EE324 Experiment 2

Tuesday Group 16

Jay Mehta (22B1281)

Tanay Bhat (22B3303)

Aagam Shah (22B1201)

August 2024

Contents

1 Objective **2**
- 1.1 Aim of the experiment . . . 2
- 1.2 Design Requirements . . . 2
- 1.3 Control Flow Block Diagram . . . 2

2 Control Algorithm - LQR **3**
- 2.1 Introduction . . . 3
- 2.2 System Model . . . 3
- 2.3 Cost Function . . . 3
- 2.4 Feedback Control Law . . . 4
- 2.5 Solution of LQR: Algebraic Riccati Equation (ARE) . . . 4
  - 2.5.1 Hamilton-Jacobi-Bellman (HJB) Equation . . . 4
  - 2.5.2 Substitute the feedback control into the system dynamics . . . 4
  - 2.5.3 Solve the Riccati Equation . . . 4
  - 2.5.4 Compute the Optimal Gain Matrix K . . . 4
- 2.6 Properties of LQR . . . 4

3 Approach **5**

4 Simulink Model **7**

5 Challenges Faced **8**
- 5.1 Faulty Motor . . . 8
- 5.2 Worked but didn't satisfy required conditions . . . 8
- 5.3 Used wrong terminals from power source leading to voltage drop for high currents . . . 8

6 Result **9**
- 6.1 Final Gain Values . . . 9
- 6.2 Defining Steady State Spreads . . . 9
- 6.3 Final Graphs . . . 10

7 Observation and Inference **11**

Chapter 1

Objective

1.1 Aim of the experiment

- Design State-Space feedback controller using LQR for gain calculation for inverted pendulum
- Implement the controller using Arduino Mega

1.2 Design Requirements

- The armature must rotate by at most 30° from the mean position while trying to balance the pendulum.
- The pendulum arm must rotate by at most 3° from the mean position when upright.

1.3 Control Flow Block Diagram

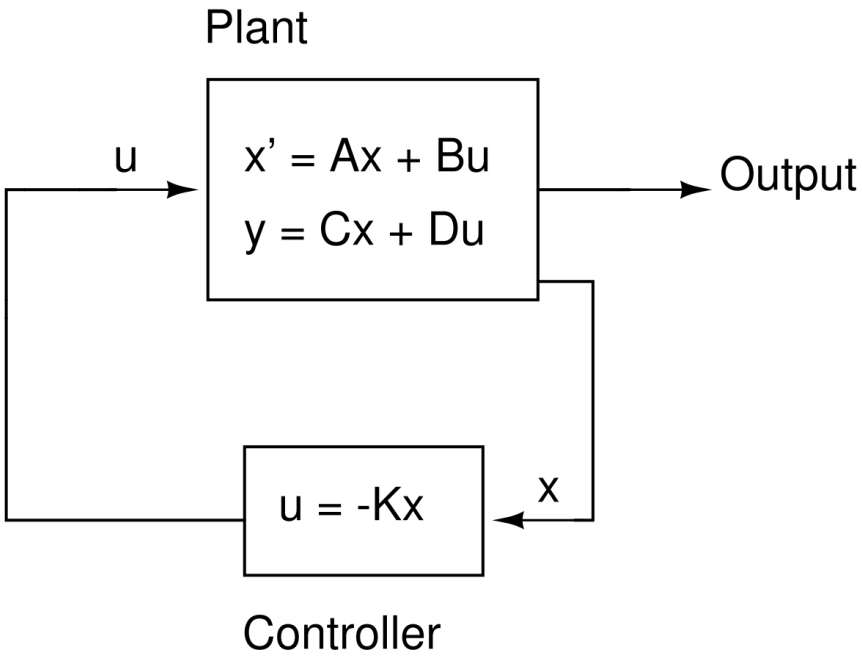


Figure 1.1: Control Flow

Chapter 2

Control Algorithm - LQR

2.1 Introduction

The **Linear Quadratic Regulator (LQR)** is a control strategy used to design optimal controllers for linear systems. It minimizes a cost function that captures both the energy of the system's control inputs and the deviation of the system's state from the desired reference.

2.2 System Model

LQR applies to systems that can be described by **linear time-invariant (LTI)** state-space equations:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{2.1}$$

where $x(t) \in \mathbb{R}^n$ is the state vector, $u(t) \in \mathbb{R}^m$ is the control input, $y(t) \in \mathbb{R}^p$ is the output, and:

- A is the state matrix of size $n \times n$.
- B is the input matrix of size $n \times m$.
- C is the output matrix of size $p \times n$.
- D is the feedforward matrix of size $p \times m$.

The goal is to find a control input $u(t)$ that stabilizes the system and minimizes a cost.

2.3 Cost Function

The LQR minimizes a quadratic cost function over time, which penalizes both the state deviation and the control effort:

$$J = \int_0^\infty \left(x(t)^T Q x(t) + u(t)^T R u(t) \right) dt \tag{2.2}$$

where:

- $Q \succeq 0$ is the state penalty matrix (of size $n \times n$), which weights how much we penalize deviations of the state from zero.
- $R \succ 0$ is the control penalty matrix (of size $m \times m$), which weights the magnitude of the control effort.

The **objective** is to minimize J by choosing the optimal control $u(t)$.

2.4 Feedback Control Law

LQR designs the control input as a **linear feedback** based on the state $x(t)$:

$$u(t) = -Kx(t) \tag{2.3}$$

where K is the **feedback gain matrix** to be determined. The key challenge in LQR is to compute this matrix K, which will minimize the cost function J.

2.5 Solution of LQR: Algebraic Riccati Equation (ARE)

To find K, we need to solve the **Algebraic Riccati Equation (ARE)**. The steps are as follows:

2.5.1 Hamilton-Jacobi-Bellman (HJB) Equation

The optimal control problem is solved using dynamic programming, leading to the HJB equation. The HJB approach leads to a solution in the form of a quadratic value function:

$$V(x) = x^T P x \tag{2.4}$$

where P is a symmetric, positive semi-definite matrix, called the **Riccati matrix**.

2.5.2 Substitute the feedback control into the system dynamics

Substituting $u(t) = -Kx(t)$ into the system equation:

$$\dot{x}(t) = (A - BK)x(t) \tag{2.5}$$

The goal is to ensure the closed-loop system is stable, i.e., $A - BK$ has eigenvalues with negative real parts.

2.5.3 Solve the Riccati Equation

The Riccati matrix P is the solution of the **Algebraic Riccati Equation (ARE)**:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{2.6}$$

This equation is derived from minimizing the cost function and ensuring stability. P is computed by solving this matrix equation.

2.5.4 Compute the Optimal Gain Matrix K

Once P is known, the optimal feedback gain matrix K is given by:

$$K = R^{-1}B^T P \tag{2.7}$$

2.6 Properties of LQR

- **Stability**: The closed-loop system with $u(t) = -Kx(t)$ is guaranteed to be stable if A and B are controllable.
- **Optimality**: LQR gives an optimal solution in terms of minimizing the quadratic cost function.

Chapter 3

Approach

- Determine ADC output to angle mapping and map the angles accordingly to get the origin of α and θ at the desired locations.
- Calculate A, B, C, D matrices for the system using the model parameters as outlined in the lab manual. Using these matricies and the Q and R values, calculate the optimal gain matrix K using MATLAB.
- Connect the driving circuit for the inverted pendulum and test the obtained gain values from MATLAB.
- Based on the way the system behaves, we can adjust the Q and R matrices to get the gains that give the desired response.

```
Mp = 0.027;          %Mass of assemby
lp = 0.153;          %Length of COM from pivot
Lp = 0.191;          %Total length
r = 0.08260;         %Arm pivot to pendulum pivot
Jm = 3e-5;           %Shaft MOI
Marm = 0.028;        %Mass of arm
g = 9.81;            %Acceleration due to gravity
Jeq = 1.23e-4;       %Equiv. Moment of Inertia about Motor Shaft Pivot Axis
Jp = 1.1e-4;         %Pendulum MOI about pivot axis
Beq = 0;             %Arm viscous damping
Bp = 0;              %Pendulum viscous damping
Rm = 3.3;            %Motor Armature Resistance
Kt = 0.02797;        %Motor torque constant
Km = 0.02797;        %Motor Back-EMF constant
```

Listing 3.1: Initialize the System Parameters

```
denom = (Jp*Jeq + Mp*lp*lp*Jeq + Jp*Mp*r*r);    %common term in denominator in all terms
A32 = (r*Mp*Mp*lp*lp*g)/denom;
A33 = -(Kt*Km*(Jp+Mp*lp*lp))/(denom*Rm);
A42 = (Mp*lp*g*(Jeq+Mp*r*r))/denom;
A43 = -(Mp*lp*Kt*r*Km)/(denom*Rm);

A = [0 0 1 0;
    0 0 0 1;
    0 A32 A33 0;
    0 A42 A43 0];

B3 = Kt*(Jp + Mp*lp*lp)/(denom*Rm);
B4 = Mp*lp*Kt*r/(denom*Rm);

B = [0;
    0;
    B3;
    B4];
```

Listing 3.2: Define A and B matricies

```
%LQR Design
Q = diag([1200 2000 20 100]);    %State Penalty Matrix
R = 1;      %Control Penalty Matrix

% K = Optimal Gain Matrix
% S = Solution of Riccati equation
% E = Closed Loop Poles
[K, S, E] = lqr(A,B,Q,R);
```

Listing 3.3: LQR Function to get K

This way we get the gain matrix K which can be used to control the system. The K matrix is of the form:

$$K = \begin{bmatrix} K_\theta & K_\alpha & K_{\dot{\theta}} & K_{\dot{\alpha}} \end{bmatrix}^\top \tag{3.1}$$

By multiplying each gain with the appropriate parameter, we get the ouput value which is fed to the motor to control its speed and direction of rotation.

```
    #define MAX            16384.0 // Maximum encoder output

    int en0 = getPositionSPI(ENC_0, RES14);
    int en1 = getPositionSPI(ENC_1, RES14);
    theta = (en0/MAX)*360; // Mapping encoder data to angle
    alpha = (en1/MAX)*360 - 180; // Mapping encoder data to angle
                                 // Subtract 180 to get alpha 0 at top
    if(theta>180.0)
    {
      theta-=360; // Map theta to [-179, 180]
    }

    theta_r=theta*PI/180.0; // Convert to radians
    alpha_r=alpha*PI/180.0; // Convert to radians
```

Listing 3.4: Mapping Encoder Data to Angle

The current and previous values of θ and α are used to calculate $\dot{\theta}$ and $\dot{\alpha}$ which are then multiplied by the gain matrix K to get the output value. This ouput is mapped to $[0, 255]$ (range of PWM) and fed to the motor to control its speed and direction of rotation.

```
    // dt = 1 for simplicity
    dtheta=(theta_r-theta_prev)/dt*1000.0; // Calculate dtheta
    dalpha=(alpha_r-alpha_prev)/dt*1000.0; // Calculate dalpha
    theta_prev=theta_r; // Update theta_prev
    alpha_prev=alpha_r; // Update alpha_prev

    // Gains are stored in the K array
    u=(K[0]*theta+K[1]*alpha+K[2]*dtheta+K[3]*dalpha); // Calculate output value
```

Listing 3.5: Calculation of Ouput Value

```
    if(u>0)
    {
        Vm=map(u,0,12,0,255); // Map to PWM values
        Vm = max(Vm, offset);
        //Offset used to ensure the minimum required PWM is supplied to the motor
        analogWrite(motor_A,0); // Rotate motor in clockwise direction
        analogWrite(motor_B,constrain(Vm,0,255));
    }
    else
    {
        Vm=map(-u,0,12,0,255); // Map to PWM values
        Vm = max(Vm, offset);
        analogWrite(motor_A,constrain(Vm,0,255)); // Rotate motor in anticlockwise direction
        analogWrite(motor_B,0);
    }
```

Listing 3.6: Direction Control of Motor

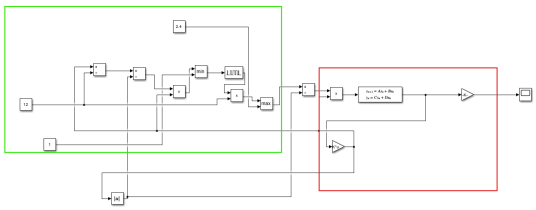
Chapter 4

Simulink Model

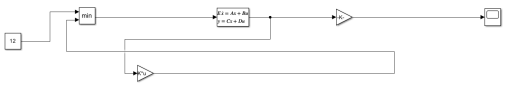
The A, B, C, D matricies obtained previously help model a continuous time system in Simulink. Since the Arduino control is dicrete, we need to get the dicrete time matricies for simulation.

```
%%
sys_c = ss(A, B, C, D);  % Continuous-time system
Ts = 0.001;  % Sampling time = 0.001s which is the delay used in Arduino code
sys_d = c2d(sys_c, Ts, 'zoh');  % Zero-order hold (ZOH) discretization
[A_d, B_d, C_d, D_d] = ssdata(sys_d);  % Discrete-time matrices
```

Listing 4.1: Getting Discrete Time Matricies

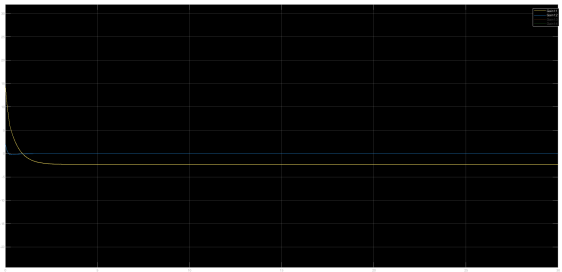


(a) Discrete Model

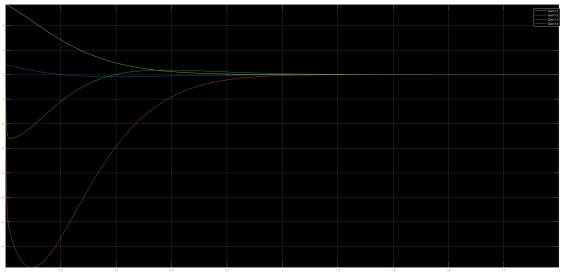


(b) Continuous Model

The green box denotes the PWM generation and the red box denotes the feedback system. The feedback system is the same as the one used in the Arduino code. The output of the feedback system is fed to the PWM generation block which generates the PWM signal to control the motor. PWM generation is not needed in the continuous model. The following graphs were obtained on simulation:



(c) Discrete Model



(d) Continuous Model

Discrete model takes longer to settle as compared to the continuous model. This is because the discrete model has a delay of 0.001s which is the time taken to run the code on the Arduino. The continuous model does not have this delay and hence settles faster. Note that the simulated system does not show the oscillations seen in the practical setup which may arise due to random disturbances and noise in the system.

Chapter 5

Challenges Faced

5.1 Faulty Motor

The motor used in our setup was giving poor results and the pendulum was not able to balance properly. We had to replace the motor with a new one to get the desired results.

5.2 Worked but didn't satisfy required conditions

Once the motor issue was resolved, we began fine-tuning the gains in order to achieve the required θ and α constraints. Some gain values didn't propery while others that worked, overshot the constraint requirements.

```
1    const double K[4] = {-22.3607, 220.6405, -9.1089, 30.6108};
```

These gain values resulted in very poor performance as the pendulum would quickly destabalise. Hence the Q and R matricies required further tuning.

```
1    const double K[4] = {-1.9047, 123.8395, -3.1640, 18.5507};
```

These gain values resulted in better performance however the armature rotated by large angles and hence did not fit the required constraints.

```
1    const double K[4] = {-10, 174.5739, -5.9164, 24.6447};
```

These gain values resulted in more reasonable results as the pendulum did stabalise however it still needed more fine tuning to ensure that the pendulum angle fit within the required constraints.

```
1    const double K[4] = {-31.6228, 299.0244, -12.9488, 40.9459};
```

These gain values were obtained using Q = diag([1000, 2000, 10, 100]), R = 1 which resulted in best run with worst-case θ spread = 75°, Stable θ spread = 40°. Note that these spreads were calculated in a very crude manner by eye-balling the graphs. In the results section we shall present 3 ways of defining steady state spread of the system.

5.3 Used wrong terminals from power source leading to voltage drop for high currents

Another error on our part was using the wrong terminals of the power supply which lead to the supply going into constant current mode which caused incorrect voltage input at the motor terminals. This was rectified by changing the terminals and increasing the current to a value that ensured constant voltage operation.

Chapter 6

Result

6.1 Final Gain Values

The values of gains that we finally arrived at were as follows:

```
const double K[4] = {-34.6410, 342.4425, -14.7675, 46.7231}
```

The Q and R matricies used for this were:

$$Q = \begin{bmatrix} 1200 & 0 & 0 & 0 \\ 0 & 2000 & 0 & 0 \\ 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix} \quad R = [1] \tag{6.1}$$

We shall now define steady state spread of θ and α using 3 different ways with the bound becoming looser with each successive definition.

6.2 Defining Steady State Spreads

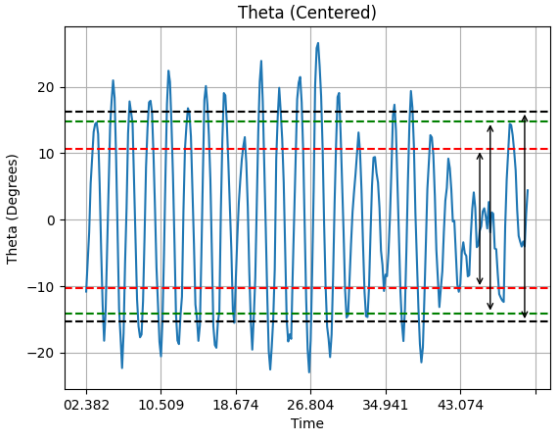
Since it is not easy to quantify steady state in a practical system, we settle for some approximations. We first center our data by subtracting the mean from each point. Let θ_C and α_C denote the centered readings. Then split the data as:

$$\begin{aligned} \{\theta, \alpha\}_{\text{above}} &= \{x \in \{\theta_C, \alpha_C\} | x > 0\} \\ \{\theta, \alpha\}_{\text{below}} &= \{x \in \{\theta_C, \alpha_C\} | x < 0\} \end{aligned} \tag{6.2}$$

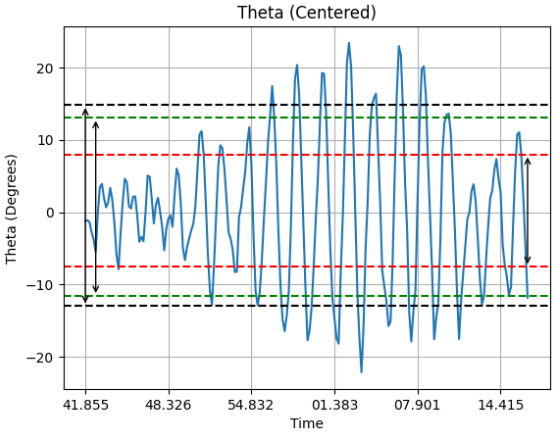
We now define the following:

- **Mean Spread :**
$$\text{Spread}_{\text{mean}} = \{\bar{\theta}_{\text{above}}, \bar{\alpha}_{\text{above}}\} - \{\bar{\theta}_{\text{below}}, \bar{\alpha}_{\text{below}}\} \tag{6.3}$$
- **RMS Spread - 4th Power :** The RMS Spread - 4th Power is defined as:
$$\text{Spread}_{\text{RMS-4}} = \sqrt[4]{\frac{1}{N}\sum_{i=1}^{N}\{\theta, \alpha\}^4_{\text{above}}} - \left(-\sqrt[4]{\frac{1}{N}\sum_{i=1}^{N}\{\theta, \alpha\}^4_{\text{below}}}\right) \tag{6.4}$$
- **RMS Spread - 6th Power :** The RMS Spread - 6th Power is defined as:
$$\text{Spread}_{\text{RMS-6}} = \sqrt[6]{\frac{1}{N}\sum_{i=1}^{N}\{\theta, \alpha\}^6_{\text{above}}} - \left(-\sqrt[6]{\frac{1}{N}\sum_{i=1}^{N}\{\theta, \alpha\}^6_{\text{below}}}\right) \tag{6.5}$$

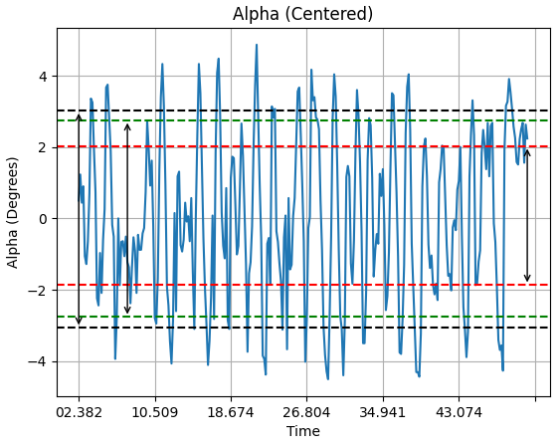
6.3 Final Graphs



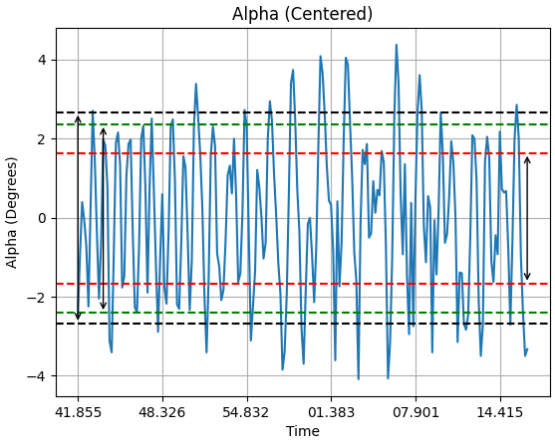
(a) Run 1



(b) Run 2



(c) Run 1



(d) Run 2

Run	$\text{Spread}_{\text{Mean}}$	$\text{Spread}_{\text{RMS-4}}$	$\text{Spread}_{\text{RMS-6}}$
Run 1 θ	20.87°	28.79°	31.58°
Run 2 θ	15.51°	24.58°	27.76°
Run 1 α	3.90°	5.52°	6.09°
Run 2 α	3.29°	4.76°	5.33°

Chapter 7

Observation and Inference

We see that mean spread provides a very bad underesitmate while the RMS caputures the steady state spread in a much better manner. There were some outliers present in the measured data which caused the worst case spread to exceed the required limits at times. For these two runs however we can see that the steady state spreads are well within the required bounds ($\theta_{\text{spread}} < 60°, \alpha_{\text{spread}} < 6°$). This experiment was successfully completed in its entirety.