EE324: Controls System Lab

Experiment 2: Inverted pendulum

Team Members :
Prajwal Nayak (22B4246)
Aman Rishal C H (22B3914)
Hrishikesh S (22B4217)

Thursday Batch - Table 14

November 1, 2024

1 Aim

To design and implement control action for maintaining a pendulum in the upright position (even when subjected to external disturbances) through LQR technique in an Arduino Mega.

2 Objective of the Experiment

- To restrict the pendulum arm vibration (α) within $\pm$ 3°
- To restrict the base angle oscillation (θ) within $\pm$ 30°

3 Control Algorithm

3.1 Theory

3.1.1 Control Input Equation:

$$u(t) = -Kx(t)$$

In the context of the inverted pendulum:

- $u(t)$ represents the **control force (or torque)** applied to the cart or base of the pendulum. The goal of this input is to keep the pendulum upright.
- K is the **gain matrix** that multiplies the state vector to determine how much control force should be applied.
- $x(t)$ is the state vector of the system. For an inverted pendulum, the state vector could include:
  - The angle of the pendulum (α)
  - The angular velocity of the pendulum ($\dot{\alpha}$)
  - The position of the cart (θ)
  - The velocity of the cart ($\dot{\theta}$)

The control input $u(t)$ adjusts the force or torque to counteract any deviation in the state vector (i.e., the pendulum's angle, velocity, etc.) from its equilibrium point (upright position).

In simple terms, this equation tells us that if the pendulum starts to fall in one direction, the controller applies a force proportional to the current state of the system to bring it back upright.

3.1.2 Gain Matrix:

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

For the inverted pendulum system, K is a matrix that defines how sensitive the control force is to the different components of the state vector $x(t)$. The gain matrix K is derived based on system matrices and optimal control theory:

- R is related to how much "effort" or energy you want to minimize in the control input. For example, applying a large force requires more energy, so R helps to limit unnecessary control input.
- B is the input matrix, which defines how the control force affects the dynamics of the pendulum. For an inverted pendulum, this matrix could describe how applying force to the cart or torque to the base influences the angle and angular velocity of the pendulum.
- P is a matrix resulting from solving the **Riccati equation** (explained in the next section), which captures the optimal balance between keeping the pendulum upright and minimizing the control input.

The **gain matrix** K is crucial because it **determines how much force should be applied based on the current state** (e.g., how far the pendulum has deviated from the upright position and how fast it is moving).

3.1.3 Algebraic Riccati Equation (ARE):

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

In this equation, the matrices involved are directly related to the dynamics of the inverted pendulum:

- A is the **system matrix**, which represents how the state of the pendulum (angle, velocity, etc.) changes over time without any control input. For example, if left uncontrolled, the pendulum would naturally fall due to gravity.
- P is a **matrix** that is **determined by solving the Riccati equation**. It represents the trade-off between minimizing the error in the state (keeping the pendulum upright) and minimizing the control effort (using as little force or energy as possible).
- Q is a **weighting matrix** that penalizes the deviation of the pendulum from the upright position. It assigns a higher cost to states that represent larger deviations (like a big tilt or high angular velocity).

The Riccati equation is part of solving an optimal control problem. The goal is to find a control law (through the matrix P) that minimizes a cost function. The cost function balances the need to keep the pendulum upright (by penalizing deviations in angle and velocity) and the desire to use minimal force or energy in doing so.

4 Calculations

State Space Matrices :

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{rM_p^2 l_p^2 g}{J_p J_{eq} + M_p l_p^2 J_{eq} + J_p M_p r^2} & \frac{-K_t K_m (J_p + M_p l_p^2)}{(J_p J_{eq} + M_p l_p^2 J_{eq} + J_p M_p r^2) R_m} & 0 \\ 0 & \frac{M_p l_p g (J_{eq} + M_p r^2)}{J_p J_{eq} + M_p l_p^2 J_{eq} + J_p M_p r^2} & \frac{-M_p l_p K_t r K_m}{(J_p J_{eq} + M_p l_p^2 J_{eq} + J_p M_p r^2) R_m} & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ \frac{K_t(J_p+M_pl_p^2)}{(J_pJ_{eq}+M_pl_p^2J_{eq}+J_pM_pr^2)R_m} \\ \frac{M_pl_pK_tr}{(J_pJ_{eq}+M_pl_p^2J_{eq}+J_pM_pr^2)R_m} \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

After substituting the above values, the Matrices came out to be :

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 123.979 & -1.577 & 0 \\ 0 & 111.623 & -0.725 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 56.389 \\ 25.930 \end{bmatrix}$$

5 Results

K_θ	K_α	$K_{\dot{\theta}}$	$K_{\dot{\alpha}}$
-5.000	115.312	-4.0513	15.304

$$Q = \begin{bmatrix} 350 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} 14 \end{bmatrix}$$

6 Arduino Code

```
#include <SPI.h>
#define BAUDRATE          115200
#define AMT22_NOP         0x00
#define AMT22_RESET       0x60
#define AMT22_ZERO        0x70
#define NEWLINE           0x0A
#define TAB               0x09
#define RES12             12
#define RES14             14
#define ENC_0              2
#define ENC_1              7
#define SPI_MOSI          51
#define SPI_MISO          50
#define SPI_SCLK          52

#define motorPin1  12
#define motorPin2  11

int alpha_raw;
int theta_raw;
float alpha_rad;
float theta_rad;

float theta_prev = 0;
float alpha_prev = 0;
float theta_dot;
float alpha_dot;

int t_old = 0;
int t_new = 0;
int dt = 0;

float K_theta = -5.000000;
float K_alpha = 115.312753;
float K_theta_dot = -4.088818;
float K_alpha_dot =  15.304854;
```

```
37
38 float control_signal;
39 int motor_voltage = 0;
40 float scaling_factor = 3;
41 void setup()
42 {
43
44   pinMode(SPI_SCLK, OUTPUT);
45   pinMode(SPI_MOSI, OUTPUT);
46   pinMode(SPI_MISO, INPUT);
47   pinMode(ENC_0, OUTPUT);
48   pinMode(ENC_1, OUTPUT);
49
50   Serial.begin(BAUDRATE);
51
52   digitalWrite(ENC_0, HIGH);
53   digitalWrite(ENC_1, HIGH);
54   SPI.setClockDivider(SPI_CLOCK_DIV32);     // 500 kHz
55   //start SPI bus
56   SPI.begin();
57 }
58 void loop()
59 {
60   delay(5);
61   alpha_raw = getPositionSPI(ENC_0, RES14);
62   alpha_rad = convertToRadians(alpha_raw);
63
64   theta_raw = getPositionSPI(ENC_1, RES14);
65   theta_rad = convertToRadians(theta_raw);
66
67   t_new = millis();
68   dt = t_new - t_old;
69
70   theta_dot = (theta_rad - theta_prev)*1000 / dt;
71   alpha_dot = (alpha_rad - alpha_prev)*1000 / dt;
72
73   control_signal = -(K_theta * (theta_rad-255)
74   + K_alpha * (alpha_rad - 182) + K_theta_dot * theta_dot
75   + K_alpha_dot * alpha_dot);
```

```
76
77
78  float duty_cycle = control_signal / 905.0;
79  float c = 1;
80    if (duty_cycle >=c) {
81      duty_cycle = c;
82
83    } else if (duty_cycle <= -c) {
84      duty_cycle = -c;
85    }
86
87    if (duty_cycle > 0) {
88      analogWrite(motorPin2, duty_cycle * 255);
89      analogWrite(motorPin1, 0);
90    } else {
91      analogWrite(motorPin1, -duty_cycle * 255);
92      analogWrite(motorPin2, 0);
93    }
94
95    theta_prev = theta_rad;
96    alpha_prev = alpha_rad;
97    t_old = t_new;
98    Serial.print("Theta: ");
99    Serial.print(theta_rad, DEC);
100   Serial.write(NEWLINE);
101   Serial.print(" Alpha: ");
102   Serial.print(alpha_rad, DEC);
103   Serial.write(NEWLINE);
104     Serial.print(" Control Signal: ");
105   Serial.println( duty_cycle * 255);
106
107 }
108
109 float convertToRadians(int raw_value) {
110   float angle_deg = (raw_value) * (360.0 / 16384);
111   return angle_deg;
112 }
113
114 uint16_t getPositionSPI(uint8_t encoder, uint8_t resolution)
```

```
{
  uint16_t currentPosition;       //16-bit response from encoder
  bool binaryArray[16];           //after receiving the position we
                                  //will populate this array and
                                  //use it for
                                  //calculating the checksum

  currentPosition = spiWriteRead(AMT22_NOP, encoder, false) << 8;

void setCSLine (uint8_t encoder, uint8_t csLine)
{
  digitalWrite(encoder, csLine);
}
void setZeroSPI(uint8_t encoder)
{
  spiWriteRead(AMT22_NOP, encoder, false);
  delayMicroseconds(3);
  spiWriteRead(AMT22_ZERO, encoder, true);
  delay(250); //250 second delay to allow the encoder to reset
}

void resetAMT22(uint8_t encoder)
{
  spiWriteRead(AMT22_NOP, encoder, false);
  delayMicroseconds(3);
  spiWriteRead(AMT22_RESET, encoder, true);
  delay(250);
}
```

7 Problems Faced

1. This experiment was challenging in many ways as we were supposed to implement the LQR algorithm, which we were new to, as well as were required to take of rotation about two pivots.

2. After compiling the code successfully we were not able to make the motor to settle at the settling point, and it was behaving very randomly. We then proceeded to figure out the rest of the code, on the assumption that the zero of the Potentiometer will coincide with $0°\alpha$ then after a number of failed attempts to balance the Pendulum upright we realized that the zeroes on the Potentiometer and that of the Physical Angle ($\alpha = 0°$) are not matching. Thus we figured out the Zero for the Pendulum

manually by using the Serial Monitor of the Arduino IDE and then made the necessary tweaks in the code.

3. After figuring out and fixing the zero of the Pendulum, still we were unable to get satisfactory results for the balancing as the angles α and θ were out of required ranges of 3° and 30°.Thus we were required to optimize the parameter K for θ , α , $\dot{\theta}$ and $\dot{\alpha}$.After a lot of hit and trials, along with fine tuning, we were finally able get the Pendulum balance by itself and also resist any pertubations, with no small oscillations in the end and a stable steady state.

8 Results Obtained

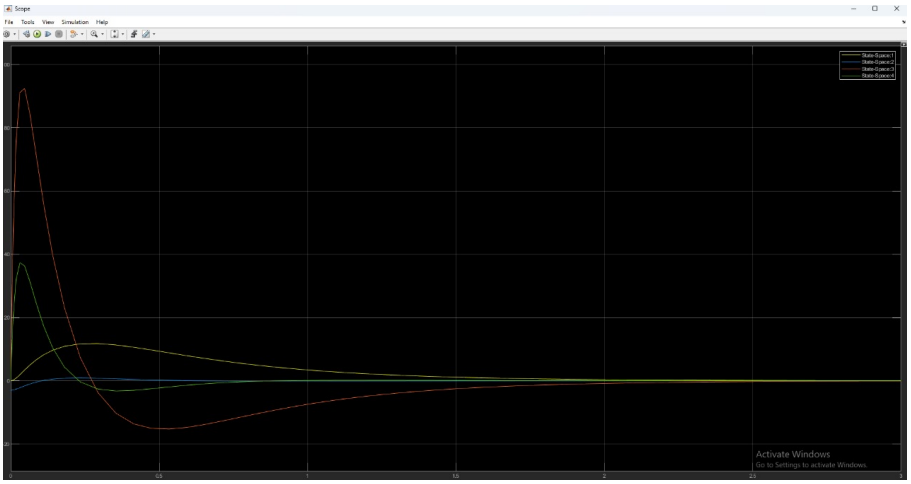

Figure 1

9 Observation and Inference

This experiment enabled us to know more about the LQR method and its applications.We were required to calculate the values of the matrices A and B manually which depends on the parameters of the Pendulum like Mass of Pendulum , Length of COM of pendulum from pivot ,etc.We then used Matlab for simulation in order to obtain the value of Q matrix by the LQR method.

After carefully selecting the parameters, we observed the impact of each of the constants on the behavior of the system.

- K_θ This factor influences the angle made by the support of pendulum about the Y - Axis,on assuming the Pendulum lies in the XY plane.
- $K_{\dot{\theta}}$
  This factor determines the rate at which the angle made by the support of the pendulum about the Y - Axis is changing with time.
- K_α The above factor influences the angle made by the pendulum with the Z - Axis
- $K_{\dot{\alpha}}$ This factor is responsible for determining the rate at which the angle made by the pendulum with the Z- Axis is changing