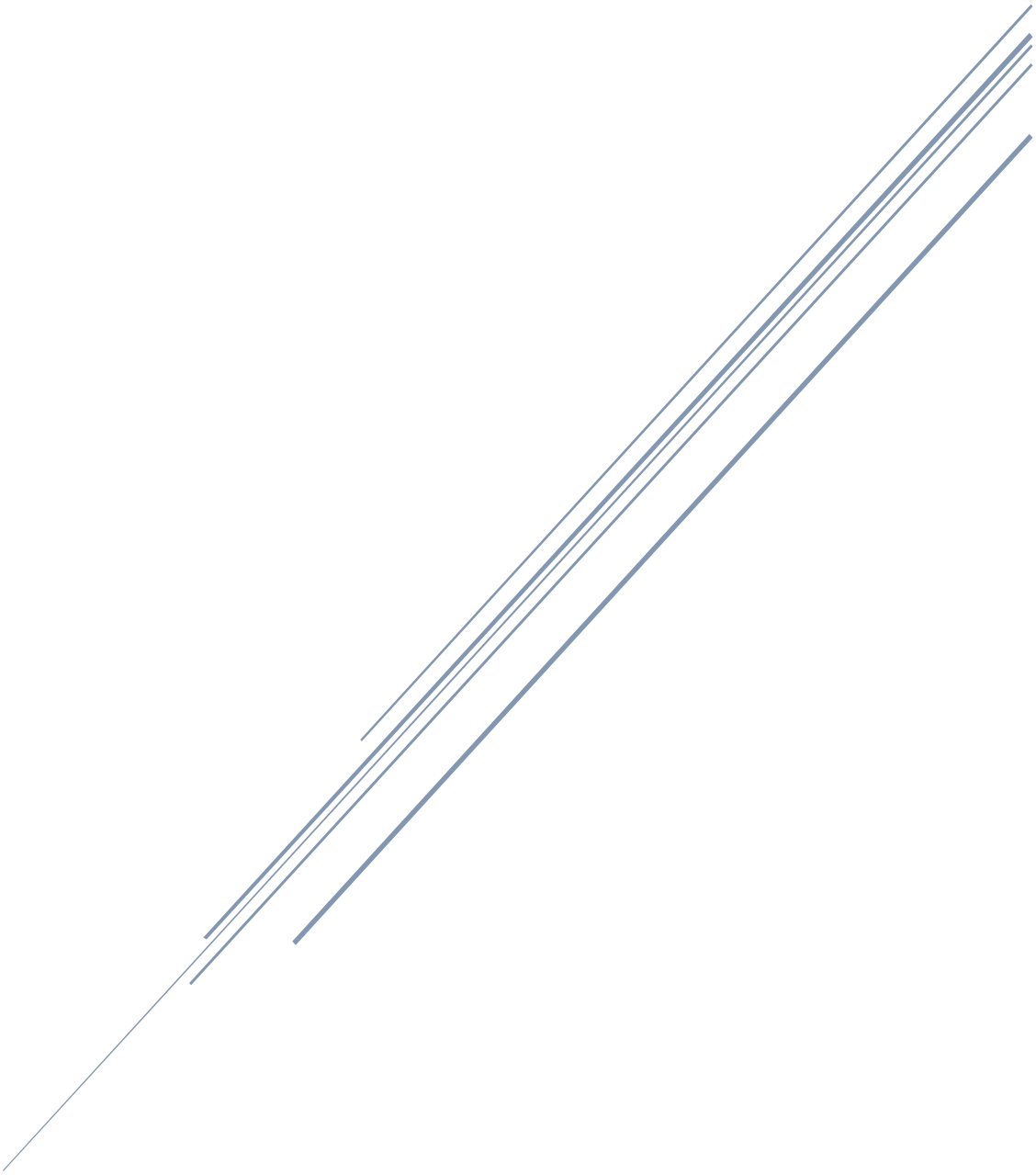
Swing-Up Control of a Robot Arm

Design Project

Kuan Hou Lee – 25952625
Eric Horng - 26935449



Monash University
ECE4132 – Control Systems Design

Introduction

As aspiring Electrical Engineers, it is fundamental that we learn how to create a controller for a control system. In this project, we were given a simulated planar robot system which consists of a two-link robot arm with one end connected at the top to a fixed pivot. Two actuators, one at the base and one at the pivot was used to manipulate the arm. A diagram of the described system can be seen below in Figure 1 and a differential equation model of the described system can be seen in Equation 1.

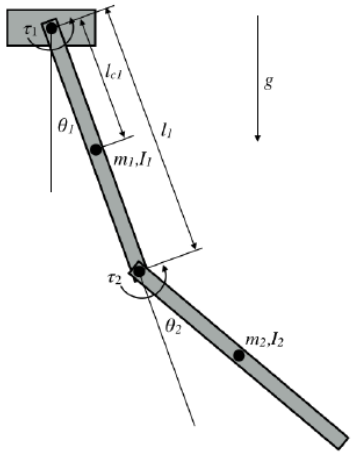


Figure 1 - A schematic diagram of the planar robot system [1]

$$\begin{bmatrix} I_1 + I_2 + m_2 l_1^2 + 2m_2 l_1 l_{c2} \cos(\theta_2) & I_2 + m_2 l_1 l_{c2} \cos(\theta_2) \\ I_2 + m_2 l_1 l_{c2} \cos(\theta_2) & I_2 \end{bmatrix} \begin{bmatrix} \frac{d^2\theta_1}{dt^2} \\ \frac{d^2\theta_2}{dt^2} \end{bmatrix} =$$
$$\begin{bmatrix} \tau_1 + 2m_2 l_1 l_{c2} \sin(\theta_2) \left(\frac{d\theta_1}{dt}\right)\left(\frac{d\theta_2}{dt}\right) + m_2 l_1 l_{c2} \sin(\theta_2) \left(\frac{d\theta_2}{dt}\right)^2 - (m_1 l_{c1} + m_2 l_1) g \sin(\theta_1) - m_2 g l_2 \sin(\theta_1 + \theta_2) \\ \tau_2 - m_2 l_1 l_{c2} \sin(\theta_2) \left(\frac{d\theta_1}{dt}\right)^2 - m_2 g l_2 \sin(\theta_1 + \theta_2) \end{bmatrix}$$

Equation 1 - Differential equation model of the planar robot system

Given this system, we were given the overall aim to swing the arm from the downwards equilibrium point to the upwards equilibrium point and balance it.

The robot arm had to adhere to following design specifications: Percentage overshoot for both θ_1 and $\theta_1 + \theta_2$ is at most 2% and 2% settling time for both θ_1 and $\theta_1 + \theta_2$ is at most 2 seconds.

Some hardware specifications for the quality of the sensors and actuators that also need to be considered include a sensor sampling rate of 20Hz, measurement noise variance of 0.0025 and actuator torque saturation of 10Nm.

To guide us through to creating the most ideal controller for the system, we were given five core project goals to complete.

1. Implement an ideal model of the system given the differential equation model of the system
2. Find the mass of the first link/pendulum
3. Design a continuous-time control design for the ideal model
4. Design a discrete-time control design for the ideal model
5. Evaluate control design on the true system

This report will aim to justify the design choices that we make to achieve the desired controller that satisfy the project goals with the limitations that have been set on us.

Goal 1 – Implement an ideal model of the system

Obtaining the F function from the differential equation model

To implement an ideal model of the system, we were asked to write a MATLAB function which would then be used by a Simulink model that would then simulate the system. To complete this we first looked at the given differential equation model of the system, Equation 1, and rewrote it into our desired form seen in Equation 2.

$$\begin{bmatrix} \frac{d^2\theta_1}{dt^2} \\ \frac{d^2\theta_2}{dt^2} \end{bmatrix} = F\left(\begin{bmatrix} \theta_1(t) \\ \theta_2(t) \\ \frac{d\theta_1}{dt} \\ \frac{d\theta_2}{dt} \end{bmatrix}, \begin{bmatrix} \tau_1(t) \\ \tau_2(t) \end{bmatrix} \right)$$

Equation 2 - Desired system model form

To get the differential model into the desired form, we aimed to isolate the $\begin{bmatrix} \frac{d^2\theta_1}{dt^2} \\ \frac{d^2\theta_2}{dt^2} \end{bmatrix}$ term. To simplify the differential equation, we will create a simplified form of the equation which is shown below in Equation 3.

$$K = JF \; where:$$

$$J = \begin{bmatrix} I_1 + I_2 + m_2 l_1^2 + 2m_2 l_1 l_{c2} \cos(\theta_2) & I_2 + m_2 l_1 l_{c2} \cos(\theta_2) \\ I_2 + m_2 l_1 l_{c2} \cos(\theta_2) & I_2 \end{bmatrix}, \; F = \begin{bmatrix} \frac{d^2\theta_1}{dt^2} \\ \frac{d^2\theta_2}{dt^2} \end{bmatrix}$$

$$K = \begin{bmatrix} \tau_1 + 2m_2 l_1 l_{c2} \sin(\theta_2)\left(\frac{d\theta_1}{dt}\right)\left(\frac{d\theta_2}{dt}\right) + m_2 l_1 l_{c2} \sin(\theta_2)\left(\frac{d\theta_2}{dt}\right)^2 - (m_1 l_{c1} + m_2 l_1) g sin(\theta_1) - m_2 g l_2 \sin(\theta_1 + \theta_2) \\ \tau_2 - m_2 l_1 l_{c2} \sin(\theta_2)\left(\frac{d\theta_1}{dt}\right)^2 - m_2 g l_2 \sin(\theta_1 + \theta_2) \end{bmatrix}$$

Equation 3 - Simplified system model

To get the F matrix by itself, we would need to multiply the inverse of the J matrix on the left-hand side of both sides of the equation.

$$\begin{aligned} K &= JF \\ J^{-1}K &= J^{-1}JF \\ J^{-1}K &= IF \\ J^{-1}K &= F \end{aligned}$$

The above matrix multiplication was calculated on MATLAB to minimise any potential error. The calculation was then put into a MATLAB function *Fmy_planar_robot.m* which would allow us to input the state variable x, both input torque values and the mass of the first pendulum. The function would then output a new F matrix that would be used in the Simulink model.

How does the Simulink model work?

The function that was created in the last step, *Fmy_planar_robot.m*, is called by the Simulink model *my_planar_robot.slx* which is used to simulate our ideal model of the system.

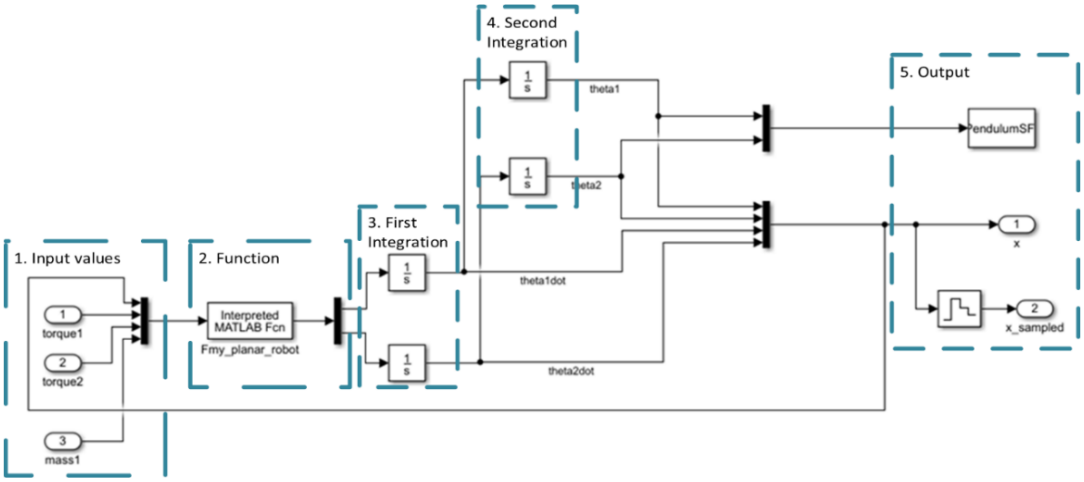


Figure 2 - my_planar_robot.slx Simulink model annotated

To better explain how the Simulink model works, an annotated model of the system can be seen above in Figure 2. As shown above, step 1 illustrates the input values that we input into the function Fmy_planar_robot in step 2. As expected, the output of the function returns us a 2x1 column matrix that consists of the values that represent $\begin{bmatrix} \frac{d^2\theta_1}{dt^2} \\ \frac{d^2\theta_2}{dt^2} \end{bmatrix}$. In step 3, we integrate the output of the function to return us with $\begin{bmatrix} \frac{d\theta_1}{dt} \\ \frac{d\theta_2}{dt} \end{bmatrix}$. In step 4, we calculate another integration to return us with $\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$, which gives us the angle of the two arms (See Figure 1). These results are then combined to form a new state variable, x, as shown in step 5, which is also then fed back to step 1.

As discussed above, the Simulink model implements the differential equation model as it uses the function, *Fmy_planar_robot.m*, that contains the differential equation re-arranged to give us $\begin{bmatrix} \frac{d^2\theta_1}{dt^2} \\ \frac{d^2\theta_2}{dt^2} \end{bmatrix}$. This output is necessary for us to determine the change in theta and the theta value.

Goal 2 – Identify the mass of the first pendulum

After creating an identical model of the system using the differential equation, we were then tasked to work out the unknown mass of the first pendulum. To find the mass, we had to input appropriate signals to the two systems in *goal2.slx* and edit the script *goal2_fit.m* that would help us accomplish this goal.

How does goal2_fit.m work?

The *goal2_fit.m* MATLAB script gives us a brute force method of determining the mass of the first robot arm. The MATLAB script calculates the least squared error between the 'real' physical system and the simulated system. Hence by running the script on a range of masses, we can find the minimum error which will indicate the mass that is closest to the 'real' mass.

The work flow of the script consists of:

1. Set up initial position of the robot arm
2. Define the range of estimate mass values to try
3. Run the model using the estimated mass one-by-one

4. Print out the mass with the smallest error value

How does the script implement a least-squares parameter estimation method?

To get a quantitative amount for the error value of the model, we will need to look at the Simulink model first. As shown below in Figure 3, the simulation error value is the difference between the output value of the real system, *planar robot output*, and the output value of our ideal model with our estimated mass value, *my planar robot output*.

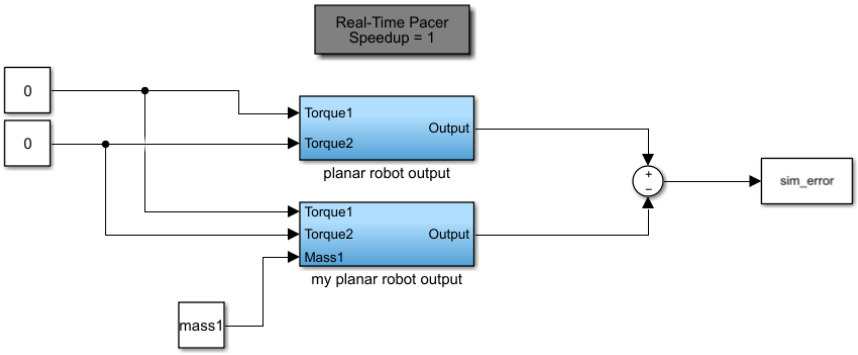


Figure 3 - goal2.slx Simulink model

The sim_error variable then stores the error value of both $\frac{d^2\theta_1}{dt^2}$ and $\frac{d^2\theta_2}{dt^2}$. The *goal2_fit.m* MATLAB script then calculates the square of each value and sums both $\frac{d^2\theta_1}{dt^2}$ and $\frac{d^2\theta_2}{dt^2}$ squared values. All these squared values are then summed up to give us a total error score. The mass with the least amount of error is then returned from the *goal2_fit.m* script.

Finding the mass

To find the estimated mass of the first pendulum, we would give zero input into both models. This is to reduce the complexity of the differential model, remove a variable and give a less chaotic result. We also set a simulation time of 10 seconds per run to reduce calculation times.

During our first run of this goal, we would set our initial position to be $[\pi \quad 0 \quad 0 \quad 0]$, which denotes a starting position of the robot arm pointing upwards. This would yield very large error results as the area of operation of the double pendulum was large. Small changes in movement would lead to large differences later in the simulation, akin to the butterfly effect. Hence it was extremely difficult to find the correct mass.

Our next strategy in solving this goal was to set our initial position a little bit lower so that we don't get large variances between the two models. By setting the initial position to $\left[\frac{\pi}{4} \quad 0 \quad 0 \quad 0\right]$ we were able to find the mass of the system with very little error. The area of operation of the double pendulum is much smaller, as the system is bounded by conservation of energy, and the movement is less chaotic.

To find the mass, we initially set the search range to be a large window, $0{:}\,0.1{:}\,1.5$. Once the mass associated with the minimum error is calculated, we narrow the search around this mass and choose a smaller incremental value e.g. $0.5{:}\,0.01{:}\,0.7$ then $0.65{:}\,0.001{:}\,0.67$… etc. This was repeated until we got a value of reasonable resolution.

The final mass of the first pendulum was found to be 0.66283kg.

Goal 3 – Continuous-time full-state control design for ideal model

After creating an ideal model of the system and finding the mass of the first pendulum, the next goal is to design a feedback control system for the ideal model.

Designing the controller

To design the controller, we first needed to find the state space model of the system and then linearize the system around the upward vertical equilibrium point since the differential equation contains non-linear sine/cos terms. To do this, we initially tried to calculate this by hand, but we eventually gave up as it proved too complex. Eventually, we managed to figure out that we could use symbolic variables in MATLAB and the *diff* function to calculate partial derivatives which would allow us to linearize the system very quickly. As seen from the *goal3_setup.m* MATLAB script, the script follows the following workflow:

1. Calculate F matrix (Equation 3)
2. Linearize F matrix around the upwards equilibrium point by making a change of variables $x \to z$ for the input and $u \to v$ for the output given by the following equation:

$$z(t) = x(t) - x_e \ , \ \ v(t) = u(t) - u_e$$

Equation 4- Change of variables w.r.t equilibrium point

As we are linearizing with reference to the upwards equilibrium point $[\pi \quad 0 \quad 0 \quad 0]$, we substitute for x:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} z_1 + \pi \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}$$

3. As the continuous time state equation is given by:

$$\frac{dx}{dt} = Ax(t) + Bu(t)$$

Equation 5. State space equation

we can equate terms to see that $\frac{dx_1(t)}{dt}$ is given by the $x_3(t)$, $\frac{dx_2(t)}{dt}$ is given by $x_4(t)$, $\frac{dx_3(t)}{dt}$ and $\frac{dx_4(t)}{dt}$ is given by the differential equation (Equation 3). τ_1, τ_2 are our inputs hence they are removed from F. Our undifferentiated A matrix is:

$$A = \begin{bmatrix} x_3 \\ x_4 \\ F_{2x1} \end{bmatrix}$$

A is then partially differentiated in terms of z to give:

```
         0         0    1.0000         0
         0         0         0    1.0000
    3.5081   31.2139         0         0
  -46.8359 -126.4901         0         0
```

4. Work out the partial derivatives of the linearized F matrix, in terms of v, which will form our B matrix. Since this is a 2-input system and the inputs only appear in $\frac{dx_3}{dt}$ and $\frac{dx_4}{dt}$ terms, our undifferentiated B matrix is:

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ J^{-1}\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \end{bmatrix}$$

Partial differentiating in terms of τ_1 and τ_2 gives:

```
         0         0
         0         0
    2.1234   -6.1048
   -6.1048   22.2387
```

5. Our output is the full state hence set our C matrix to be a 4x4 identity matrix
6. Use the Linear-Quadratic Regulator (LDR) MATLAB function to determine the most optimal gain matrix *K*.

Once this MATLAB script was created, we needed to design the controller on Simulink. We used a state feedback design as shown below in Figure 4. Note that there is no reference term as per usual feedback control systems. This is because we calculated the reference gain k_r and discovered it to be negligible.

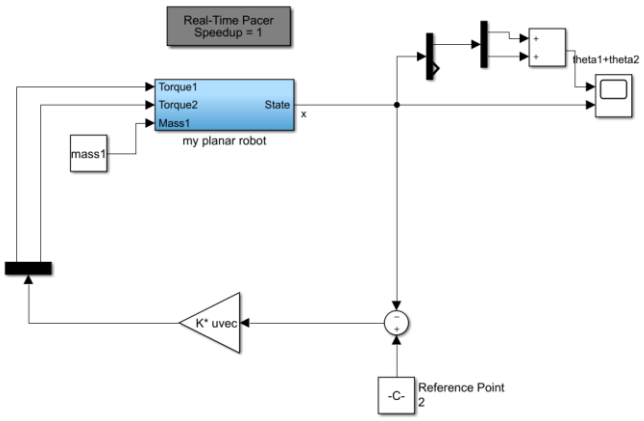


Figure 4 - Simulink model of the continuous-time controllers

Performance specifications and results

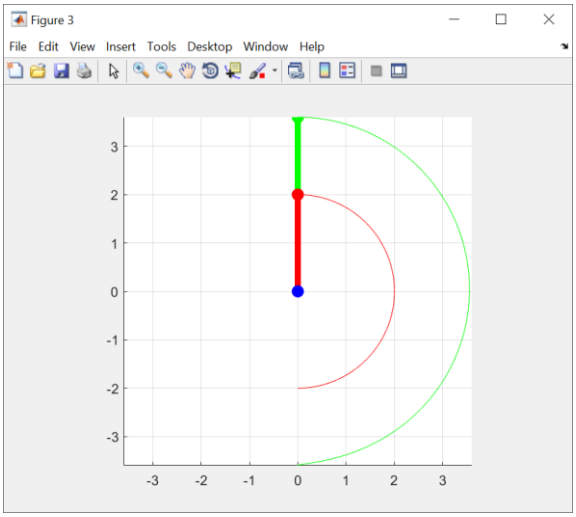


Figure 5 - Balanced robot arm with continuous-time controller

Figure 5 shows us the path that the robot arm takes when using our controller. To further show that our controller had met the requirements, we connected a scope to our Simulink model to view the output from the "my planar robot" block, which can be seen below in Figure 6. In the plot, you can see that θ_1 and $\theta_1 + \theta_2$, yellow and blue respectively, has a settling time of roughly 1 seconds and that there are no overshoots for θ_1 and θ_2. Based on the requirements, specified in the introduction, the controller that we designed meets the specifications.

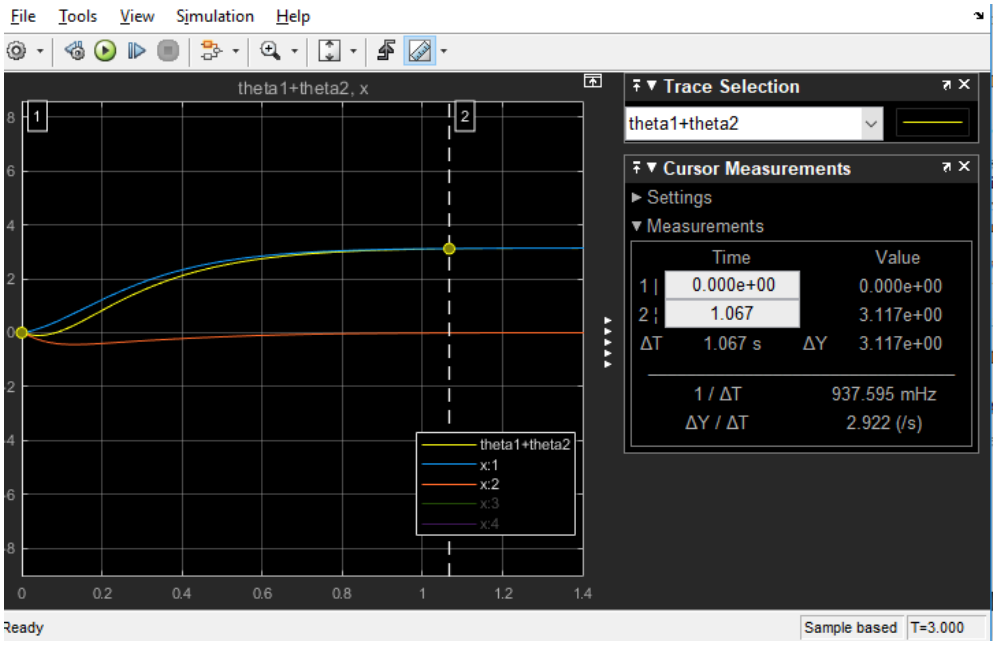


Figure 6 - Continuous-time controller system response

A difficult task that we encountered was trying to figure out what eigenvalues to use to create our K matrix. At first, we calculated our *K* matrix using the *place* function in MATLAB with our desired eigenvalues $a \pm bj$ based on the settling time and overshoot values as detailed by the design specification given by:

$$T_{settling} = \frac{1}{x}\ln(0.002), \qquad \%_{overshoot} = \exp\left(\frac{\pi a}{b}\right)$$

Equation 6 - Eigenvalue calculation based on settling and overshoot

Whilst choosing the other two eigenvalues to be a scalar multiple of the real part of the complex conjugate since a large negative eigenvalue was chosen to provide a fast-transient response. However, this led to us using trial-and-error to figure out the correct eigenvalues to choose as some eigenvalues were stable and gave correct operation and others were not. In the end, we eventually used the *lqr* function with the Q and R matrices highlighted in Equation 7. The MATLAB lqr function calculates the k feedback gains given weighted parameters Q and R. These matrices determine the control effort and error associated with the final system, hence by changing these values we can tailor specific overshoot and settling time parameters.

$$Q = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}$$

Equation 7 - Q and R matrices for LQR function with continuous-time system

The resulting K (gain) matrix was:

```
K =

   81.5708   -5.6503   21.1688    3.0079
   -7.8313   92.7489    2.6039   11.1173
```

Goal 4 – Discrete-time full-state control design for ideal model

Designing the controller

After designing a continuous-time controller for the ideal model, our next goal was to create a discrete-time controller instead. To design a discrete-time controller, we can reuse the continuous-time controller that we made in Goal 3 and find an equivalent discrete-time controller by using the *c2d* MATLAB function. Note that to get the zero-order hold equivalent, we simply need to specify that we want to use the zero-order hold discretization method into the *c2d* MATLAB function.

```
% Get zero order hold equivalent system
sys = ss(A,B,C,0);
sys_discrete = c2d(sys, sampling_period, 'zoh');
[A,B,C,D] = ssdata(sys_discrete);
```

Figure 7 - Code snippet of how to get zero-order hold equivalent controller

After getting the discrete-time system, we reused the same Simulink model (Figure 4) to simulate our ideal model in discrete time. The gain of the system was found using the *lqrd* MATLAB function which is discrete time equivalent of lqr used in goal 3. Since this system is different to the continuous-time system, we needed to find a new Q and R matrix. Using trial-and-error we ended up using the values seen in equation 8.

$$Q = \begin{bmatrix} 1000 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}$$

Equation 8 - Q and R matrices for LQR function with discrete-time system

Performance specifications and results

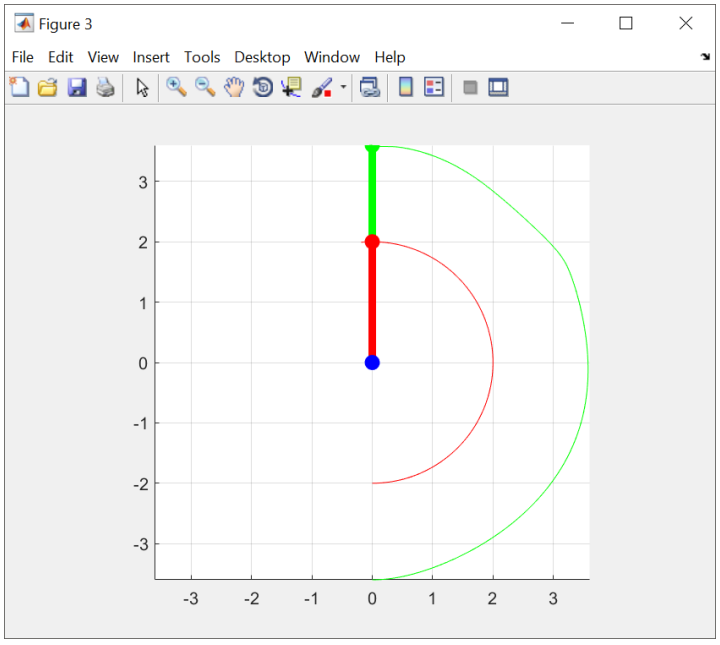


Figure 8 - Balanced robot arm with discrete-time controller

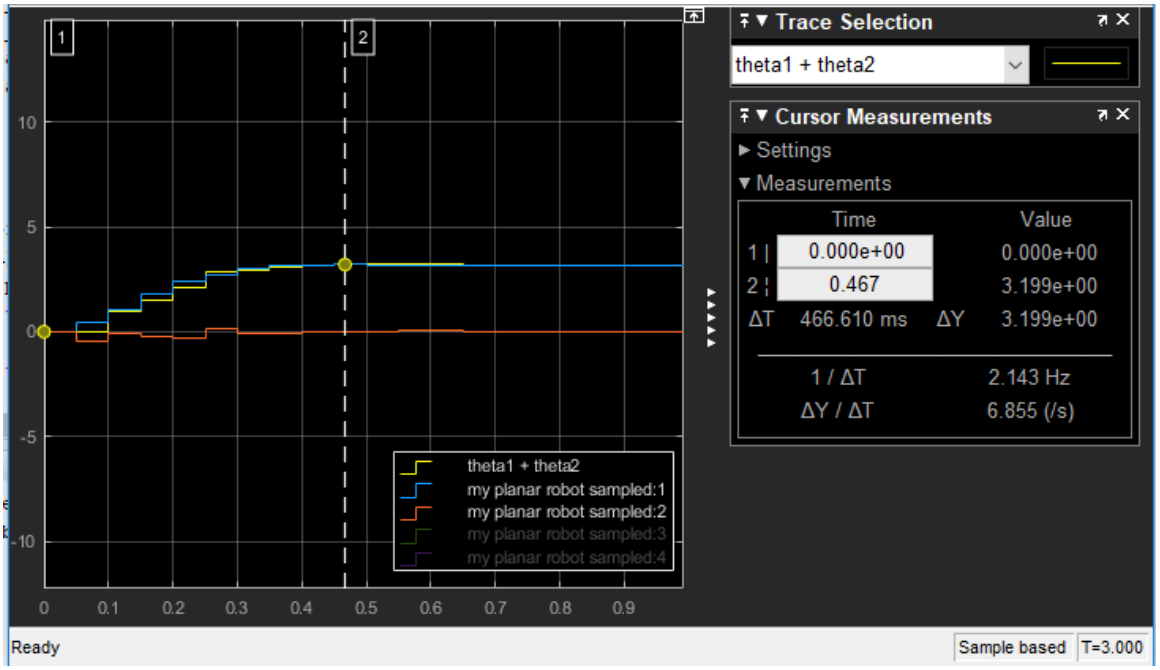


Figure 9 - Discrete-time controller system response

From Figure 8 and Figure 9, we can see that the system correctly balances and meets design specifications, with no overshoot and a settling time of approximately 0.466 seconds.

Goal 5 – Evaluation of design on system

Goal 5 is similar to goal 4 except that the system is harder to control due to input saturation and added measurement noise. This made it difficult to control as the range of stable poles has been reduced- we needed new eigenvalues which had a longer settling time and is more robust to account for the added noise. Initially a Karman filter was employed to reduce the amount of noise in the system, but this proved unsuccessful. A moving average filter was considered to smooth out the state, however this introduced a delay equal to the length of the sliding window. In the end, the model was reverted the original system (Figure 4) used in goal 4 and LQR was used again in this goal to determine optimal feedback gains in a trial and error approach. The following Q and R matrices proved to be successful in balancing the arm in the upwards position:

$$Q = \begin{bmatrix} 1000 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}$$

Equation 9 - Q and R for goal 5

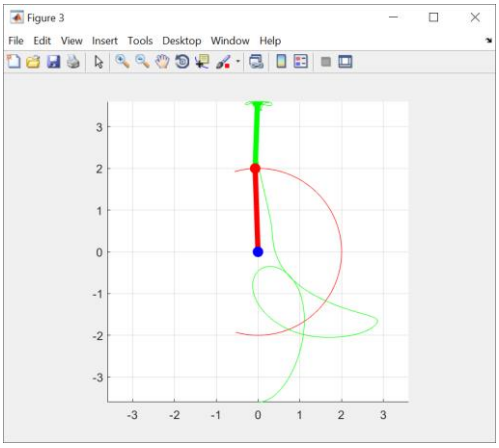


Figure 10 - Pendulum movement with input saturation and measurement noise.

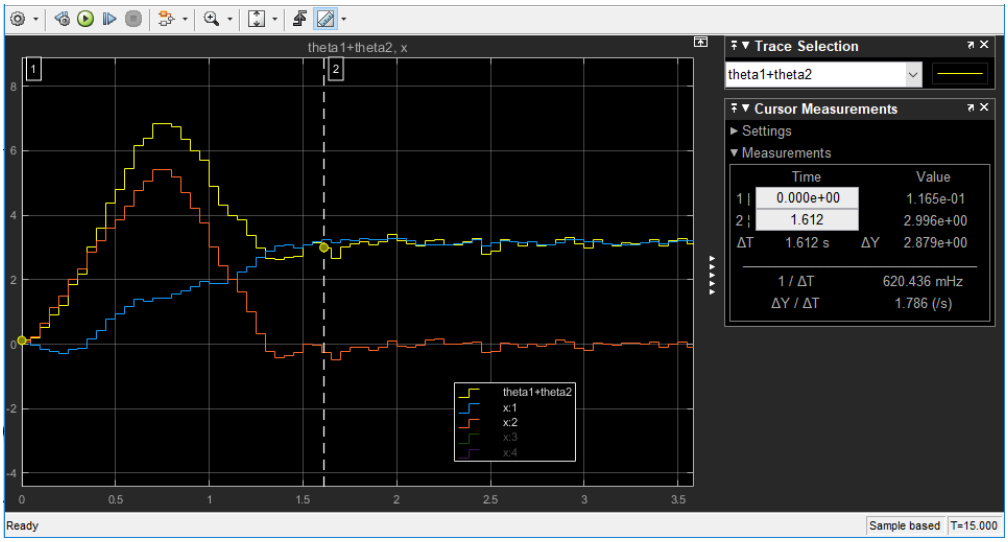


Figure 11 - theta1, theta, theta 1+theta 2 variables for goal 5

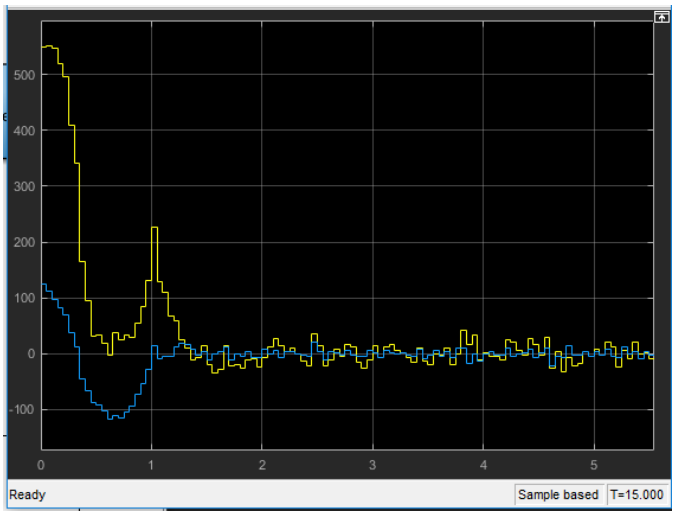


Figure 12. Input signals into system

From Figure 11 we can see that the system balances on the upwards equilibrium point however it took considerably longer to settle. This is due to input saturation as the system needs to take a longer, less stressful path to the equilibrium point to account for reduced torque.

Further Goals: Optimising sampling period

The sampling period was chosen to be optimised as an extension to the project. As we had success in goal 5, we chose to keep the structure and use the same method to find stable feedback gains. The basic procedure was as follows:

1. Increase sampling period by a small amount
2. Run the model and see if it balances
    a. If it doesn't balance, tune the Q and R matrices till it does
    b. If it balances, increase the sampling period again
3. Repeat until we cannot find suitable Q and R to make the system balance.

The optimal sampling period we found was 0.08 seconds (12.5Hz), which is an increase of 60% from the original sampling period of 0.05 seconds (20Hz). The following Q and R matrices were used:

$$Q = \begin{bmatrix} 1000 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}$$
$$R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}$$

Equation 10 - Q and R for optimal sampling period

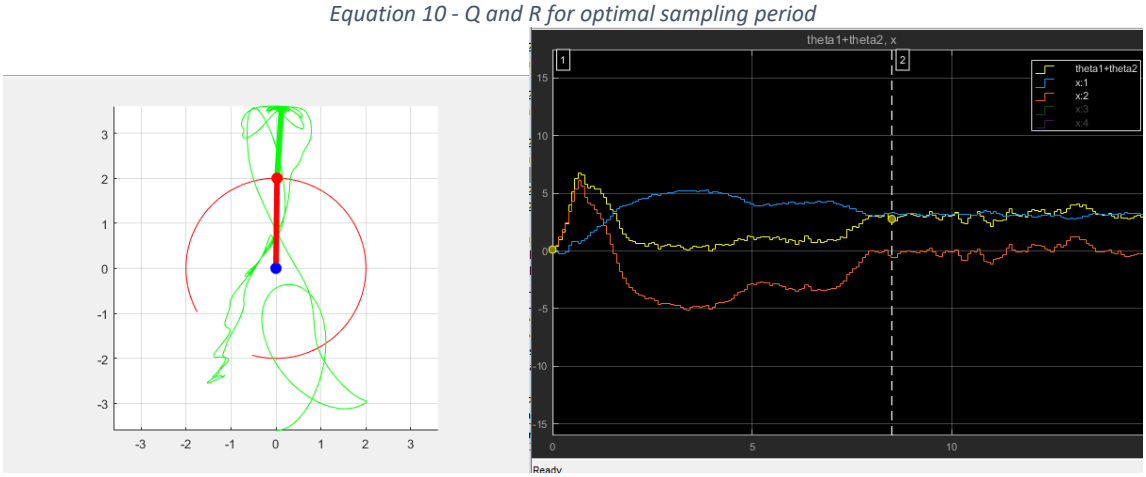


Figure 13 (Left). Path of additional goal pendulum and Figure 14 (Right) Theta1, Theta2, Theta1+Theta2 performance characteristics for optimal sampling period

As shown by the green line in Figure 13, the system is struggling to get to the upwards equilibrium point and took an alternate path than other goals. From Figure 14 we can see that that the system took much longer to settle (approximately 8.5 seconds) than the other ones due to the increase in sampling period.

Conclusion

We met all design specifications and optimised the sampling period. For goal 1, we successfully modified the differential equation to give us proper operation in the double pendulum model. For goal 2, we used a least squares estimation method to get the mass of pendulum 1 which was calculated to be 0.66283kg. For goal 3 and 4 we successfully balanced the arm in the upwards position using continuous and discrete models respectively. For goal 5, we used the lqrd function to find optimal feedback gains to balance the arm in the presence of input saturation and measurement noise. Finally, in goal 6 the sample period was optimised and the largest value we discovered was 0.08 seconds (12.5Hz).