PROJECT REPORT

1 INTRODUCTION

The aim of this project was to demonstrate angular position control of a surface along two cartesian axises. It involved the system to reject disturbances in the form of weight on the controlled surface, and more importantly in the form of changes in orientation of structure onto which the controlled surface is mounted. This effectively led to the tilt stabilization of the controlled surface. It can have applications ranging simple gimbal-based camera devices to aeronautical equipment.

2 METHODOLOGY & MATERIALS

2.1 Design Approach

We built a square shaped structure, with two internal frames (inner and outer), each with free movement about one of the axises. The free movement of these internal frames was achieved by using bearing for fixation at one end, minimizing friction, while a DC Motor at the other to enable position control. See appendix for dimensions.

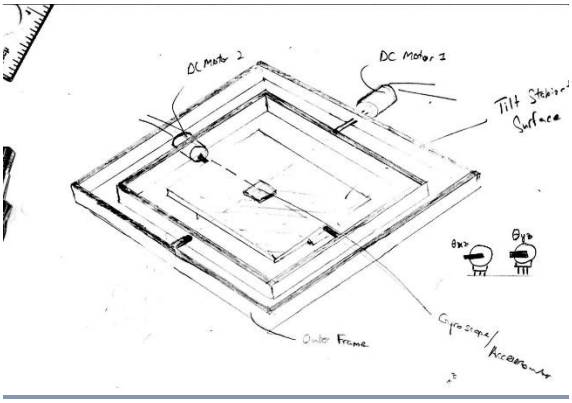
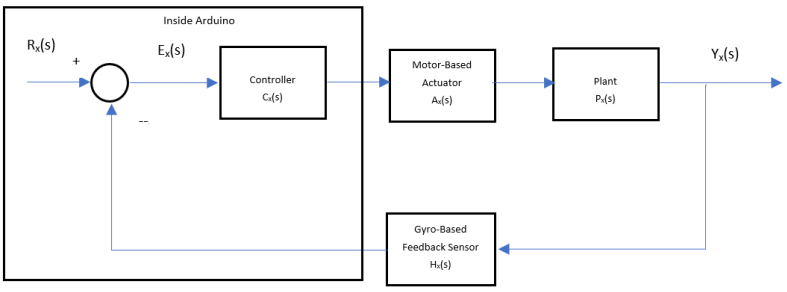
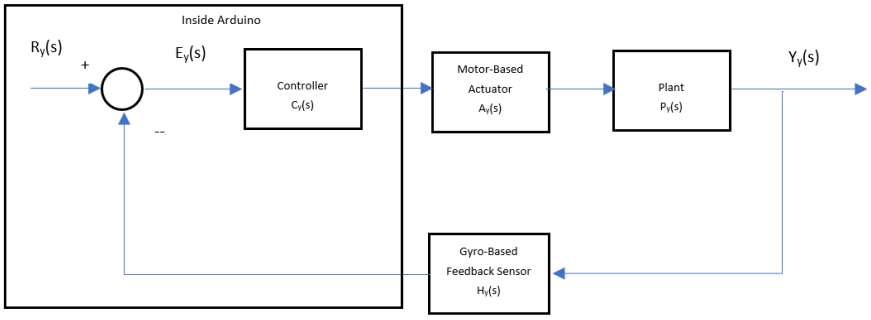


Figure 1 Structural Design

2.2 System Block Diagram

It is worth noting, that our system is composed of two independent, but identifical systems, each for one of the angles with respect to the two cartesian axises, x and y.





2.3 Transfer Functions and Methodologies

2.3.1 DC Motor Transfer Function A(s)

We used a 24 V DC Motors rated at 142 RPM. Its transfer function was obtained using Black Box modelling. Some values we're obtained using first principles, while some approximated, and yet insignificant ones neglected. You can find a detailed derivation of this in the Appendix

$$A(s)_x = A(s)_y \frac{\theta(s)}{V_a(s)} = \frac{0.0182}{(7.725 * 10^{-8})s^3 + (1.1845 * 10^{-4})s^2 + 3.48 * 10^{-4}s}$$

$$A(s)_{approx} = \frac{2.356 * 10^5}{s(s + 2.94)}$$

The model was validated using MATLAB against data obtained from voltage variations when coupled with a dynmo.

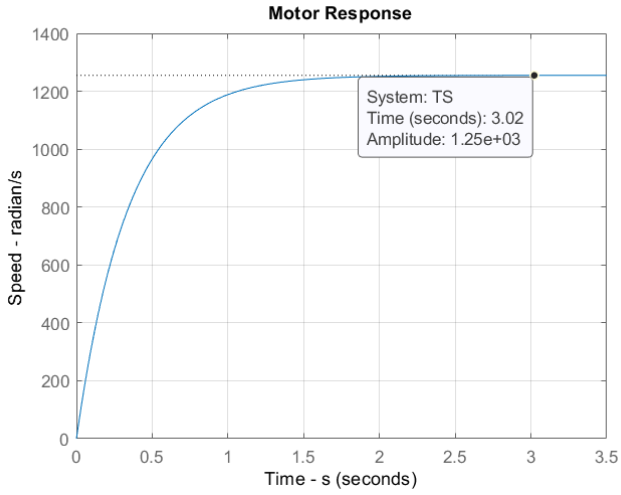


Figure 2 Motor Response Validation on MATLAB

2.3.2 Plant Transfer Function P(s)

This obtained using first principles, by taking into account the various inertia forces acting on the frame due its position in the structure. Effects due to friction and damping were neglected, as

bearing were used, which effectively diminished these. Furthermore, effects of gravity were also neglected, as the structure was gravitationally balanced across a pivot. The derivation for this can found in the Appendix.

$$P(s)_{\mathrm{x-approx}} = \frac{\mathrm{T_t(s)}}{\mathrm{\theta_t(s)}} = \mathrm{J_x s^2}$$

$$P(s)_{\mathrm{y-approx}} = \frac{\mathrm{T_t(s)}}{\mathrm{\theta_t(s)}} = \mathrm{J_y s^2}$$

Where J is the moment of inertia of the structure along an axis.

$$P(s)_{\mathrm{x-approx}} = 41.376\mathrm{s^2}$$

$$P(s)_{\mathrm{y-approx}} = 4.170\mathrm{s^2}$$

2.3.3 Controller Transfer Function

This application only required a proportional controller, which was implemented through an Arduino coupled with a L289 Motor Driver for power amplification. Based on the error-value obtained from the gyro-scope in the angle along X and Y direction, an actuating signal was generated by the controller to drive the motor proportionally until the disturbance was rejected.

$$C(s)_x = C(s)_y = C(z) = K_p = 2.83$$

After fine-tuning, the value was achieved to optimum. The adjusted based on allow the Motor Driver to generate a strong enough actuating signal, even for the smallest of values in error, however, a slight discrepancy still remained. This is due to inaccuracies in the feedback from the Gyroscope, and the inability of the Motor Driver signal to produce a significant voltage to drive the Motor.

2.3.4 Feedback Sensor Transfer Function

Our system is Arduino based, and utilizes a MPU 6050 Gyroscopic sensor for feedback. Since the feedback sensor is digital and coupled with a digital controller, and hence, unity feedback is assigned to it.

$$H(s)_x = H(s)_y = 1$$

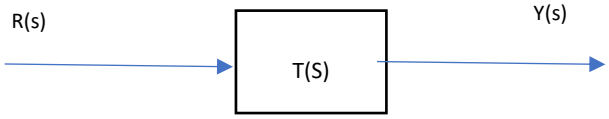
Sensing feedback was feed to an Arduino. The MPU6050 library was used to couple the gyroscopic sensor and get necessary feedback, so that an error could be generate. A complimentary filter was used to receive fairly accurate readings from the sensor, this involved taking into account data from both the Gyroscope and Accelerator, as only one or the other would lead to drift or steady-state error in values respectively.

2.3.5 Open Loop Transfer Function

$$G(s)_x = \frac{2.76s}{s + 2.94}$$
$$G(s)_y = \frac{0.3s}{s + 2.94}$$

2.3.6 Closed Loop Transfer Function

$$T(s)_x = \frac{2.78s}{2.78(s + 2.94)}$$
$$T(s)_y = \frac{0.3s}{0.3(s + 2.94)}$$



3 Results & Remarks

3.1 Model Validation

3.1.1 Experimental Model Simulation

This was done using the closed transfer function above. We can see how the system is rejecting disturbances in the for various angular disturbances. Furthermore, we can see the system achieve steady-state over time as-well. These results confirm that our experimental modelling of the system was accurate to a fair degree.

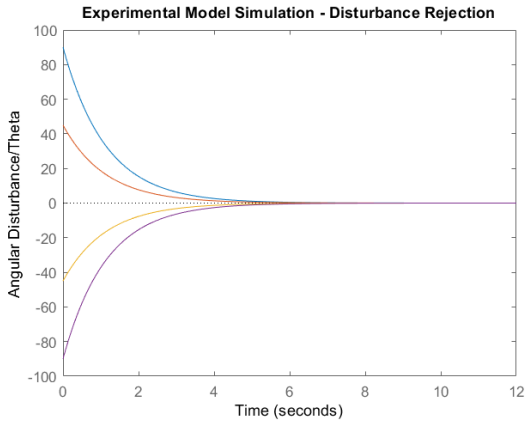


Figure 3 System Experimental Mode along X axis

Response Parameters:

RiseTime: 2.4884 sec SettlingTime: 4.4310 sec SettlingMin: 0.0024 sec SettlingMax: 8.5949 sec Overshoot: Inf
Undershoot: 0 degrees Peak: 90 degrees

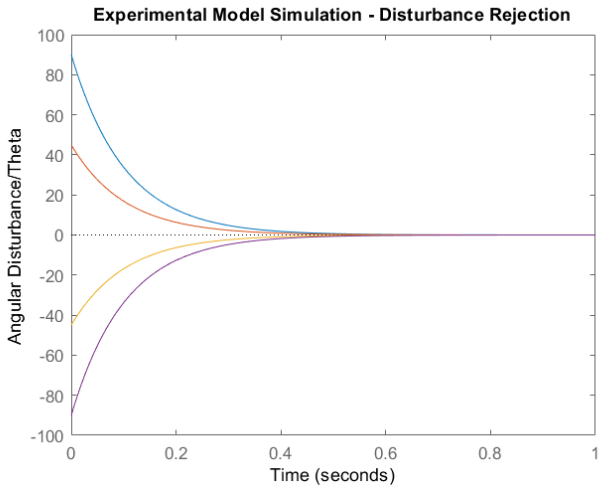
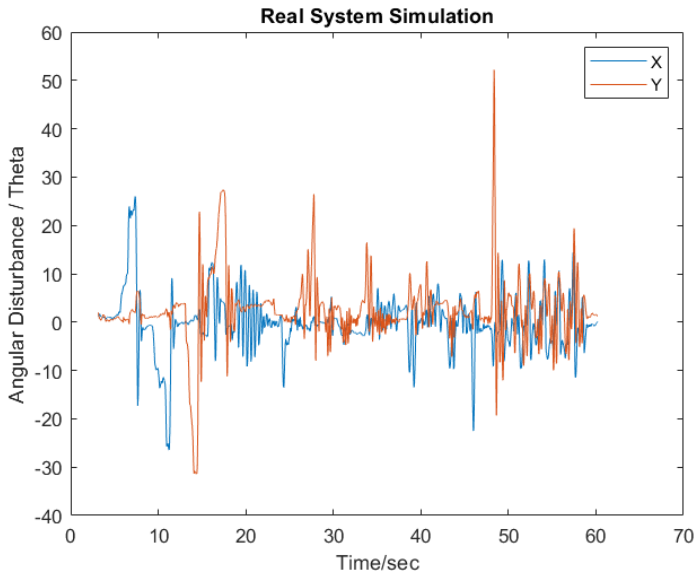


Figure 4 System Experimental Model Simulation along Y Axis

Response Parameters:
RiseTime: 0.2242 sec SettlingTime: 0.3992 sec SettlingMin: 0.0024 sec SettlingMax: 8.5949 sec Overshoot: Inf
Undershoot: 0 degrees Peak: 90 degrees

3.1.2 Real System Simulation

This was achieved by reading data from the Arduino, which was then plotted with MATLAB to generate a graph of the system response.



To get these results, the system was initially slightly disturbed in all four possible positions. These can be observed between 5 – 20 seconds. Then, two disturbances, that resulted in 90-degree rotation of the system were given along the x and y axis, which can be observed in the peak's around

30 and 46 seconds. Finally, a sharp disturbance, that result in a 90-degree rotation about the Y-axis was given at around 50 seconds, after which the system was randomly oscillated.

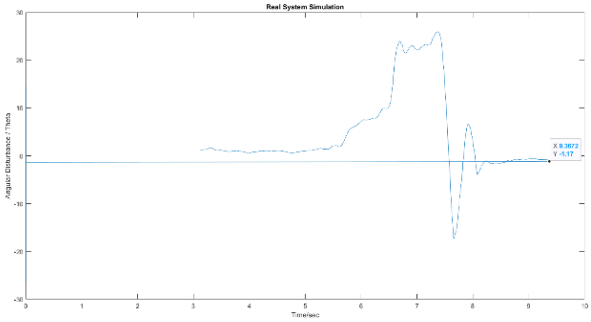


Figure 5 Real System Response along X Axis

Response Parameters:
SettlingTime: about 2-3 seconds

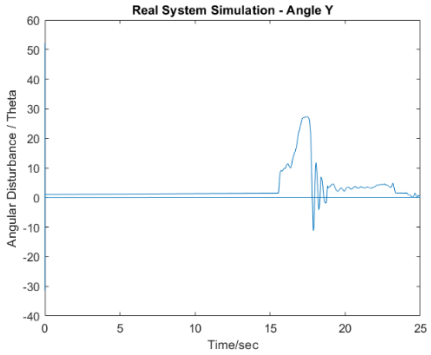


Figure 6 Real System Response along Y Axis

Response Parameters:
SettlingTime: about 4 seconds

Throughout this process, we observe how the system aims to quickly reject disturbances, in the form of peaks and how the steady-state value has mean position of almost 0 degrees along the X-axis, further , with only slight inaccuracies Furthermore, the settling time is also similar. This matches our experimental model However, there is some steadystate error along the Y-axis, and there is some discrepancy in the settling time as well. Hence, this does not exactly match our experimental model, but leads to similar response. However, it does validate our overall model to some extent.

4 Conclusion

Our Project turned out to be fairly successful. We achieved all our key aims and objectives that had been initially proposed. We able to apply our understand of control system, with respect to position and proportional control in this system. Furthermore, we also developed an extended understanding using sensors and their caliberation.

Appendix

1 Structure Dimension Details

M_{in} = *Mass of inner frame = 110.3 g = 0.110 kg*
M_{out} = *Mass of outer frame = 431.2 g = 0.431 kg*
L_{xa} = L_{xb} =*Length of outer frame = 24 cm*
L_{ya} = *Length of inner frame = 16.5 cm*
L_{yb} = *Length of inner frame = 13.5 cm*

2 Derivation of Motor Transfer Function A(s)

General Transfer Function:

$$\frac{\theta(s)}{V_a(s)} = \frac{K_t}{s[J_m\, s\, (sL_a + R_a) + K_b K_t]}$$

Determining Motor Parameters:

By using first principles, we came up with the below values:

- R_a = 23 Ohm's – obtained using Digital Multimeter
- L_a = 15 mH – obtained using Impedance Analyzer
    - Alternatively, also used a theoretical approach to calculate the impedance which involved using the root means square voltage and current values to calculate the impedance, for a sinusoidal signal to the motor. Using, $Z^2 = \sqrt{(R^2 + X^2)}$

$$L = \frac{X}{2\pi f}$$

Using this, L_a obtained to be 1.41 H, however, this result has inaccuracies.

The Motor Back Emf Constant: $K_b = \frac{V_a - I_a R_a}{\omega}$

where V_a = 24 V, I_a=50 mA using DMM, R_a= 23 Ohms, w = 1253.5 rad/s

$$K_b = 0.0182\ \text{Vs/rad}$$

By experience, based on a vast majority of DC motor K_t, the motor torque constant is equal to K_t.

$$K_t = 0.0182\ \text{Nm/A}$$

For the damping coefficient K_m, we use the relation:

$$D_m = \frac{K_t I_m}{\omega}$$

then,

$$D_m = 7.3 * 10^{-7}\text{Nm s/rad}$$

For J_m we used a motor with a low-rpm motor, who's J can be assumed to be zero. This motor was coupled with the modelling motor. A square wave was applied to the modelling motor,

which in turn forced the low-rpm motor to rotate and generate an emf. The emf was used to display a response of the model motors transient characteristics on a Digital Oscilloscope, using which τ was estimated at 0.63 of the final value. Then using the relation:

$$J_m = \frac{\tau\,(D_m R_a + K_b K_t) - D_m L_a}{R_a}$$

$$J_m = 5.15 * 10^{-6}\ \text{N m}$$

Summary of Motor Transfer Function Parameters

Parameter	**Value**
R_a	23 Ω
L_a	15 mH
J_m	5.15×10^{-6} Nm
K_b	0.0182 Vs/rad
K_m	0.0182 Nm/ A
D_m	7.3×10^{-7} Ns/m
N(gear ratio)	1/84

3 Plant Model Derivation

Our system can be assumed to be a model of a simple rotational mass across a motor shaft.

$$\boldsymbol{T_x(s) = (J_x s^2 + B_x s + K_x)\theta_x(s)}$$
$$\boldsymbol{T_y(s) = (J_y s^2 + B_y s + K_y)\theta_y(s)}$$

Where,

- $\boldsymbol{M_{in}}$ = Mass of inner frame = 110.3 g = 0.110 kg
- $\boldsymbol{M_{out}}$ = Mass of outer frame = 431.2 g = 0.431 kg
- $\boldsymbol{L_{xa} = L_{xb}}$ =Length of outer frame = 24 cm
- $\boldsymbol{L_{ya}}$ = Length of inner frame = 16.5 cm
- $\boldsymbol{L_{yb}}$ = Length of inner frame = 13.5 cm

Using the dimensions and masses of the inner and outer frames, the J can be calculated, and then neglecting the damping and friction, we can easily estimate the transfer function of the plant along the x and y axis, for rotational motion.

$$J_x = \frac{M_{out}(L_{xa}^2 + L_{xb}^2)}{12} = \frac{0.431(24^2 + 24^2)}{12} = 41.376 \ kg\ cm^2$$

$$J_y = \frac{M_{in}(L_{ya}^2 + L_{yb}^2)}{12} = \frac{0.110(16.5^2 + 13.5^2)}{12} = 4.116\ kg\ cm^2$$

These two can be added via super-position, to achieve the desired transfer function.

$$\frac{T_t(s)}{\theta_t(s)} = (41.376_y + 4.116_x)s^2$$

$$\frac{T_t(s)}{\theta_t(s)} = 45.492s^2$$

Using the above equation and the gear ratio

$$N^2 = \frac{N_2}{N_1}$$

This can be the be coupled with the transfer function of the motor

4 Sensing and Controller Code

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"

// ================================================================
// ===                INITILIZATIONS                ===
// ================================================================


//---- Defining Motor Control Pins on Arduino
int pin_fwd = 7;              //-- IN1 - Motor 1
int pin_bwd = 6;              //-- IN2 - Motor 1
int pin_fwd2 = 5;             //-- IN3 - Motor 2
int pin_bwd2 =  4;            //-- IN4 - Motor 2

int pin_pwm1 = 9;             //-- ENA - Motor 1
int pin_pwm2 = 10;            //-- ENB - Motor 2

int motorPWM1 = 0;            //-- PWM Value Motor 1
int motorPWM2 = 0;            //-- PWM Value, Motor 2

int Kp = 2.83;                //-- Proportional Controller Constant
int Error = 0;

//---- I2C Library for using Wire
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

MPU6050 mpu;                  //-- Creating Sensor Object

#define OUTPUT_READABLE_YAWPITCHROLL
#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards

float correct;
int j = 0;
bool blinkState = false;

//---- MPU control status vars
bool dmpReady = false;        //-- set true if DMP init was successful
uint8_t mpuIntStatus;         //-- holds actual interrupt status byte from MPU
uint8_t devStatus;            //-- return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;          //-- expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;           //-- count of all bytes currently in FIFO
```

```
uint8_t fifoBuffer[64];       //-- FIFO storage buffer

// orientation-motion vars
Quaternion q;               //-- [w, x, y, z]       quaternion container
VectorInt16 aa;             //-- [x, y, z]          accel sensor measurements
VectorInt16 aaReal;         //-- [x, y, z]          gravity-free accel sensor measurements
VectorInt16 aaWorld;        //-- [x, y, z]          world-frame accel sensor measurements
VectorFloat gravity;        //-- [x, y, z]          gravity vector
float euler[3];             //-- [psi, theta, phi]   Euler angle container
float ypr[3];               //-- [yaw, pitch, roll]  yaw/pitch/roll container and gravity vector


// ================================================================
// ===            INTERRUPT DETECTION ROUTINE                   ===
// ================================================================

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

// ================================================================
// ===                  INITIAL SETUP                           ===
// ================================================================

void setup() {
  pinMode(pin_fwd, OUTPUT);
  pinMode(pin_bwd, OUTPUT);
  pinMode(pin_fwd2, OUTPUT);
  pinMode(pin_bwd2, OUTPUT);

  //---- Prepare IC2 for use on Arduino
  #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();
    Wire.setClock(400000);    //-- 400kHz I2C Clock
  #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
  #endif

  //---- Initiliing Serial Communication
  Serial.begin(115200);
  while (!Serial);

  //---- Initializing Sensor Communication
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);
    devStatus = mpu.dmpInitialize();

    //-- Gyro Offset Values Settings
    mpu.setXGyroOffset(17);
    mpu.setYGyroOffset(-69);
    mpu.setZGyroOffset(27);
    mpu.setZAccelOffset(1551);

    //-- Confirmation Check
    if (devStatus == 0) {
      mpu.setDMPEnabled(true);
      //-- Attaching Interrupt to obtain information
      attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
      mpuIntStatus = mpu.getIntStatus();
      packetSize = mpu.dmpGetFIFOPacketSize();
      Serial.println("Initialization");
     }
     else { Serial.println("Initialization failed");  }
}


// ================================================================
// ===                  MAIN PROGRAM LOOP                       ===
// ================================================================

void loop() {
  //---- If Initilization Failed. Return & End
  if (!dmpReady) return;



  // wait for MPU interrupt or extra packet(s) available
  while (!mpuInterrupt && fifoCount < packetSize) {
    if (mpuInterrupt && fifoCount < packetSize) {
      // try to get out of the infinite loop
      fifoCount = mpu.getFIFOCount();
    }
  }
  // reset interrupt flag and get INT_STATUS byte
  mpuInterrupt = false;
```

```
  mpuIntStatus = mpu.getIntStatus();

  // get current FIFO count
  fifoCount = mpu.getFIFOCount();

  // check for overflow (this should never happen unless our code is too inefficient)
  if ((mpuIntStatus & _BV(MPU6050_INTERRUPT_FIFO_OFLOW_BIT)) || fifoCount >= 1024) {
    // reset so we can continue cleanly
    mpu.resetFIFO();
    fifoCount = mpu.getFIFOCount();
    Serial.println(F("FIFO overflow!"));
    // otherwise, check for DMP data ready interrupt (this should happen frequently)
  } else if (mpuIntStatus & _BV(MPU6050_INTERRUPT_DMP_INT_BIT)) {
    // wait for correct available data length, should be a VERY short wait
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
    // read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);

    // track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an interrupt)
    fifoCount -= packetSize;



    //---- Data from Gyroscope: Yaw, Pitch and Roll values
#ifdef OUTPUT_READABLE_YAWPITCHROLL
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

    //---- Pitch, Roll values - Radians to degrees
    ypr[1] = ypr[1] * 180 / M_PI;
    ypr[2] = ypr[2] * 180 / M_PI;

    //---- Some additional adjustments
    ypr[2] = -1*ypr[2] -1 ;
    ypr[1] = ypr[1]-3;
    }
    else {

      //---- Implementing Proportional Controller
     if (ypr[2] >= 0){                  //---- If +ve 90 w.r.t to x-axis
        Error = abs(ypr[2])
        digitalWrite(pin_fwd2, LOW);
        digitalWrite(pin_bwd2, HIGH);
        motorPWM2 = 35 + (Error*Kp);
        analogWrite(pin_pwm2, motorPWM2);
        }
     else if (ypr[2] < 0){              //---- If  -ve 90 w.r.t to x-axis
        Error = abs(ypr[2])
        digitalWrite(pin_fwd2, HIGH);
        digitalWrite(pin_bwd2, LOW);
        motorPWM2 = 60 + (Error*Kp);
        analogWrite(pin_pwm2, motorPWM2);
        }
     else{ ypr[2] = 0;   }


     if (ypr[1] >= 0){                  //---- If +ve 90 w.r.t to y-axis
        Error = abs(ypr[1])
        digitalWrite(pin_fwd, HIGH);
        digitalWrite(pin_bwd, LOW);
        motorPWM1 = 35 + (Error*Kp);
        analogWrite(pin_pwm1, motorPWM1);
        }
     else if (ypr[1] < 0){              //---- If -ve 90 w.r.t to y-axis
        digitalWrite(pin_fwd, LOW);
        digitalWrite(pin_bwd, HIGH);
        motorPWM1 = 55 + (Error*Kp);
        analogWrite(pin_pwm1, motorPWM1);
        }
     else { ypr[1] = 0;  }

     //---- Printing Actuating Signal and Angle Values from Gyro.
     Serial.print("MotorX: ");     Serial.print(motorPWM2);     Serial.print (" | ");
     Serial.print("MotorY: ");     Serial.print(motorPWM1);     Serial.print(" | ");
     Serial.print("AngleX: ");     Serial.print(ypr[2]);        Serial.print(" | ");
     Serial.print("AngleY: ");     Serial.print(ypr[1]);        Serial.println(" ");
    }

#endif
  }
}
```

5 Real Time Model Validation Code

```
plot(time, X);
hold on;
plot(time, Y);
title('Real System Simulation')
ylabel('Angular Disturbance / Theta');
xlabel('Time/sec');
legend('X', 'Y');
```

13 Experimental Model Validation Code

```
input = 90;
TS = tf([input*3.33, 0],[3.33,2.94]);

step(TS);
stepinfo(TS)
hold on;
input = +45;
TS = tf([input*3.33, 0],[3.33,2.94]);
step(TS);
hold on;
input = -45;
TS = tf([input*3.33, 0],[3.33,2.94]);
step(TS);
hold on;
input = -90;
TS = tf([input*3.33, 0],[3.33,2.94]);
step(TS);
title('Experimental Model Simulation - Disturbance Rejection');
ylabel('Angular Disturbance/Theta');
```

33 Real System Picture