INVERTED PENDULUM USING LQR CONTROLLER

PROJECT REPORT

EKLAVYA MENTORSHIP PROGRAMME

AT

SOCIETY OF ROBOTICS AND AUTOMATION, VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE, MUMBAI -19

ACKNOWLEDGEMENT

We are extremely grateful to our mentors Jash Shah, Ayush Kaura and Chinmay Lonkar for their support and guidance throughout the duration of the project. We would also like to thank all the members of SRA VJTI for their timely support as well as for organizing Eklavya and giving us a chance to work on this project.

Mahesh Tupe
tupemahesh91@gmail.com

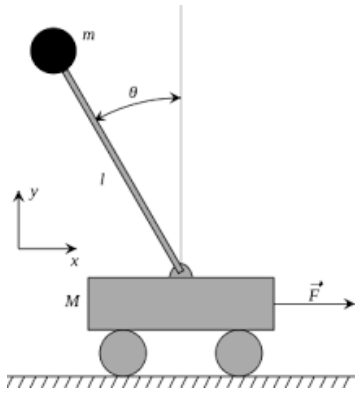
Vedant Nimje
vedantnimjed@gmail.com

TABLE OF CONTENTS

1. INTRODUCTION

- Goal

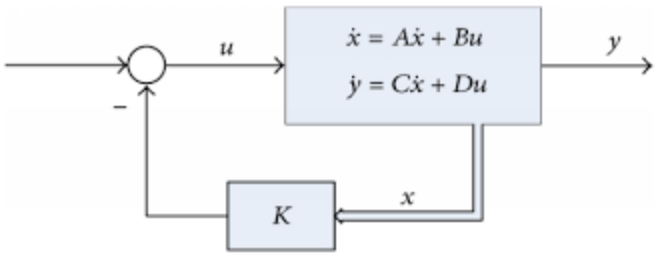
  The aim of this project is to implement LQR (Linear Quadratic Regulator) control and apply it to the classical problem of **Inverted Pendulum on Rails**, in both simulation (Octave) and hardware.



- What are Control systems?

  A control system is defined as a system of devices that manages, commands, directs, or regulates the behavior of other devices or systems to achieve a desired result. A control system achieves this through control loops, which are a process designed to maintain a process variable at a desired set point.
  Basically a control system is a system to control other systems.
  In our case, we used **LQR (Linear Quadratic Regulator)**.

  - Linear Quadratic Regulator: The theory of optimal control is concerned with operating a dynamic system at minimum cost. The case where the system dynamics are described by a set of linear differential equations and the cost is described by a quadratic function is called the LQ problem. That problem is solved using the LQR control.

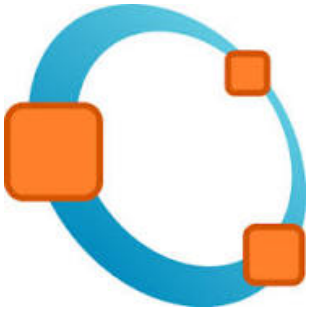


- Embedded systems

  Embedded C programming plays a key role in performing specific functions by the processor. In day-to-day life we use many electronic devices such as mobile phones, washing machines, digital cameras, etc. These all devices working are based on microcontrollers that are programmed by embedded C.

- GNU Octave

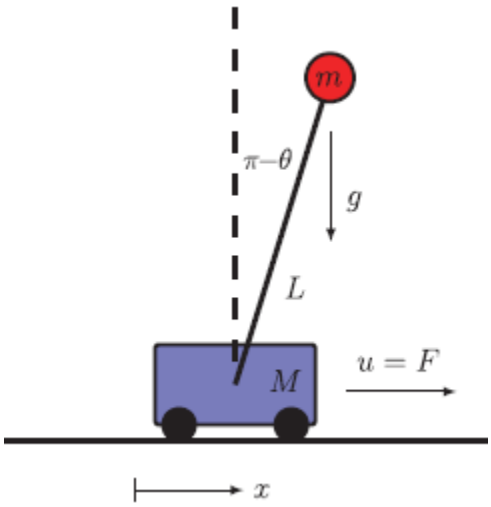
  GNU Octave is a high-level programming language primarily intended for scientific computing and numerical computation. Octave helps in solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with MATLAB. As part of the GNU Project, it is free software under the terms of the GNU General Public License. We used Octave to model and simulate the system using its various libraries.



2. MODELING AND SIMULATING THE SYSTEM IN OCTAVE

- **Calculating the dynamics equations**

  Considering the following representative diagram of our system, where M is the weight of the cart, m is the weight of the bob, g is acceleration due to gravity and L is the length of the rod.



  Our state space variables are Θ (deviation from lower normal), x (displacement from initial position), v (velocity) and ω (angular velocity of rod about pivot point).

  Now, to find the dynamics equations of motion, we made use of the **Euler–Lagrange method**.

  - Euler-Lagrange method

    It states that the equations of motion of a system can be obtained by solving the equation:

$$\frac{\partial L}{\partial f} - \frac{d}{dt}\frac{\partial L}{\partial f'} = 0$$

    where f is a state variable, and **L is called the Lagrangian**, which is the difference between the Kinetic and Potential energies of the system.\
    Hence, L = K.E - P.E

    The equations we got after solving this equation are

$$\dot{x} = v$$

$$\dot{v} = \frac{-m^2L^2gcos(\theta)\sin(\theta) + mL^2(mL\omega^2\sin(\theta) - dv) + mL^2u}{mL^2(M + m(1 - cos(\theta)^2))}$$

$$\dot{\theta} = \omega$$

$$\dot{\omega} = \frac{(m+M)mglsin(\theta) - mlcos(\theta)(mL\omega^2\sin(\theta) - dv) + mlcos(\theta)u}{mL^2(M + m(1 - cos(\theta)^2))}$$

- Representation of system in Matrix form

  To represent the system in matrix form, we first **linearise** it around an equilibrium point. For our case, we chose the point (π,0) and calculated the jacobian matrix. Then, we substituted the values of the unstable point in the jacobian, to get

$$\frac{d}{dt}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}=\begin{bmatrix}0 & 1 & 0 & 0\\ 0 & \frac{-d}{M} & \frac{bmg}{M} & 0\\ 0 & 0 & 0 & 1\\ 0 & \frac{-bd}{ML} & \frac{-b(m+m)g}{ML} & 0\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}+\begin{bmatrix}0\\ \frac{1}{M}\\ 0\\ \frac{b}{ML}\end{bmatrix}u,\ for\ \begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}=\begin{bmatrix}x\\v\\\theta\\\omega\end{bmatrix}$$

or

$$\dot{x} = Ax + Bu$$

- **LQR Controller**

  In LQR, the feedback matrix, K, is calculated by minimizing the cost function

$$J = \frac{1}{2}\int_0 \left(x^{\mathrm{T}}Qx + u^{\mathrm{T}}Ru\right)\mathrm{d}t$$

  Here, the **Q** matrix is a diagonal matrix, in which each of the entries corresponds to **the penalty that should be applied when a particular state variable deviates from the intended set point**. Similarly, the **R** matrix imposes **a penalty on the inputs of the system**. In our case, we want to the keep angular deviation of rod, Θ as minimum as possible, but the position x of the cart is not that important. So, our Q matrix looks like

$$Q = \begin{bmatrix}100 & 0 & 0 & 0\\ 0 & 10 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

- **Modeling the system in Octave**

  Now that we have all the equations and matrices, we can put them and model the system in Octave, by representing the system in form of matrices as

$$\dot{x} = Ax + Bu$$

```
function dy = inverted_pendulum_dynamics(y, m, M, L, g, u)
  sin_theta = sin(y(1));
  cos_theta = cos(y(1));
  f = 5;
  dy(1,1) = y(2);
  dy(3,1) = y(4);
  dy(4,1) = (-(m*g*cos_theta*sin_theta) + (m*L*sin_theta*((y(2))^2) - f*y(4)) + u)/(M + m*(sin_theta^2));
  dy(2,1) = (-(g*sin_theta) + (dy(4,1)*cos_theta))/L;

endfunction

function [A,B] = inverted_pendulum_AB_matrix(m, M, g, L)
  f = 0.8;
  A = [0 1 0 0; (g+(m*g/M))/L 0 0 -f/(M*L); 0 0 0 1; -m*g/M 0 0 -f/M];
  B = [0; -1/(L*(M)); 0; 1/(M)];
endfunction
```

Following are some of the libraries and functions that we used in Octave to implement the same

- Octave packages

Package name	Function
1. Symbolic	Adds symbolic calculation features to GNU Octave. These include common Computer Algebra System tools such as algebraic operations, calculus, equation solving, Fourier and Laplace transforms and other features.
2. Control	Used for accessing various algorithms. In our case, we used it for accessing the function lqr()

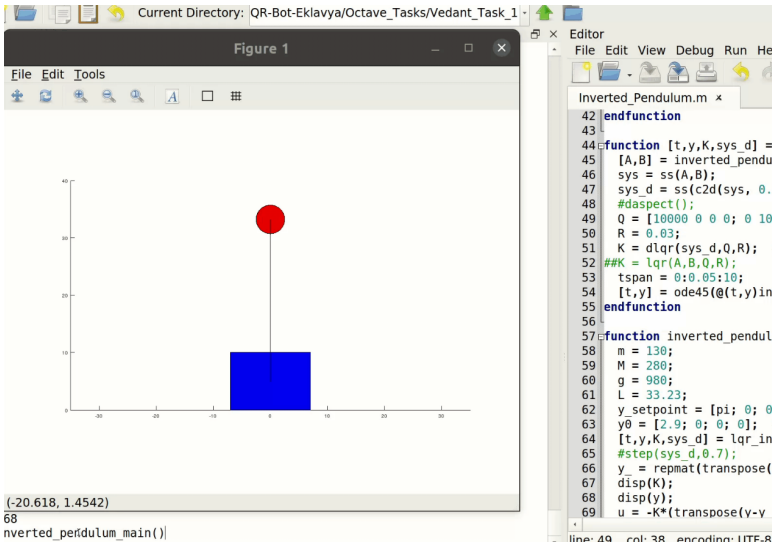
- Important functions used

Function name	Application
1. ss(A,B,C,D)	Creates a state space model
2. c2d(sys, t)	Converts a continuous model into a discrete-time equivalent
3. dlqr(sys_d,Q,R)	Applies LQR to a discrete-time system, and returns the feedback matrix, K
4. step(sys_d)	Gives the step response of a system

- **Simulation**

  To simulate the system, we first had to solve the differential equations and get the values of all the system variables in each time step over the given time interval. Then, using various features in Octave, we graphically built the system.

  Representative image of the system

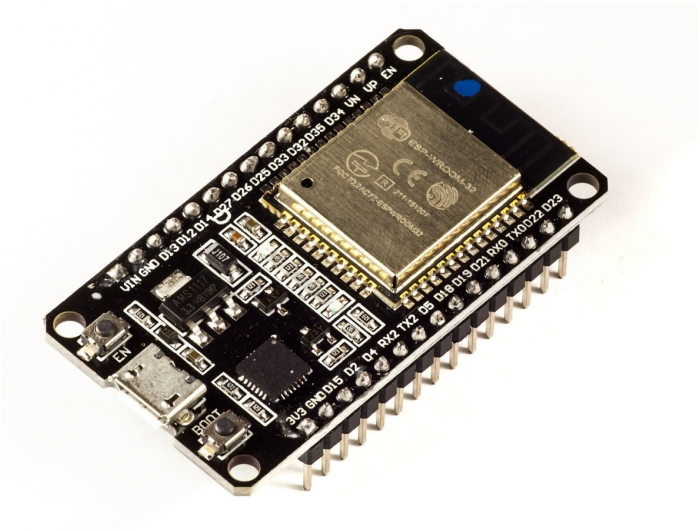


  Link for GIF of the simulation:
  https://user-images.githubusercontent.com/103848930/193095170-ae7c0de0-c5a4-43df-be26-f1580bf4394c.gif

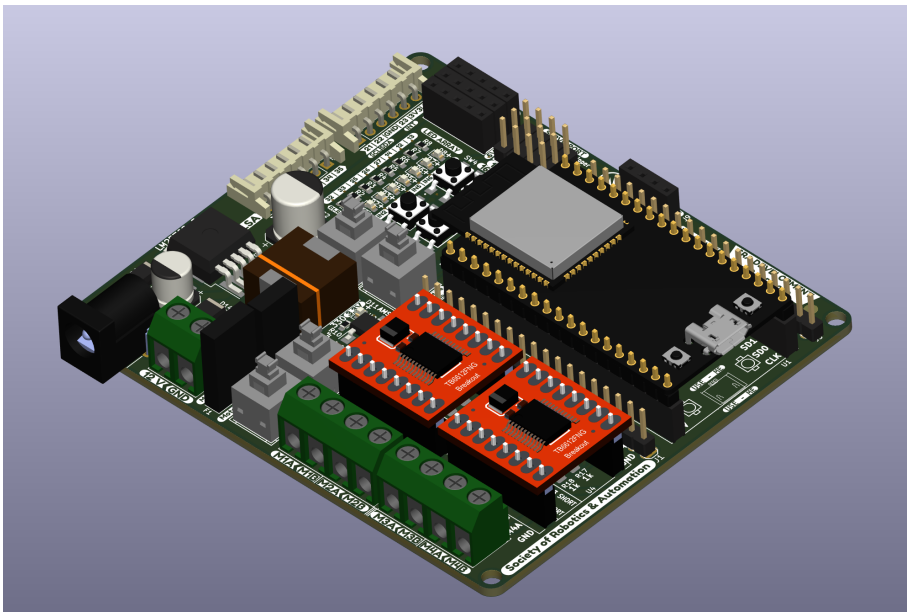
3. Hardware Implementation

- **ESP32 microcontroller and SRA board**

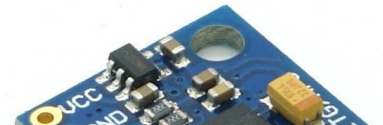
  The **ESP32** is a very versatile System On a Chip (SoC) that can be used as a general purpose microcontroller with quite an extensive set of peripherals including WiFi and Bluetooth wireless capabilities.



  The **SRA board** is a development board based on ESP32 with on-board peripherals like programmable LEDs, switches, sensor ports for Line Sensor Array and MPU-6050, protection circuit for overcurrent and reverse voltage and motor drivers.



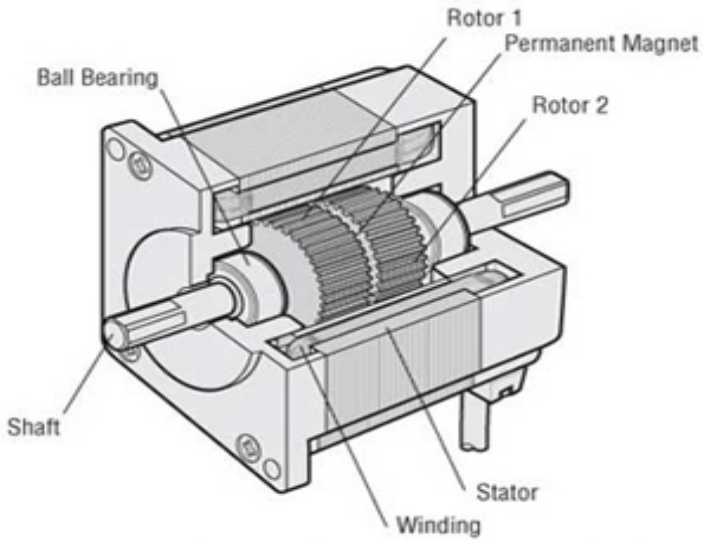
- **MPU-6050 (IMU)**



MPU6050 sensor module is a complete 6-axis Motion Tracking Device. It combines a 3-axis Gyroscope, 3-axis Accelerometer and Digital Motion Processor all in a small package.It has an **I2C bus interface** to communicate with the microcontrollers.

- ## Stepper Motor and DRV8825 Motor Driver

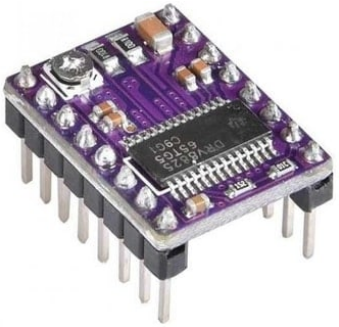
Stepper motors are DC motors that move in discrete steps. They have multiple coils that are organized in groups called "phases". By energizing each phase in sequence, the motor will rotate, one step at a time. With a computer controlled stepping you can achieve very precise positioning and/or speed control



Motor Structural Diagram: Cross-Section Parallel to Shaft

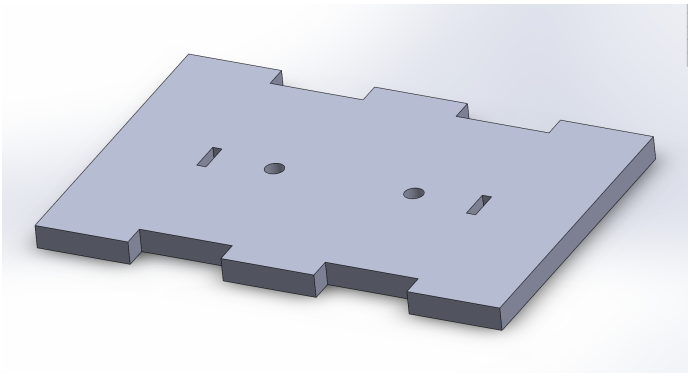
In our case, the stepper motor is rotated by providing required frequencies through the **LEDC peripheral of ESP32**. As ESP32 is also being used to calculate required frequency and reading MPU readings we needed a stepper driver to control the stepper motor.

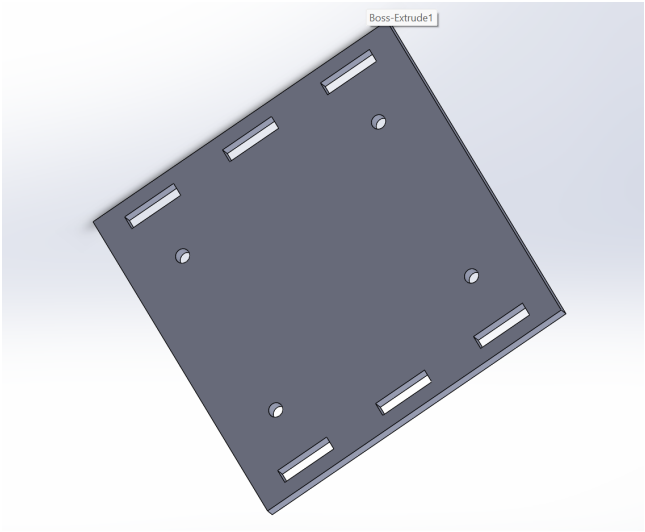
We are using **DRV8825 stepper driver** which is a complete Microstepping Motor Driver with a built-in translator for easy operation. It can deliver up to approximately 1.5 A Current per phase without a heat sink or forced airflow. It is rated for 2.2 A Current per coil with sufficient additional cooling.

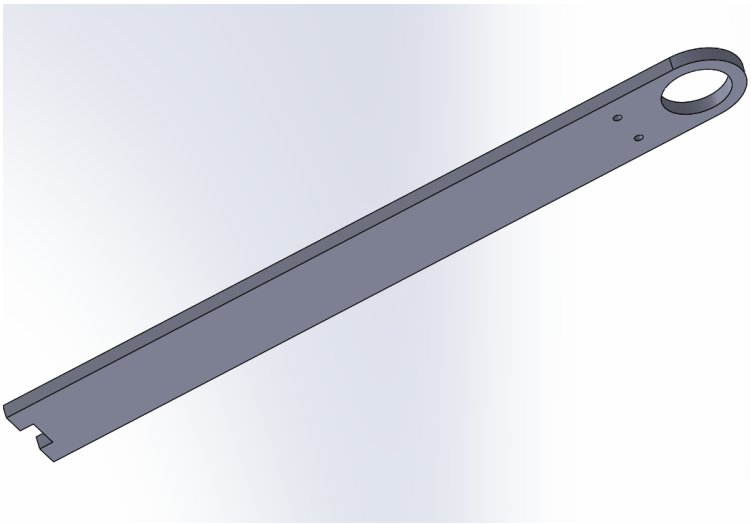


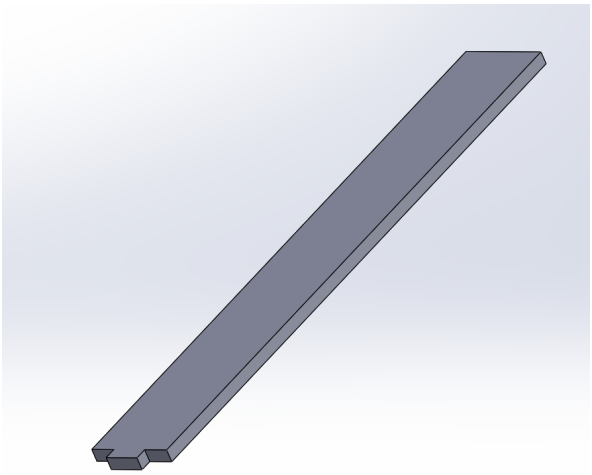
- ## Designing in SOLIDWORKS

To implement this project in hardware we needed a cart which could move on pre-existing rails and a rod like structure to be used as a pendulum. We designed these parts in Solidworks and had them laser cut.



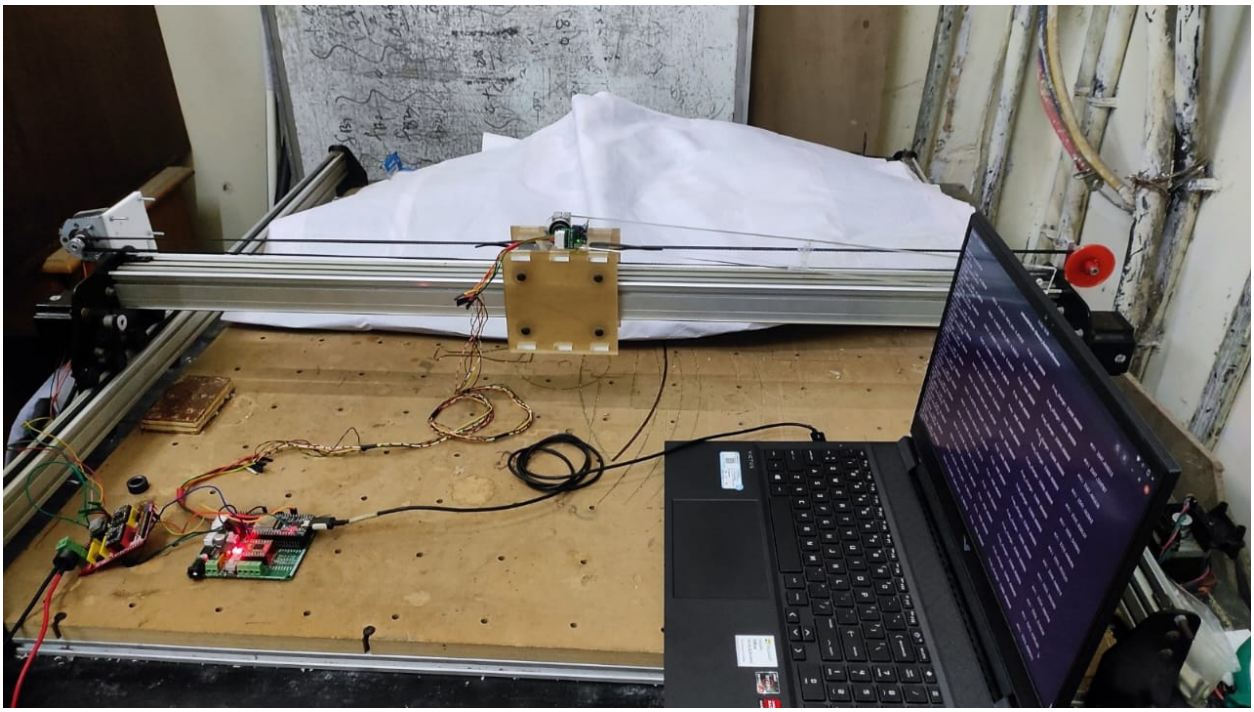


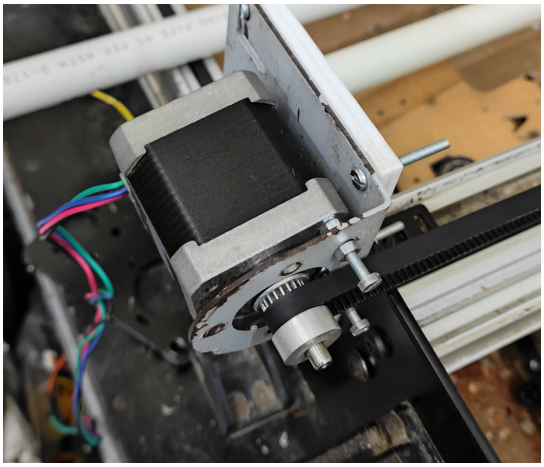


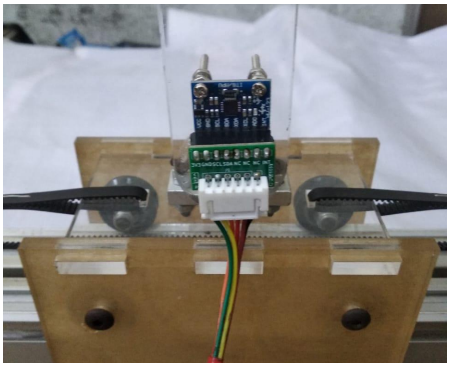


- **Assembly**

We started with assembling the cart and sliding it on rails. Stepper motor and ideal pulley are mounted on either side of the rails. The MPU6050 is mounted on the rod, with the help of screws, near the pivot point. The rod itself contains a bearing, which allows it to rotate about the screw. The screw is held in place on the cart by means of a mount.







- **Problems faced during testing**

    - Delay in MPU readings

        In initial firmware code reading MPU values and running stepper motor was being done in a single task. This caused delay in MPU readings as delay intended for stepper motor was affecting MPU readings. This delay in MPU readings further caused slow response from the motor.

        To solve this problem reading MPU values and running stepper motor is now split into two different tasks. This reduces the delay significantly. The delay which remains is inherent in the Complementary filter method of MPU6050. To reduce this delay even further we plan to use the DMP(Digital Motion Processor) of MPU6050 in future.

        More information on DMP and its advantage over complementary filter can be found here

    - Interruption in MPU readings

        MPU works on I2C protocol. It is vulnerable to electromagnetic interference. Using high frequency SMPS will give the following error.

        ```
        i2cdev could not read from devices 0x68
        ```

        To solve this issue, we used a 12V battery rather than a 12V smps.

    - Varying tension in belt

        Initial hardware setup consisted of a timing belt in trapezoidal shape, its top side being the base of the cart. Whenever the cart moved to extreme ends of rails the tension in the belt increased. Increase in tension was enough to break the acrylic cart during testing.

Simple fix to this problem is using a belt in a rectangular shape so that movement of the cart doesn't change tension in the belt.

- **Results**

  Currently, the system we have built is still not able to self-balance the pendulum, and requires one of us to hold it by hand. The motor is giving the required response, but falls short in terms of acceleration, or is a little slow to give it.

  Link for the demo video:
  https://drive.google.com/file/d/1kR3V9MuX7RSvx0oKI2xWOUD_GPdfEYIV/view?usp=sharing

4. Conclusion and Future work

- ## Conclusion

  We successfully modeled and simulated an inverted pendulum on Octave. Even though we were not able to successfully implement it on hardware, we understood a lot about control systems and realized that hardware implementation of any project is always harder than software implementation.

- ## Future work

  As current hardware implementation is not good enough, the first task would be to improve it. Additional things we could implement are:

  1. Swing up controller**:** LQR on its own is not capable of handling high deviation of pendulum as the system has been linearized about the unstable equilibrium point, (π,0). In future we would like to add a swing up controller which can handle shortcomings of LQR.

     Video reference:
     Swing-up and balancing control of an inverted pendulum mech…

  2. Self balancing bot**:** A self balancing bot is very similar to an inverted pendulum. Creating a self balancing bot using LQR would be a future task.

5. References

- Control Bootcamp: Overview
- Essence of linear algebra preview
- https://www.ato.com/how-to-prevent-stepper-motor-losing-step
- Electronic Basics #24: Stepper Motors and how to use t…
- https://newscrewdriver.com/2021/02/25/esp32-exercise-stepper-motor-pulses-with-ledc-pwm/
- https://docs.espressif.com/projects/esp-idf/en/latest/esp32/
- https://www.instructables.com/Inverted-Pendulum-Control-Theory-and-Dynamics