Control Systems Lab (EE324) Report

Team Members:
Utkarsh Maurya (22B3910)
Chinmay Tripurwar (22B3902)
Dishant (22B3984)
Group/Table Number: 4

September 11, 2024

Experiment No: 2
Title: Inverted pendulum

1 Objective

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.

- To restrict the pendulum arm vibration (α) within ± 3 degrees
- To restrict the base angle oscillation (θ) within ± 30 degrees.

2 Control Algorithm

The control algorithm used in this experiment is the Proportional control.

$$u(t) = -Kx(t) \tag{1}$$

$$K = R^{-1}B^TP \tag{2}$$

$$A^TP + PA - PBR^{-1}B^TP + Q = 0 \tag{3}$$

3 Challenges Faced and their Solutions

- Tuning the LQR Gains: Tuning the LQR gains (Q and R) to achieve optimal performance was a challenge, as it requires a trade-off between stability and performance.
- Meeting the Objectives: Meeting the objectives of restricting the pendulum arm vibration within ± 3 degrees and base angle oscillation within ± 30 degrees was a challenge, as it requires precise control and tuning of the algorithm.

4 Results

$$Q = \begin{bmatrix} 19 & 0 & 0 & 0 \\ 0 & 0.75 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$
$$R = \begin{bmatrix} 5.5 \end{bmatrix}$$

$$k_1 = -1.858$$
$$k_2 = 44.1596$$
$$k_3 = -1.433$$
$$k_4 = 5.718$$

5 Observations and Inference

- Initial Oscillations: When the pendulum is first released, it oscillates wildly, indicating a high degree of instability.
- High Sensitivity on K : The LQR controller's performance is highly sensitive to the value of the gain matrix K, with small changes in K leading to significant differences in closed-loop behavior.
- Convergence to Upright Position: As the LQR controller is activated, the pendulum gradually converges to the upright position, demonstrating the effectiveness of the control algorithm.
- Damping of Oscillations: The amplitude of the pendulum's oscillations decreases over time, indicating that the controller is successfully damping out the oscillations.
- External Disturbances: When external disturbances (e.g., tapping the pendulum) are applied, the pendulum deviates from the upright position, but the controller quickly corrects the deviation.
- Steady-State Error: After a few seconds, the pendulum settles into a steady-state position, with minimal oscillations, indicating that the controller has achieved a stable equilibrium.

5.1 Code

```
/* Initial Encoder Code */
#define motorPin1  8
#define motorPin2  9

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 = -1.858;
float K_alpha = 44.1596;
float K_theta_dot = -1.433;
float K_alpha_dot = 5.718;

float control_signal;
int motor_voltage = 0;

void setup(){
  //Set the modes for the SPI IO
}

void loop(){
  delay(5);

  // Read alpha (pendulum 1 angle) using SPI
  alpha_raw = getPositionSPI(ENC_0, RES14);
  alpha_rad = convertToRadians(alpha_raw);

  // Read theta (pendulum 2 angle) using SPI
  theta_raw = getPositionSPI(ENC_1, RES14);
  theta_rad = convertToRadians(theta_raw);

  t_new = millis();
  dt = t_new - t_old;

    // Calculate angular velocities (x. = dx/dt)
  theta_dot = (theta_rad - theta_prev)*1000 / dt;
```

```
  alpha_dot = (alpha_rad - alpha_prev)*1000 / dt;

  // Calculate control signal using LQR (u = -kx)
  control_signal = -(K_theta * theta_rad + K_alpha * (alpha_rad-180)
      + K_theta_dot * theta_dot + K_alpha_dot * alpha_dot);

float duty_cycle = control_signal / 905.0;
float c = 1;
  if (duty_cycle >=c) {
    duty_cycle = c;

  } else if (duty_cycle <= -c) {
    duty_cycle = -c;
  }

  if (duty_cycle > 0) {
    analogWrite(motorPin2, duty_cycle * 255);
    analogWrite(motorPin1, 0);
  } else {
    analogWrite(motorPin1, -duty_cycle * 255);
    analogWrite(motorPin2, 0);
  }

  // Save current angles for the next iteration
  theta_prev = theta_rad;
  alpha_prev = alpha_rad;
  t_old = t_new;
}

float convertToRadians(int raw_value) {
  float angle_deg = (raw_value) * (360.0 / 16384);
  return angle_deg;
}
/* Rest of Encoder Code */
```