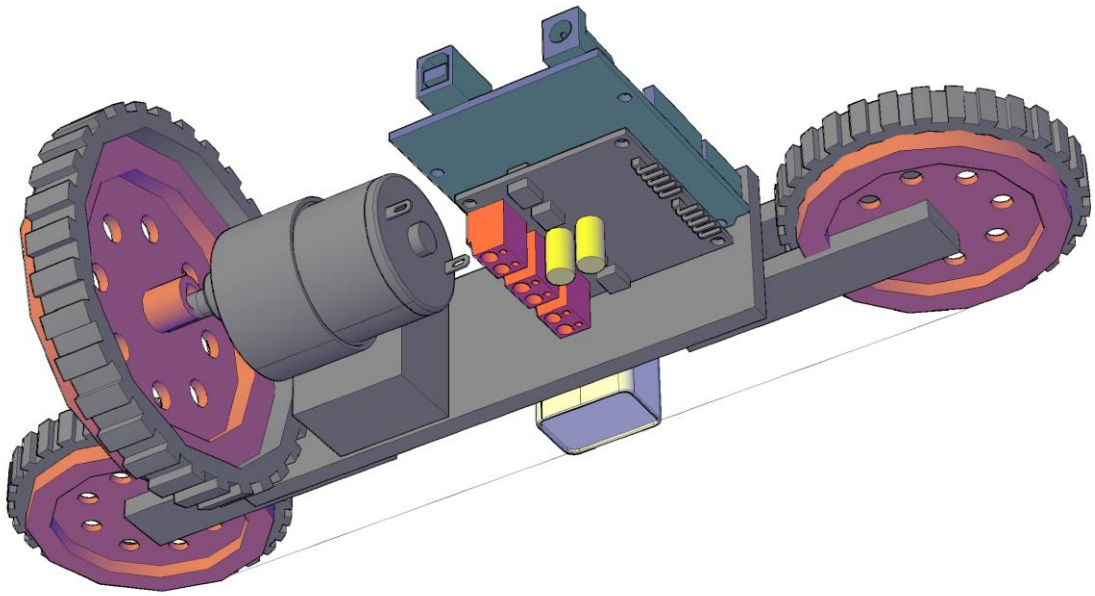
ME2400

Self-Balancing Cycle

Group No. 110 - ME16B014, ME16B045, ME16B070, ME16B076, ME16B112



PROBLEM STATEMENT

- To create a model cycle which can balance itself against gravity.
- It should as well balance itself against external disturbances like pushing.

LEARNING OBJECTIVE

- To identify and do research on the components required for balancing the cycle.
- To implement a Proportional-integral-derivative controller for balancing the cycle by taking input of the tilted angle and giving output as required voltage to spin a flywheel.

COMPONENTS USED

- GYI-521 MPU 6050
- Arduino UNO
- Motor (12volt, 1000rpm, 2Kg-cm torque)
- Heavy Flywheel (8cm diameter)
- Motor Driver L293D
- 12v Battery source (battery eliminator and 12volt batteries)
- Battery source for Arduino (9v battery or laptop)
- LM7805 (Voltage regulator for arduino)
- Motor Clamp
- Plastic Wheels
- Miscellaneous

DESIGN OF THE SYSTEM

System Modelling

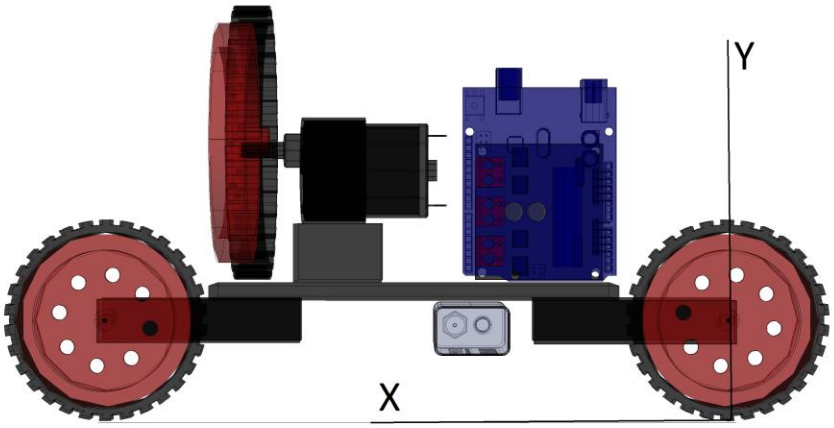


Fig1: The origin and the axes of the cycle

- The Moment of Inertia (I) and Center of Gravity (G) of the system are obtained from the CAD model.

- The CAD model does not include the jumper wires, glue and tapes we have used in the real model. Including them makes the analysis of I and G out of scope, due to their complex shapes and orientations. This gives rise to tolerance in the I and G values.
- The material used in CAD model and the materials used in real model need not be exactly the same. This again is a source error in the analysis.

Notations used:

- Let I_s be the Moment of Inertia of the cycle system about X axis
- Let I_f be the Moment of Inertia of the flywheel about the X axis
- Let ω be the angular velocity of the flywheel
- Let θ be the angle between the vertical and the cycle
- Let m be the mass of the whole system
- Let g be the acceleration due to gravity

The torque due to gravitational force is $mgGsin(\theta)$.The torque exerted on the flywheel by the cycle system, and the reaction torque exerted by the flywheel on the system is $I_f\frac{d\omega}{dt}$.
By torque balance we get the equation,

$$I_s \frac{d^2\theta}{dt^2} = mgGsin(\theta) - I_f \frac{d\omega}{dt}$$

For small angles of $\theta, sin(\theta) = \theta$

The output of our control system is voltage given to DC motor and the input is the angle tilted by the cycle. To do this, we need to understand the working of a DC motor. The equivalent circuit of the DC motor is as follows:

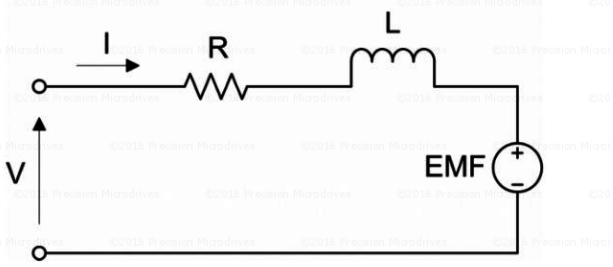


Fig2: Equivalent circuit of a DC motor

$$V = IR + L\frac{dI}{dt} + E \quad E = back\ emf$$

At the steady state, the change in current is negligible. Hence the effect of inductor can be ignored. Also at steady state, the torque produced by the motor and the load can be considered as the same. The torque produced by the motor is given by:

$$T = K_t \varphi I$$

Where $K_t$represents a constant inherited from it's internal design, $\varphi$is the total flux and *T* represents the load torque. Substituting the value of *I* in the previous equation, we obtain:

$$V = \frac{T}{K_t \varphi} R + E$$

$$E = K_E \varphi n \quad K_E = constant, \qquad \varphi = flux, n = speed\ of\ motor$$

$$V = \frac{T}{K_t \varphi} R + K_E \varphi n$$

The values of $K_E$and K_t are nearly equal. Also the flux produced in the motor is usually a constant. Hence we can assume $K_E \varphi = K_T \varphi = k$. We can also replace *n* by ω. The equation obtained is:

$$\omega = \frac{V}{k} - \frac{T}{k^2} R$$

Substituting this equation in the torque balance equation, we obtain:

$$I_s \frac{d^2\theta}{dt^2} = mgGsin(\theta) - I_f \frac{d}{dt}(\frac{V}{k} - \frac{T}{k^2}R)$$

$$I_s \frac{d^2\theta}{dt^2} = mgGsin(\theta) - I_f \frac{dV}{dt}\frac{1}{k}$$

Now doing Laplace transform of the equation, with $\theta(t=0)=0$, $\frac{d\theta}{dt}(t=0)=0$ and $\omega(t=0)=0$. The Laplace transform have been represented as:-

$$\Theta(s) = Laplace\ transform\ of\ \theta, V(s) = Laplace\ transform\ of\ V$$

$$I_s s^2\Theta(s) = mgG\Theta(s) - \frac{I_f}{k}sV(s)$$

$$\frac{I_f}{k}sV(s) = mgG\Theta(s) - I_s s^2\Theta(s)$$

$$V(s) = \frac{(mgG - I_s s^2)\Theta(s)}{I_f s/k}$$

$$\frac{V(s)}{\Theta(s)} = \frac{(mgG - I_s s^2)}{I_f s/k}$$

The open loop transfer function is $\frac{mgG - I_s s^2}{I_f s/k}$.

Iteration 1:

- The chassis was made using wood. A platform of dimensions **14cm x 4cm x 0.6cm** was used to house all components. The wheels were attached using 4 planks (2 planks per wheel) of dimensions **2cm x 7cm x 0.6cm** to the chassis. The wheels were attached using nuts and screws and attached to the platform using fevikwik.
- A cylindrical metallic flywheel of diameter 4.9cm and thickness 1.6cm was used as a flywheel.
- Motor of 1000rpm and 2kg-cm torque was used to drive the flywheel. The motor-flywheel setup was attached to the platform using m-seal.
- A breadboard was attached at the bottom side of the platform, which housed the gyroscope.
- The arduino and motor driver were attached vertically to the platform behind the motor.
- The motor driver was powered using a 12volt eliminator and the arduino was powered using laptop.

- The circuit is shown below:

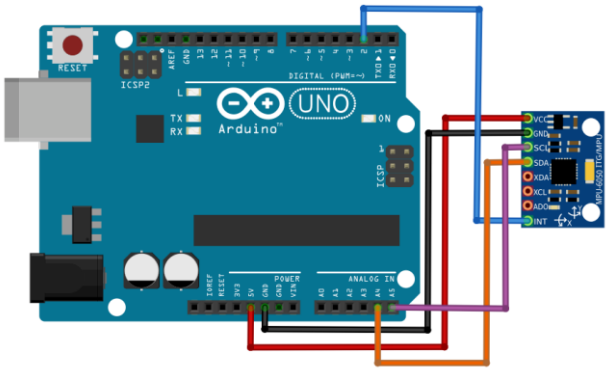


Fig3: Connections for MPU6050

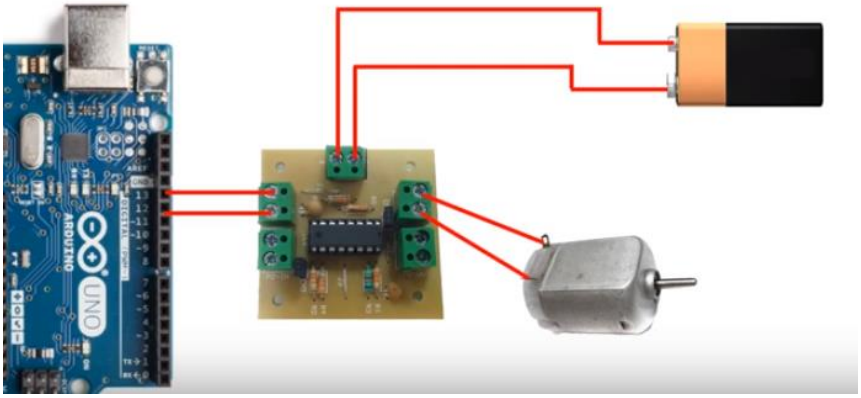


Fig4: Connections for the motor driver

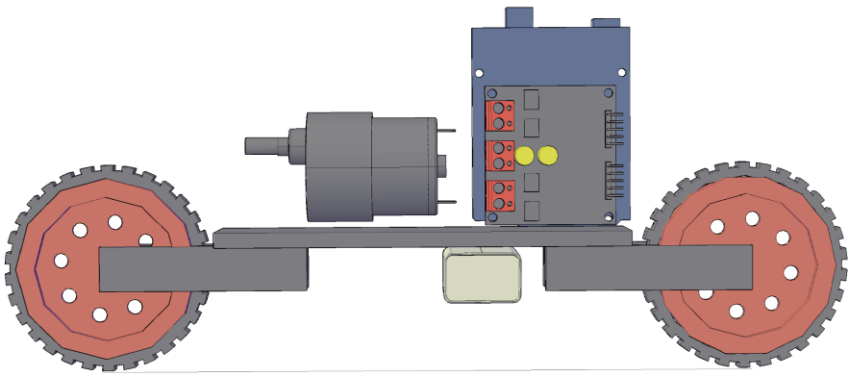


Fig5: CAD model of the assembled components

Problems Faced:

- The flywheel used was provided insufficient amount of wood. Even Though the mass of the flywheel was high, the radius was too low to provide the required moment of inertia.

Iteration 2:

- The CAD model of the cycle was made. The moment of inertia of the system was calculated to be **1757471.014 gm-mm** and the moment of inertia of the flywheel was calculated to be **1129539.827 gm-mm**.
- A different flywheel was used. The new flywheel has reduced mass and larger radius. Hence, we obtained higher torque for lower motor speed. The flywheel has 9.47cm diameter and is made of plastic with a rubber strip in the periphery.

Fig6: The new flywheel

Observation:

This flywheel was able to give the required torque to the cycle. When the motor driver was giving full output, the cycle was overshooting. Hence, by tuning and decreasing the output voltage balancing is possible.

CAD Model

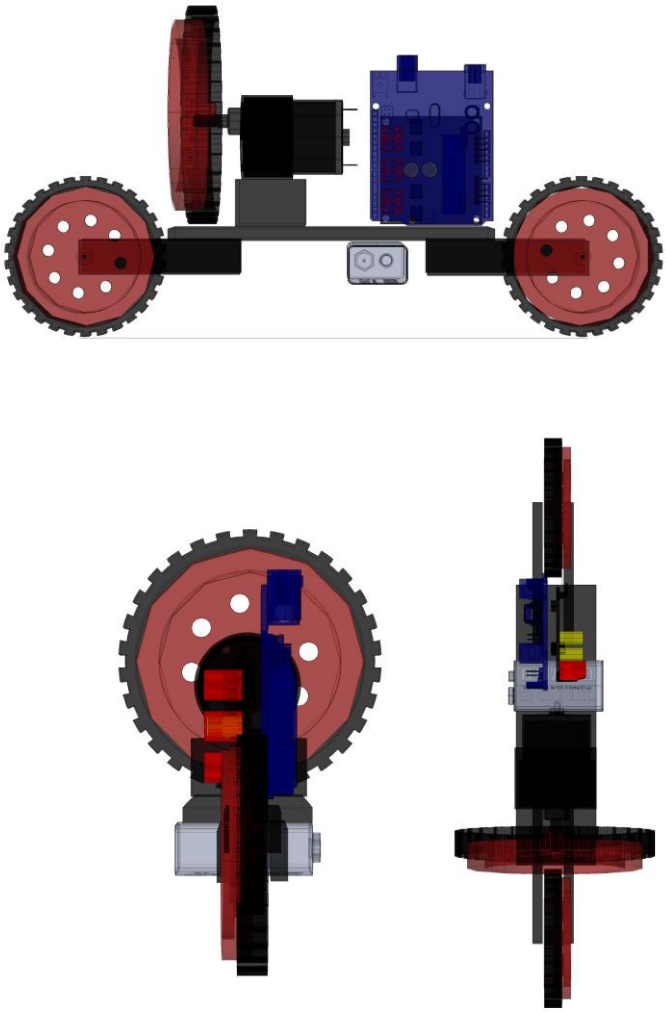


Fig7: CAD Model of the final prototype

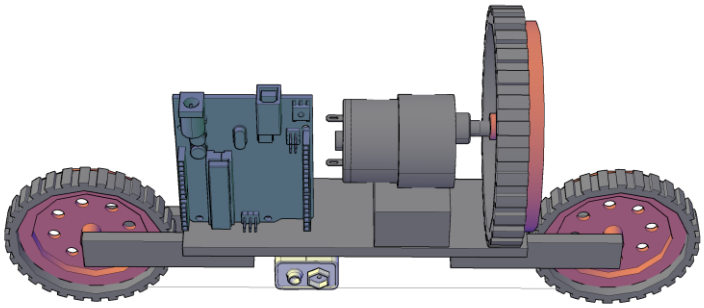


Fig8: 3D conceptual model of the cycle

Controller Law Design:

The mode of control is **PID controller**. (Proportional-integral-derivative)

- Firstly the three parameters were i.e **Kp, Ki and Kd** were defined and were set to zero.
- Then the Kp was slowly increased till we get a relatively stable (Some oscillations were still present i.e oscillatory system) system.
- Ideal method for finding Kp, Ki and Kd:
    - When the Kp was low it takes a greater angle to accelerate the rotation of motor but due to the weight of the cycle it would fall.
    - When Kp is increased a lot, the above difficulty would vanish, the motor would go to its highest power over a small angle, so now it could balance over a relatively smaller range of tilted angle.
    - There was a trade-off between these challenges, so finding Kp was a difficult task.
- After some satisfaction with the Kp tuning for Ki should be done. Ki should be increased till the offset is removed. Again we faced two challenges:
    - When Ki was small much effect was not realised as the steady state error for the system was small.
    - When we increased Ki, though steady state error reduced to some extent but the overshooting increased which caused the bicycle to fall on the other side. Hence a tradeoff had to be attained.
- After this comes the tuning of Kd. The main role of the Kd was to keep the bicycle in control whenever the motor gave the jerk.
    - Again there were trade-offs If we keep the Kd high, the bicycle wouldn't get enough torque.

- If the Kd was low, The jerks given by the motor while changing the direction couldn't be avoided and the bicycle would just fall on the other side due to overshooting.

- In this way Kp, Ki and Kd was supposed to be found.

Bode Plot:

The moment of inertia for the system and flywheel were calculated and the mass of the cycle was calculated. The transfer function obtained was:

$$G(s) = \frac{0.001757s^2 + 0.3134}{0.00113s}$$

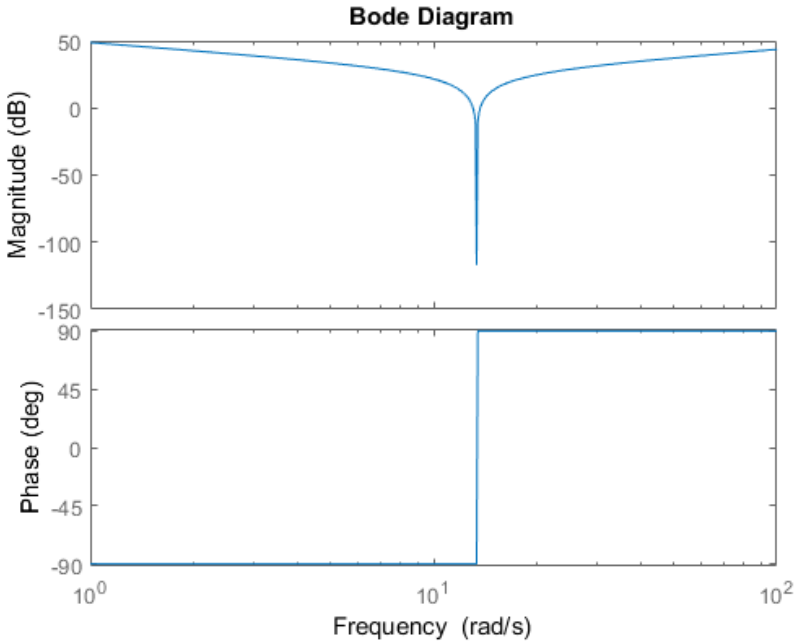


Fig9: Bode Plot

The Bode plot revealed that it has a phase margin of 90°; hence should be stable.

CODE

```
#include <PID_v1.h>           //PID library
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"          //MPU6050 library
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

MPU6050 mpu;

#define OUTPUT_READABLE_YAWPITCHROLL

#define LED_PIN 13
bool blinkState = false;
bool dmpReady = false;
uint8_t mpuIntStatus;
uint8_t devStatus;
uint16_t packetSize;
uint16_t fifoCount;
uint8_t fifoBuffer[64];

const int IN1 = 9;
const int IN2 = 8;
const int ENA = 10;
Quaternion q;
VectorInt16 aa;
VectorInt16 aaReal;
VectorInt16 aaWorld;
VectorFloat gravity;
float ypr[3];                         //variable for yaw, pitch and roll
float PID_error;

uint8_t teapotPacket[14] = { '$', 0x02, 0, 0, 0, 0, 0, 0, 0, 0, 0x00, 0x00, '\r', '\n' };

double Setpoint, Input, Output;   //setpoint is the value of pitch where the cycle balances

double Kp = 27, Ki = 15, Kd = 30;       //defining the values of the PID constants
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
```

```
volatile bool mpuInterrupt = false;  // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
  mpuInterrupt = true;
}

void setup() {

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
  Wire.begin();
  TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
  Fastwire::setup(400, true);
#endif

  Serial.begin(115200);
  while (!Serial);

  mpu.initialize();
  Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050
connection failed"));

  Serial.println(F("\nSend any character to begin DMP programming and demo: "));
  while (Serial.available() && Serial.read());
  while (!Serial.available());
  while (Serial.available() && Serial.read());

  devStatus = mpu.dmpInitialize();

  mpu.setXGyroOffset(220);
  mpu.setYGyroOffset(76);
  mpu.setZGyroOffset(-85);
  mpu.setZAccelOffset(1788);

  if (devStatus == 0) {

          mpu.setDMPEnabled(true);
          attachInterrupt(0, dmpDataReady, RISING);
          mpuIntStatus = mpu.getIntStatus();
          dmpReady = true;
          packetSize = mpu.dmpGetFIFOPacketSize();

  } else {
```

```
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
  }

  pinMode (IN1, OUTPUT);
  pinMode (IN2, OUTPUT);
  pinMode (ENA, OUTPUT);

  Setpoint = -0.76;    //the cycle balances while stationary at this angle

  myPID.SetMode(AUTOMATIC);

}

void loop() {

  if (!dmpReady) return;
  mpuInterrupt = false;
  mpuIntStatus = mpu.getIntStatus();
  fifoCount = mpu.getFIFOCount();

  if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
        mpu.resetFIFO();
        Serial.println(F("FIFO overflow!"));

  } else if (mpuIntStatus & 0x02) {
        while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

        mpu.getFIFOBytes(fifoBuffer, packetSize);

        fifoCount -= packetSize;

#ifdef OUTPUT_READABLE_YAWPITCHROLL          //displaying the values
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
        Serial.print("ypr\t");
        Serial.print(ypr[0] * 180 / M_PI);
        Serial.print("\t");
        Serial.print(ypr[1] * 180 / M_PI);
```

```
            Serial.print("\t");
            Serial.println(ypr[2] * 180 / M_PI);
            Serial.print("\t");
            Serial.println(Output);
#endif
  }

  Input = ypr[1] * 180 / M_PI;

  PID_error = Setpoint - Input ;

  Serial.println(Input);

//giving the  output voltage

  if (PID_error > 0) {
            digitalWrite(IN1, HIGH);
            digitalWrite(IN2, LOW);
            myPID.Compute();
            analogWrite(ENA, Output);

  }
  if (PID_error < 0) {
            digitalWrite(IN1, LOW);
            digitalWrite(IN2, HIGH);
            Input=-Input;
            myPID.Compute();
            analogWrite(ENA, Output);

  }

}
```

PHOTOS