pi/1000,\ r * sin\alpha,\ r * cos\alpha * 2 * \Pi/1000,\ \theta,\ \omega]$

In Qsn 1 it is mentioned that θ = 0, considering this the values of desired control trajectory could also be found. Substituting the above found state values in the system dynamics, control vector can be give by:

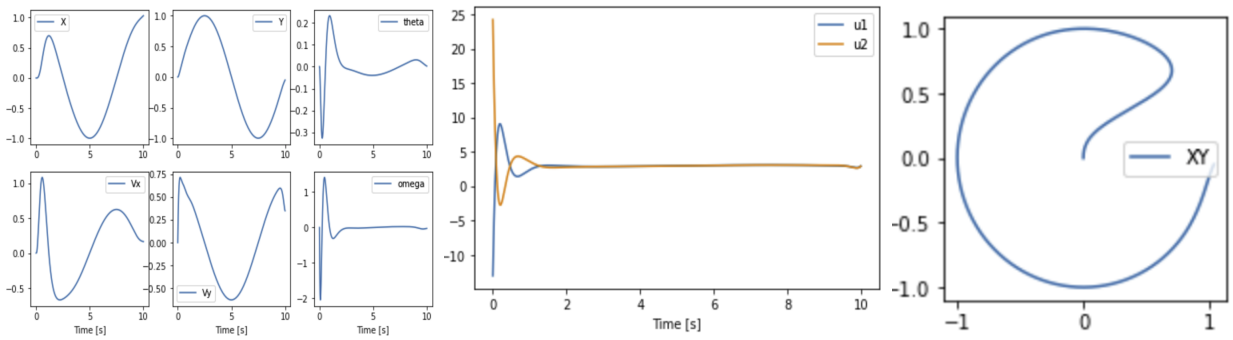
$[(m * g - m * r * sin\alpha * (2\Pi/1000) * (2\Pi/1000))/2,$
$(m * g - m * r * sin\alpha * (2\Pi/1000) * (2\Pi/1000))/2]$

It could also be simply considered as: $[(m * g)/2,\ (m * g)/2]$

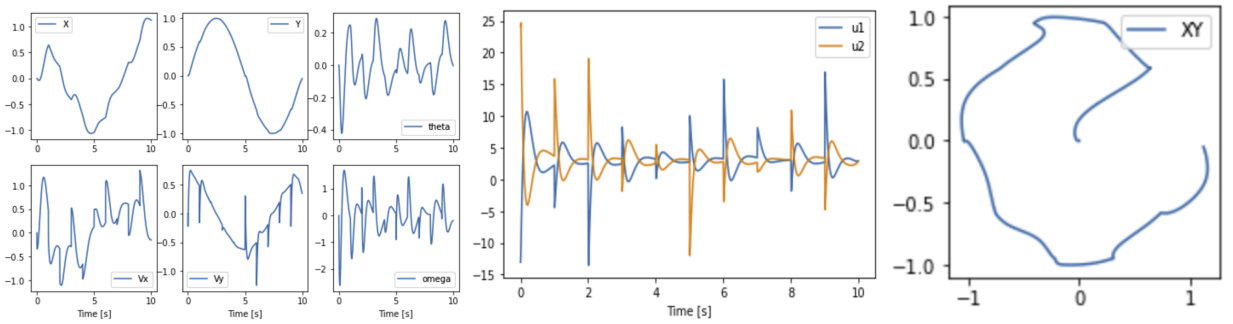
Since m*g/2 is required to keep the quadrotor at θ = 0. While tuning the controllers the cost for the control is quite less, hence this minor change doesn't affect the results. Both these results were tested and there is not much of a noticeable difference.

Qsn 3: With the above found equation, the simulation was performed. Tracking θ was not as good as tracking the x-y trajectory, since for the robot to move in a circle, it should be changing its θ values, hence θ can't always be zero as desired. But the θ will always be close to zero and doesn't deviate much.

Qsn 4: After simulation, the plots were visualized. Initially the results without any disturbances were visualized. The x-y plot has also been added for better visualization purposes.



Results with disturbances:



From the plots, it can be understood that when there were no disturbances, the robot was able successfully track the given circle trajectory, this could be clearly visualized from the X-Y graph as well. Even with disturbances, it was trying to follow the given trajectory, which can be good in real time situations. The issues are that if the control is not tuned properly for this controller, the robot can go outside and the cost might explode, which happens in some cases. Also, since the dynamics is getting linearized at every point of time, this might be an issue as there might be errors arising from this.

Qsn 5: At θ = 45, when the robot is simulated, it tries to follow the circular trajectory as well as maintain θ = 45, which is not possible both at the same time. If the robot is animated for this, it can be seen that it initially tries to reach θ = 45, but when it starts following the circle, the theta again comes close to zero. This occurs due to the conflict in the trajectory and the system dynamics.

Part 4: Iterative LQR

The tasks for this part of the project is to make the robot do complicated maneuvers like a flip, making a 90 degree orientation etc. For this iterative LQR is going to be used. While optimizing the controller we find a locally optimal trajectory. Unlike in the third part we do not have a reference trajectory that we need to follow. Here we will be having the desired states, the controller should be able to compute the local trajectory and also the trajectory should be optimized to decrease the cost. So, for iterative LQR the idea starts with a goal to minimize a cost function which is arbitrary and needs to be computed. Let, the cost function be:

$$\min_{u_0,\cdots,u_{N-1}} \sum_{k=0}^{N-1} l_k(x_k, u_k) + l_N(x_N)$$

The dynamics are non-linear as we know. So, we first start with an initial control trajectory guess (say all zeros) and then we obtain the trajectory corresponding to that control trajectory.
For that guess, the dynamics are linearized and a quadratic approximation of the cost is obtained. Then the backward Riccati equations are solved to get a new control trajectory and state vectors. The cost is calculated for this new trajectory and the process is looped until convergence. The loop should be stopped due to various conditions, how and why will be discussed in the coming sections.

Task 1:

Qsn 1: For the problem to get started, the arbitrary cost function needs to be determined. For task 1, the robot must reach x=3, y=3, θ=pi/2 at time t=5 seconds and it should stay close to origin for the rest of the time. Hence the cost function should be written accordingly. So, here the cost function has been divided into 3 different intervals, t= 0 to 4 seconds, t = 4 to 5 seconds, and the rest of the time.

If $x_0 = [0,\ 0,\ 0,\ 0,\ 0,\ 0]^T$ and $u_0 = [(m * g)/2,\ (m * g)/2]^T$ and

$x_5 = [3,\ 0,\ 3,\ 0,\ pi/2,\ 0]^T$ and $u_5 = [(m * g)/2,\ (m * g)/2]^T$, then

Cost function at time step 't' for t = 0 to 4 seconds is given by:

$$0.5 * (x_t - x_0)^T * Q1 * (x_t - x_0) + 0.5 * (u_t - u_0)^T * R1 * (u_t - u_0)$$

Cost function at time step 't' for t = 4 to 5 seconds is given by:

$$0.5 * (x_t - x_5)^T * Q2 * (x_t - x_5) + 0.5 * (u_t - u_5)^T * R2 * (u_t - u_5)$$

Cost function at time step 't' for t = 5 to 9.99 seconds is given by:

$$0.5 * (x_t - x_0)^T * Q1 * (x_t - x_0) + 0.5 * (u_t - u_0)^T * R1 * (u_t - u_0)$$

Cost function at time step 't' for t = 10 seconds is given by (terminal cost):

$$0.5 * (x_t - x_0)^T * Q1 * (x_t - x_0)$$

The values *Q1, Q2, R1, R2* should be tuned accordingly.

Qsn 2: A *compute_cost(z,u,horizon_length)* function has been written to compute the cost for any arbitrary trajectory z with control trajectory u by using the above written cost equations.

Qsn 3: The quadratic approximation of the cost function is also written. This is of form:

$$J \simeq \text{const} + q_N^T \bar{x}_N + \frac{1}{2}\bar{x}_N Q_N \bar{x} + \sum_{t=0}^{N-1} q_n^T \bar{x}_n + \frac{1}{2}\bar{x}_n^T Q_n \bar{x}_n + r_n^T \bar{u}_n + \frac{1}{2}\bar{u}_n^T R_n \bar{u}_t$$

By taking the approximation of the cost function written in Qsn 1, the values of q_n and r_n were found as: $q_n = 2 * Q * \overline{x_n}$ and $r_n = 2 * R * \overline{u_n}$

Q and R values vary per time step as they are time varying costs.

Qsn 4: A *get_quadratic_approximation(z, u, horizon_length)* function was made to return the "*q, r, Q, R"* values. Since these all are time varying they are returned in different lists so that they can be indexed later easily.

Qsn 5: After the dynamics are linearized as done in part 3 and the quadratic approximation of the cost is found, the backward Riccati equations are used to find the optimal control trajectory. This LQR is iterated until the cost decreases.
The equations are given by:

$$P_N = Q_N \qquad p_N = q_N$$

$$K_n = -(R_n + B_n^T P_{n+1} B_n)^{-1} B_n^T P_{n+1} A_n$$

$$P_n = Q_n + A_n^T P_{n+1} A_n + A_n^T P_{n+1} B_n K_n$$

$$k_n = -(R_n + B_n^T P_{n+1} B_n)^{-1} (B_n^T p_{n+1} + r_n)$$

$$p_n = q_n + A_n^T p_{n+1} + A_n^T P_{n+1} B_n k_n$$

The solution is given by:

$$u_n = u_n^* + K_n(x_n - x_n^*) + k_n$$

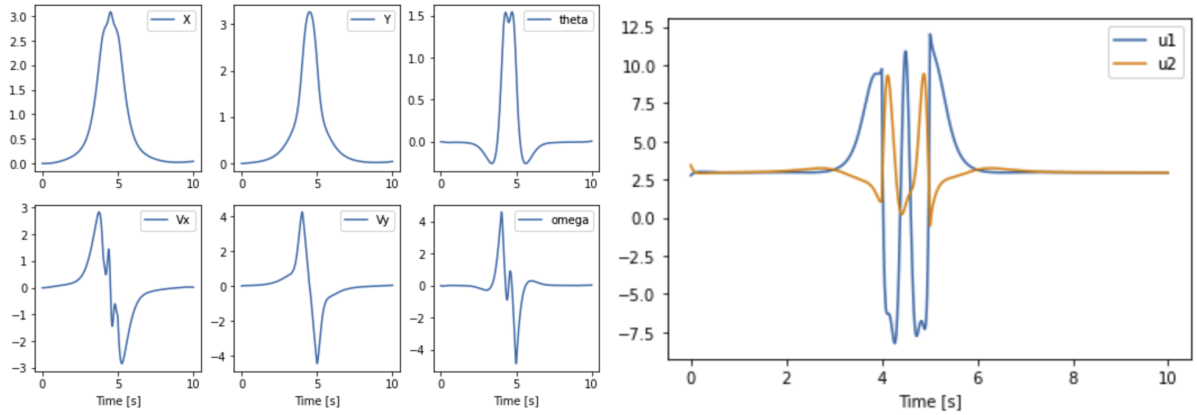
Sometimes the new controller will not be able to reduce the cost, so for that, the line search is performed, which is a very crucial step in the iLQR process.

$$u_n = u_n^* + K_n(x_n - x_n^*) + \alpha k_n$$

So, the concept of line search is to decrease the feedforward term in the equation by a factor of α, since it is the term that is responsible for the increase in cost. So whenever the cost value increases than the previous cost value, the value of α is decreased (in our case α becomes $\alpha/2$). This ensures the cost values are being decreased. When α values go beyond a particular value (say, 0.01), the loop can be terminated. It is suggested that $\alpha>0$. The loop can also be terminated when cost becomes stagnant or after a particular number of iterations (which is a hyperparameter).

Qsn 6: The algorithm is tested with an initial guess u* values initialized to zeros, or also can be initialized to the values found in Part 2, which doesn't actually matter. The algorithm is ran and loop is broken when $\alpha<0.1$, since the values have already converged for the tuned Q and R values. Initial state and control trajectories were not attached since they are already known.

After the loop converges, the final state and control trajectories can be visualized from the plots mentioned below:



It can be seen from the figures that the robot reaches the desired state at time t = 5 and except for that, the robot stays almost close to the origin for the remaining time. From the plots it can be seen that to reach the desired state at time t= 5, the robot starts moving around t = 3 seconds and comes back to the origin at around t =6 seconds. The control can also be seen changing between 3 and 6 seconds. The cost functions are tuned such a way that there is less cost for the control, so that it can freely move between time t= 4 to 6 seconds, and the cost at time t =4 to t=5 seconds, for the desired state is higher than the cost for the state in the remaining time, since it needs to be at the particular orientation at t=5 seconds. The tuning of the costs were done iteratively to determine which works best and gives less cost values.

Qsn 7: The benefits of this approach are that the cost gets calculated over and over again and it tries to converge the cost. One more thing is that it can be used in parallel, like when the controller starts running since it gives locally optimal trajectories, it can run in parallel with any other controllers to save time. Issues with this approach is that, the cost might explode sometimes, due to the decrease in alpha values, the tuning of iLQR takes a lot of time particularly, all the values should be keenly tuned to get the expected performance. Linearization of dynamics also is a drawback since it is susceptible to errors.

Task 2:

A new cost function is written for this task. The remaining process has been the same, so only the cost has been changed for this part which affects the working of the system.

If $x_0 = [0, 0, 0, 0, 0, 0]^T$ and $u_0 = [(m * g)/2, (m * g)/2]^T$,

$x_5 = [3, 0, 1.5, 0, pi, 0]^T$ and $u_5 = [(m * g)/2, (m * g)/2]^T$, and

$x_{10} = [3, 0, 0, 0, 2 * pi, 0]^T$ and $u_{10} = [(m * g)/2, (m * g)/2]^T$ then

Cost function at time step ‘t’ for t = 0 to 4 seconds is given by:

$$0.5 * (x_t - x_0)^T * Q1 * (x_t - x_0) + 0.5 * (u_t - u_0)^T * R1 * (u_t - u_0)$$

Cost function at time step ‘t’ for t = 4 to 5 seconds is given by:

$$0.5 * (x_t - x_5)^T * Q2 * (x_t - x_5) + 0.5 * (u_t - u_5)^T * R2 * (u_t - u_5)$$

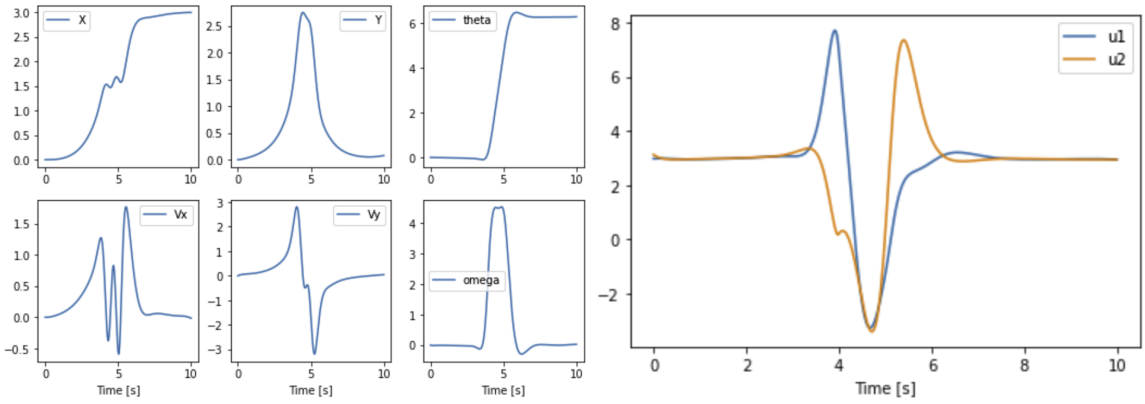
Cost function at time step ‘t’ for t = 5 to 9.99 seconds is given by:

$$0.5 * (x_t - x_{10})^T * Q2 * (x_t - x_{10}) + 0.5 * (u_t - u_{10})^T * R2 * (u_t - u_{10})$$

Cost function at time step ‘t’ for t = 10 seconds is given by (terminal cost):

$$0.5 * (x_t - x_{10})^T * Q2 * (x_t - x_{10})$$

The ‘u’ values were considered the same for all the cases because the control cost is varied accordingly so that they don't affect our system. The costs were tuned accordingly so that the cost converges, and the remaining loop is the same.



It can be understood from the plots that the state reaches our desired positions and the control is given accordingly. The plots are similar as in Task 1 (similar in the sense, reaching the desired states) and the intuition behind it is similar. But the major difference comes in the tuning part, which is pretty time consuming, since all the parameters accounted for should be taken into consideration and the costs should be tuned accordingly.
Using iLQR has benefits, but it has drawbacks as well, due to the errors occurring while linearizing the dynamics, it should be always paired with another controller (like an MPC) when used on a real time robot, so that the errors are mitigated from the other controller as well.