Experiment 3

LQR Control of Inverted Pendulum

Shambhavi Shanker, 21D070066
Aditya Anand, 21d070007
Natasha Ramineni, 21d070046

October 2023

Contents

1 Aim

The aim of this experiment was to design and implement an LQR Controller on an Arduino Mega 2560 so as to balance a pendulum in an upright position without being affected by external disturbance with these specifications:

- The angle between the topmost position and the pendulum (α) to be less than 3° in both directions.
- The angle between the arm and the original position of the arm to be less than 30° in both directions.

2 Procedure

We first went through the concepts of LQR (linear quadratic regulator) required to start working on this experiment. The LQR algorithm essentially finds the control input u(t) that minimizes the quadratic cost function J (equation 2) for a given LTI system (equation 1). Here Q is a 4 by 4 matrix and R is a 1 by 1 matrix and they are set by the user to obtain the optimal control gain K, where u = -Kx.

$$\frac{d}{dt}x = Ax + Bu \tag{1}$$

$$J = \int_0^\infty x^T Q x + u^T R u \, dt \tag{2}$$

So implementing LQR on MATLAB then came down to filling the values of Q and R and finding the corresponding control gain K such that the parameters θ and α were restricted to the given specifications.
The K obtained through MATLAB simulation was used in the Arduino code to give us the control input to be given to the inverted pendulum setup, the state system of which is given by $x = [\theta, \theta', \alpha, \alpha']$.
We then uploaded the code on the Arduino board and ran the setup and iteratively tweaked the values of Q and R in accordance with the behaviour of the pendulum.

3 Devised Algorithm and Relevant Code Snippets

- We used the openly available MATLAB code for finding K through the LQR function for the values of A, B, C, and D that were given in the lab manual and arbitrary values for Q and R. We looked at the step

response graph of the system to see whether θ and α were restricted to the specifications.

- We were given the sample code for the setup, which we used to first print out the equilibrium points for θ and α and to look at the region where the system was non-linear. The equilibrium state came out as $x_{eq} = [9290, 0, 8250, 0]$. The PWM values were normalized to radians by multiplying by the scaling factor of $\frac{\pi}{8192}$.
- The derivatives of θ and α were computed as :

```
prev_time=time;
time=(double)millis(); // typecast
dt=(time-prev_time)/1000.0;
theta_dot=(theta-prev_theta)/dt;
alpha_dot=(alpha-prev_alpha)/dt;
prev_alpha=alpha;
prev_theta=theta;
```

- The control input u = -Kx to be fed to the motor running the pendulum was computed as:

```
output=-1*(K[0]*(theta-theta_eq)+K[1]*(alpha-
alpha_eq)+K[2]*theta_dot+K[3]*alpha_dot);
////Serial.print("Output: ");
modo=(abs(output));
Serial.println((uint16_t)min((uint16_t)modo+80,240));
analogWrite(6, (uint16_t)min((uint16_t)modo+120,240));
```

where K was initialized with the appropriate values as:

```
double K[]={-7.51879699248072 , 142.898666477457,
-4.67006186366597, 16.7794291146019};
```

- The sign of the control input was used to set the direction of motion of the pendulum so that it would always go to the equilibrium points. The code is as follows:

```
if( output > 0){
  digitalWrite(24 ,HIGH);
  digitalWrite(26, LOW);
}
```

```
else{
  digitalWrite(26 ,HIGH);
  digitalWrite(24, LOW);
}
```

- The final values of Q and R that gave us the optimal K value were found using Bryson's rule which uses the inverse of variance for each parameter as the first guess. The values were:

$$\text{Q} = \begin{bmatrix} 0.0014 & 0 & 0 & 0 \\ 0 & 0.1322 & 0 & 0 \\ 0 & 0.0001 & 0 & 0 \\ 0 & 0 & 0 & 0.0004 \end{bmatrix} \quad \text{R} = 1/(200)^2$$

Bryson's Rule: *A reasonable simple choice for the matrices Q and R is given by the Bryson's rule where we select Q and R diagonal as*

$$Q_{ii} = \frac{1}{maximum\,acceptable\,value\,of\,z_{ii}^2} \quad \text{i=(1,2,..)},$$

$$R = \frac{1}{maximum\,acceptable\,value\,of\,u}$$

In essence, the Bryson's rule scales the variables that appear in J_{LQR} so that the maximum acceptable value for each term is one. This is especially important when the units used for the different components of u and z make the values for these variables numerically very different from each other.

4 Results and Observations

- Using our algorithm the pendulum was able to balance under the required specifications and it balanced until external pushes drove it into its non-linear region
- Change in delays could cause major change in balancing. Even adding or removing print lines could effect the balancing.
- The maximum θ that the arm reached was 22.5° and the maximum α that the pendulum vibrated till was under 3°.
- The pendulum was able to balance itself after giving it soft external pushes.

5 Problems Faced

- The pendulum would always go in the opposite direction that it was supposed to go in and hence would never balance. To fix this we found out that we had to comment a few lines of code and had to call the delay function only once in the entire loop to make sure the output was updated faster. The serial print functions were also giving us a problem with delays, which is why we had to use them only when required.
- The pendulum would sometimes go into the non-linear region for particular values of K even though it could balance somewhat. So pushing it a little made it jerky and not able to balance. Eventually we did find the proper value of K.
- The base of the setup was not stable which was also why for different positions of the base the same values of K weren't working in the same way.

6 Arduino code

```
/* Include the SPI library for the arduino boards */
#include <SPI.h>
/* Serial rates for UART */
#define BAUDRATE        9600
/* SPI commands */
#define AMT22_NOP       0x00
#define AMT22_RESET     0x60
#define AMT22_ZERO      0x70
/* Define special ascii characters */
#define NEWLINE         0x0A
#define TAB             0x09
/* We will use these define macros so we can write code once  /*compatible with 12
#define RES12           12
#define RES14           14
/* SPI pins */
#define ENC_0            2
#define ENC_1            3
#define SPI_MOSI        51
#define SPI_MISO        50
#define SPI_SCLK        52
#define enable 6
#define dir1 26
```

```
#define dir2 24
void setup()
{
  //Set the modes for the SPI IO
  pinMode(SPI_SCLK, OUTPUT);
  pinMode(SPI_MOSI, OUTPUT);
  pinMode(SPI_MISO, INPUT);
  pinMode(ENC_0, OUTPUT);
  pinMode(ENC_1, OUTPUT);
  //Initialize the UART serial connection for debugging
  Serial.begin(BAUDRATE);
  //Get the CS line high which is the default inactive state
  digitalWrite(ENC_0, HIGH);
  digitalWrite(ENC_1, HIGH);
  SPI.setClockDivider(SPI_CLOCK_DIV32);    // 500 kHz
  //start SPI bus
  SPI.begin();
}
void loop()
{
  //create a 16 bit variable to hold the encoders position
  uint16_t encoderPosition0;
  uint16_t encoderPosition1;
  uint8_t attempts;
  //initializing variables
  double theta = 0;
  double alpha = 0;
  double alpha_dot=0;
  double theta_dot=0;
  double prev_alpha=0;
  double prev_theta=0;
  double output;
  double wakt=0;
  double prev_wakt=0;
  double dt; //time difference
  double modo; //|outtput|
  double K[]={-7.51879699248072 , 142.898666477457,  -4.67006186366597,
  16.7794291146019}; #defining K value
  double theta_eq=9290*(PI/8192); //equilibrium value
  double alpha_eq=8250*(PI/8192); //equilibrium value
```

```
  while(1)
  {
    attempts = 0;
    encoderPosition0 = getPositionSPI(ENC_0, RES14);
    while (encoderPosition0 == 0xFFFF && ++attempts < 3)
    {
      encoderPosition0 = getPositionSPI(ENC_0, RES14); //try again
    }
    //////////again for second encoder////////////////////////////
    attempts = 0;
    encoderPosition1 = getPositionSPI(ENC_1, RES14);
    while (encoderPosition1 == 0xFFFF && ++attempts < 3)
    {
      encoderPosition1 = getPositionSPI(ENC_1, RES14); //try again
    }
    //encoder0 and 1 readings give us current theta and alpha
    theta=((uint16_t)encoderPosition1)*(PI/8192);
    alpha=((uint16_t)encoderPosition0)*(PI/8192);
    prev_wakt=wakt; // updating previous time
    wakt=(double)millis(); //reading current time
    dt=(wakt-prev_wakt)/1000.0;
    theta_dot=(theta-prev_theta)/dt; //derivative
    alpha_dot=(alpha-prev_alpha)/dt;
    //updating previous alpha and theta
    prev_alpha=alpha;
    prev_theta=theta;
    output=-1*(K[0]*(theta-theta_eq)+K[1]*(alpha-
    alpha_eq)+K[2]*theta_dot+K[3]*alpha_dot);
    modo=(abs(output)); //absolute output
    Serial.println((uint16_t)min((uint16_t)modo+8
    0,240));
    analogWrite(6, (uint16_t)min((uint16_t)modo+80,240)); //writing the
    //control onto enable
     if( output > 0){
      digitalWrite(24 ,HIGH); digitalWrite(26, LOW); }
     else{
      //  output = -1; -> CW
      digitalWrite(26 ,HIGH); digitalWrite(24, LOW);}
  }
}
```