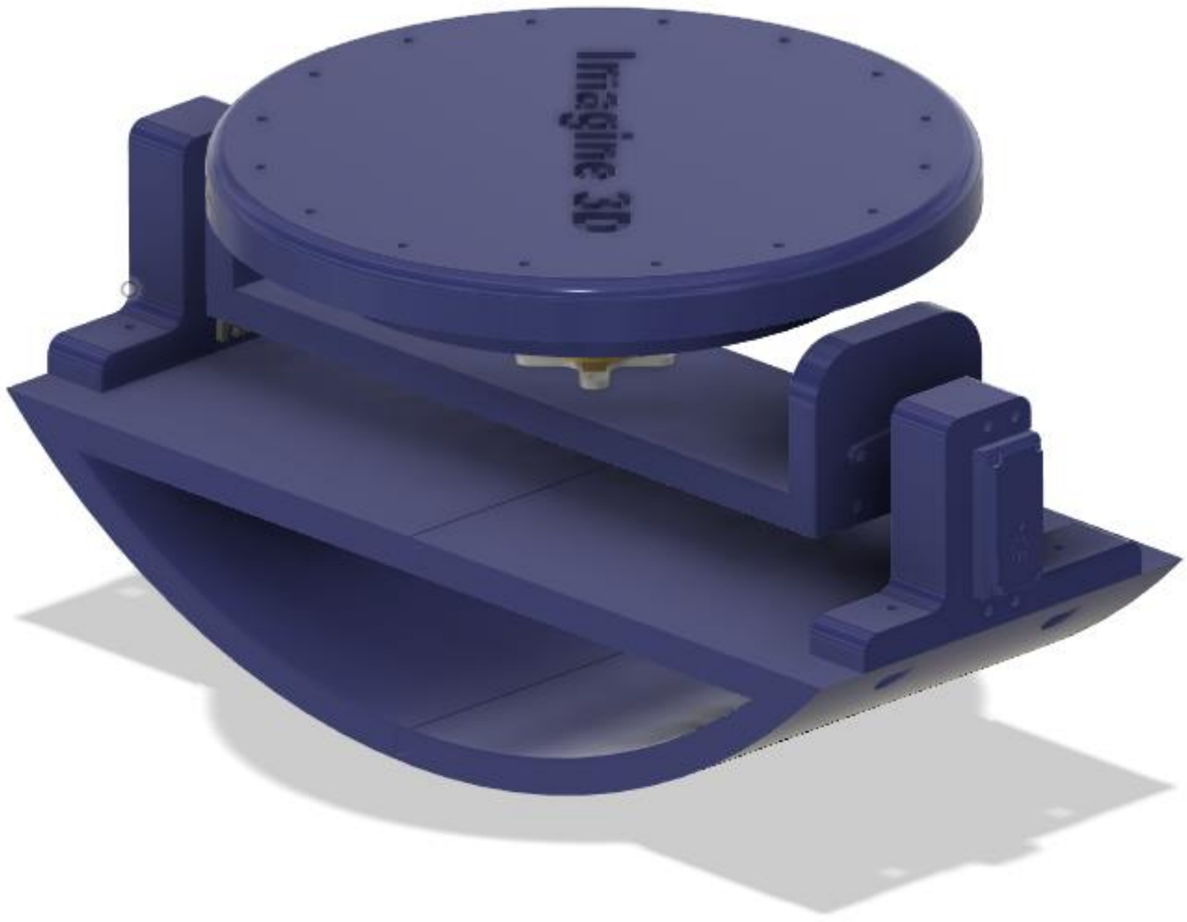
Gyroscopic Boat Stabilization System



Group Number: T-31

Group Members: Bassel AbdelHaleem 49-1931

Omar Elshemerly 49-2116

Mohamed Abdelrahman 49-857

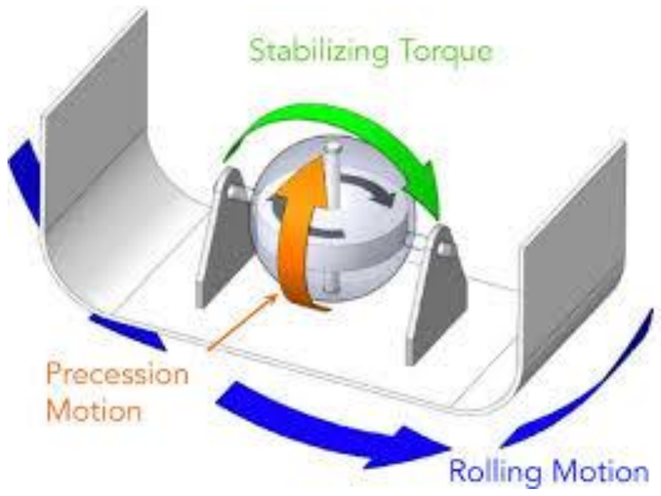
Date: 01/06/2022

Table of Contents:

Overview

During sea travel, ships and boats are subjected to a lot of forces due to wind and waves which in turn cause a boat to become unstable. As a result, marine engineers always looked for solutions to avoid all the unwanted stability-issues due to those external forces. Then, they realized that gyroscopes could be really helpful in stabilizing the ships and preventing this “rolling” phenomenon.

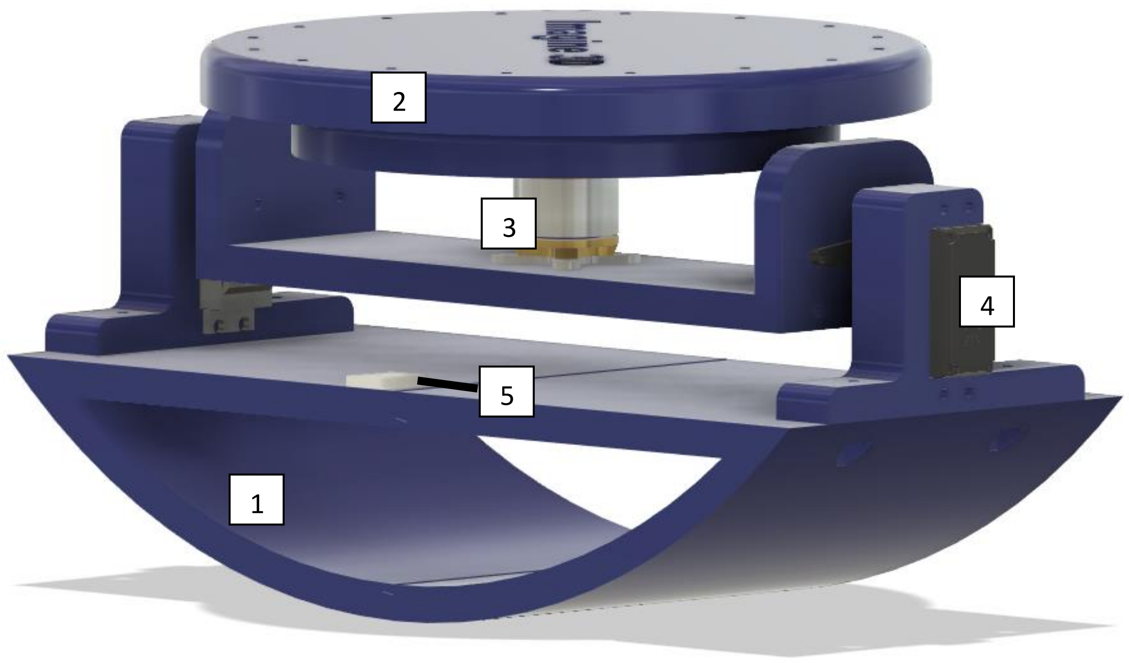
A gyroscope is a device that rotates at really high speeds which generates a high angular momentum. Therefore, the external torque which is generated by the waves of the ocean would be negligible compared to the angular momentum of the gyroscope. Which would cause a more stable trip.



In our project we recreate the stabilization effect of a gyroscope and use the gyroscopic precession phenomenon to be able to stabilize a small model of a 3D printed boat. A motor spins up a gyroscope which is mounted on a platform controlled by a servo which allows us to control the angle of the gyroscope and thus allowing us to counter and cancel the external torque that the boat would be subject to. That’s where the closed loop control of our project resides; where we would take the readings from a gyroscopic sensor to tell us the tilt angle and the error we need to compensate for, which would be calculated using a PID controller programmed on an ATMEGA328P. As an external input we would add a button to be able to switch on and off our PID controller. As our output we would use an LCD to show the output of the tilt angle and the values of the error that we need to compensate.

System Details

3D Model



1-Boat

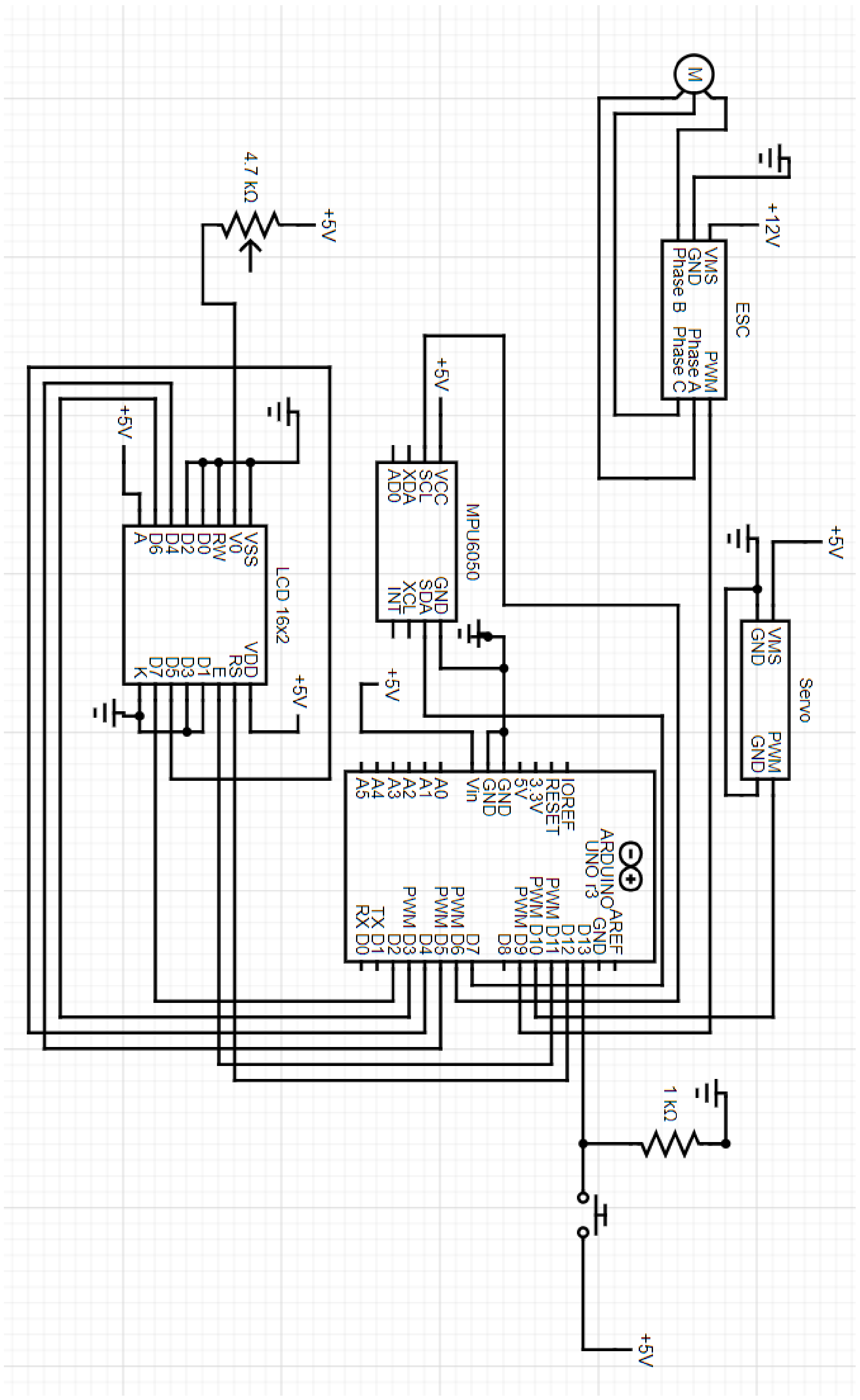
2-Flywheel

3-Brushless Motor

4-Servo Motor

5-MPU6050

Circuit diagram



Design Evaluation

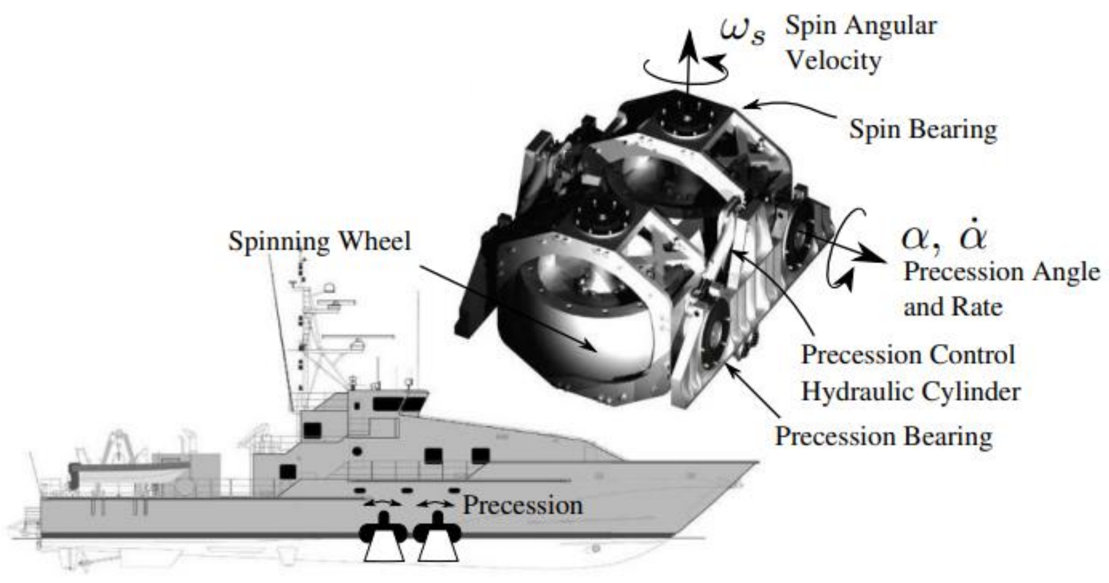

ω_s Spin Angular Velocity
Spin Bearing
Spinning Wheel
$\alpha,\ \dot{\alpha}$ Precession Angle and Rate
Precession Control Hydraulic Cylinder
Precession Bearing
Precession

Fig.1 Example of Gyrostabilizer

Even though our project is prototype of a gyrostabilizer on land, we modelled our system for a ship in water and partially used this model. The model for the motion of the ship together with the wheel gyrostabilizer can be expressed as follows (refer to block diagram in appendix1):

$I44\ddot{\phi} + B44\dot{\phi} + C44\phi = \tau w - nKg\dot{\alpha}$,

$Ig\ddot{\alpha} + Bg\dot{\alpha} + Cg\alpha = Kg\dot{\phi} + \tau p$,

where the variables and parameters of the model are:

- ϕ—ship roll angle; τw—wave-induced roll moment;
- α—precession angle; τp—precession control torque.
- I44—moment of inertia of the ship in roll including hydrodynamic added inertia;
- B44—total equivalent roll damping coefficient including potential and viscous effects;
- C44–roll restoring coefficient.
- Ig—inertia of a single spinning wheel along the precession axis;
- Bg—damping coefficient associated with friction in the precession bearings;
- Cg–restoring coefficient associated with the mass distribution of the spinning wheel (pendulum effect);

- Kg—spinning angular momentum (Kg = ωspin*Ispin); item n—number of spinning wheels.

The closed loop transfer function from wave-induced moment to roll angle is:

$$\mathrm{Hcl(s)} = \frac{\varphi(\mathrm{s})}{\tau\mathrm{w(s)}}$$
$$= \frac{(\mathrm{Igs^2 + B'g\,s + C'g})}{(\mathrm{Igs^2 + B'g\,s + C'g})\,(\mathrm{I44s^2 + B44s + C44})\,(\mathrm{Igs^2 + B'g\,s + C'g}) + \mathrm{nK^2 g\,s\,2}}$$

Check appendix3 for the uncontrolled system modelling.

The transfer function from roll angle to precession angle (Control System) is:

$$\mathrm{G(s)} = \frac{\alpha(\mathrm{s})}{\varphi(\mathrm{s})} = \frac{\mathrm{Kgs}}{\mathrm{Igs^2 + B'g\,s + C'g}}$$

Thus, the transfer function from the wave excitation roll moment to the precession angle becomes:

$$F(s) = \frac{\alpha(\mathrm{s})}{\tau\mathrm{w(s)}} = \mathrm{G(s)Hcl(s)}$$
$$= \frac{\mathrm{Kgs}}{(\mathrm{I44s\verb|^|2 + B44s + C44})\,(\mathrm{Igs\verb|^|2 + B'gs + C'g}) + \mathrm{nK2gs\verb|^|2}}$$

In our project, the only coefficients that are taken into consideration are moments of inertia (I44 & Ig), the wheels angular momentum (Kg), and roll damping coefficient (B44); and all others are taken equal to 1 as they do not affect our model. Accordingly, they are calculated as follows:

B44 is taken as 0.08 (check appendix3)

I44 = 0.5*m_b* r_b^2 = 3.9375x10^{-3},

where m_b is the mass of boat (=350gm), r_b is the radius of boat (=15cm);

I_g=0.5*m_w*r_w^2 = 1.5 x10^{-3},

where m_w is the mass of flywheel (=300gm), r_w is the radius of flywheel (=10cm);

Kg = w_s*Ig = 0.66,

where w_s is the angular velocity of flywheel and calculated as:

$Ws = \frac{2\pi\mathrm{n}}{60} = \frac{2\pi 4000}{60}$, where n is rpm of flywheel.

Substituting all coefficients in the transfer function results in:

$$F(s) = \frac{0.66s}{1.126x10^{-5}s^4 + 0.003938\, s^3 + 0.4424\, s^2 + s + 1}$$

Check appendix4 for the controlled system modelling.

Components

Part Name	*Description*	*Model No*	*Price (LE)*
Accelerometer	Measures angular velocity (tilt)	MPU6050	50
Servo Motor	Rotates the gyroscope for stabilization	MG995	230
Brushless Motor	Spins the gyroscope	A2212	300
ESC	Drives brushless motor	BLHeli 20A	250
Microcontroller	Controls	ATmega328p	100

Appendix

(1) Block Diagram

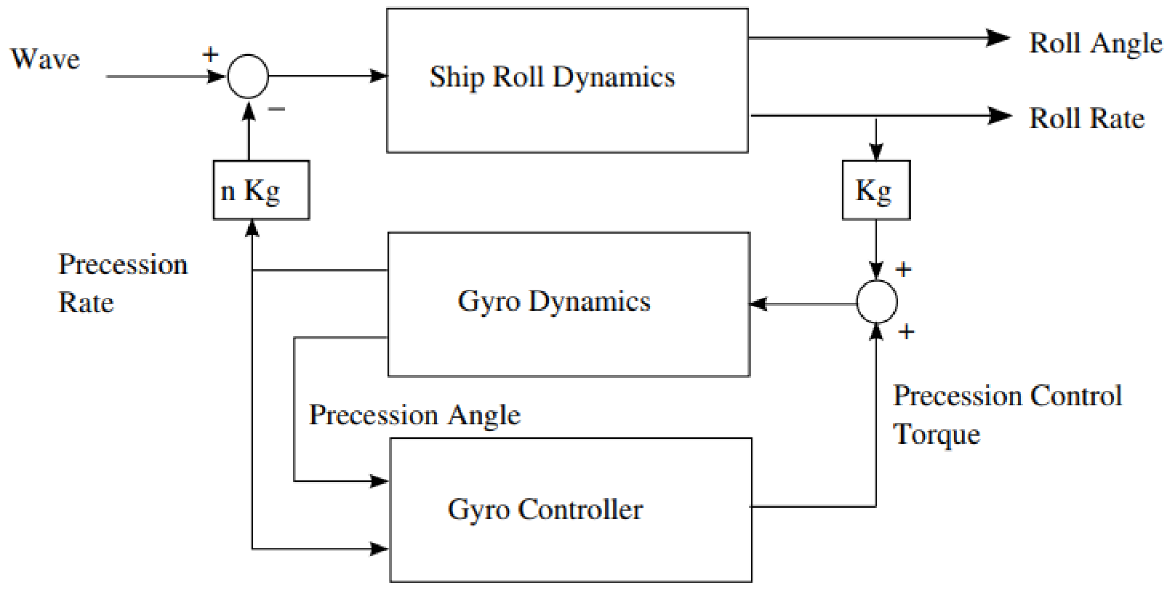


Fig.2 Gyrostabilizer Block diagram

(2) Citation to referance of B44 value

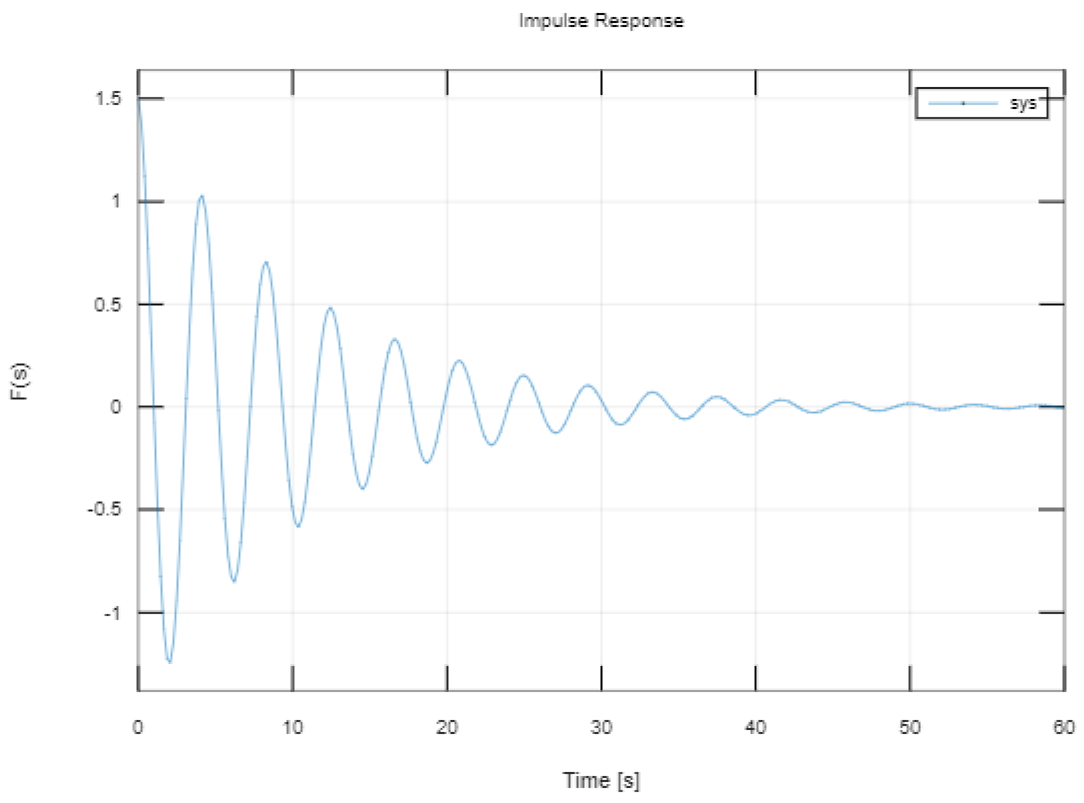
B44 is taken as 0.08 as this is the worst case scenario accoring to, *A hybrid empirical–analytical model for predicting the roll motion of prismatic planing hulls.*

Tavakoli, Sasan & Ghadimi, Parviz & Sahoo, Prasanta & Dashtimanesh, Abbas. (2017). A hybrid empirical–analytical model for predicting the roll motion of prismatic planing hulls. Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment. 232. 147509021668921. 10.1177/1475090216689212.

(3) System without control:

```
Kg=0.66;
I44=0.0039375;
Ig=0.002859375;
B44=0.08;
```

```
Bg=1;
C44=1;
Cg=1;
n=1;
sys = tf([Kg 0],[(I44+n*Kg^2) (B44) (C44)])
impulse(sys)
ylabel('F(s)');
```

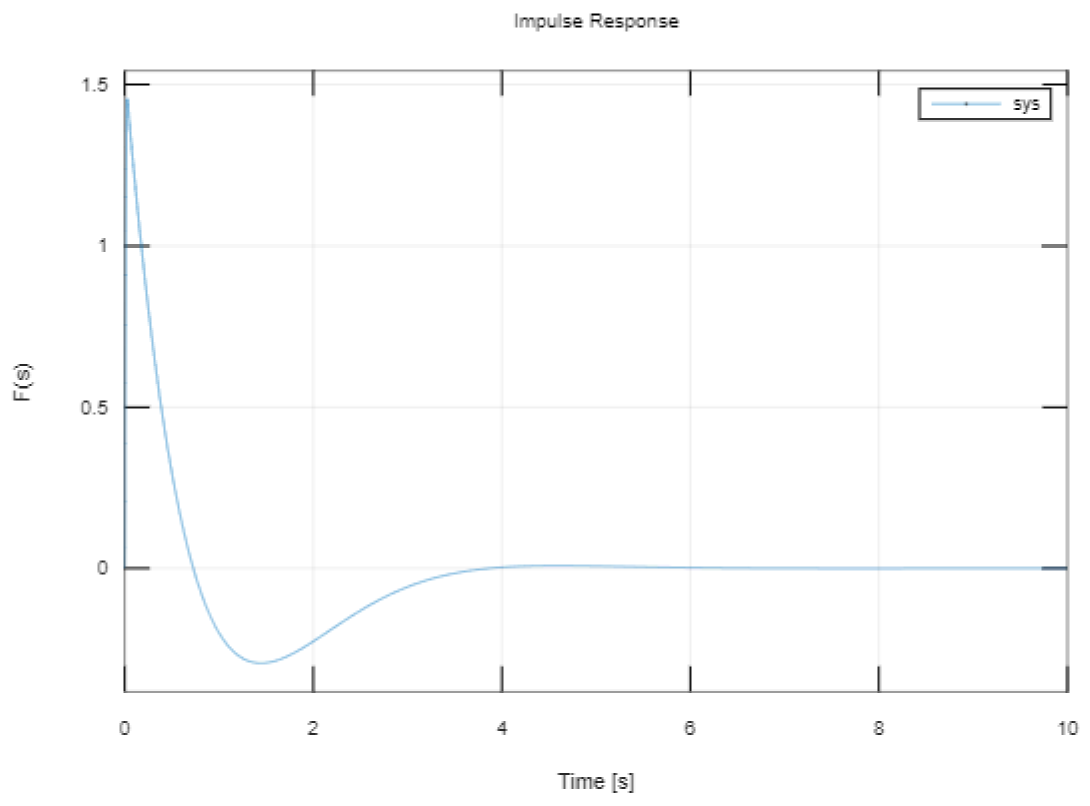


As seen from the Impulse Response plot, the uncontrolled system is unstable and takes a long time to halt.

(4) System with control:

```
Kg=0.66;
I44=0.0039375;
Ig=0.002859375;
B44=0;
Bg=1;
C44=1;
Cg=1;
n=1;
```

```
sys = tf([Kg 0],[(I44*Ig) (I44*Bg+B44*Ig) (I44*Cg+B44*Bg+C44*Ig+n*(Kg^2))
(B44*Cg+C44*Bg) (C44*Cg)])
impulse(sys)
ylabel('F(s)');
```



As seen from the Impulse Response plot, the controlled system stabilizes after a short time.

Code

Arduino code (without multi-tasking):

```
#include <Wire.h>
#include <MPU6050.h>
#include <Servo.h>
#include <LiquidCrystal.h>
MPU6050 mpu;
Servo myservo;
Servo ESC;
int error;
int potValue;
int u;
float Kp=1;// proportional constant
float Kd=0; //differential constant
float pid_p; //control values
float pid_d;
float time;
float timePrev;
float elapsedTime;
int Prevroll;
int control = 13;
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
void setup() {
  pinMode(13, INPUT);
  lcd.begin(16, 2);
  Serial.begin(115200);
   time=millis();
  Serial.println("Initialize MPU6050");
//initializing mpu
  while(!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G))
  {
    Serial.println("Could not find a valid MPU6050 sensor, check wiring!");
    delay(500);
  }
   myservo.attach(10);  // attaches the servo on pin 9 to the servo object
    ESC.attach(9,1000,2000);
}
void loop() {
  // Read normalized values
  Vector normAccel = mpu.readNormalizeAccel();
     // Calculate Pitch & Roll
   timePrev=time;
  time=millis();
  elapsedTime=(time-timePrev)/1000;
  int pitch = -(atan2(normAccel.XAxis, sqrt(normAccel.YAxis*normAccel.YAxis + 
normAccel.ZAxis*normAccel.ZAxis))*180.0)/M_PI;
  int roll = (atan2(normAccel.YAxis, normAccel.ZAxis)*180.0)/M_PI;
  //control esc with pot.
```

```
}

potValue = analogRead(A0);
potValue = map(potValue, 0, 1023, 0, 180);
ESC.write(potValue);
roll=roll-1;//offseting angle
pid_p=Kp*roll; //calculating proportional control
pid_d=Kd*((roll-Prevroll)/elapsedTime); ////calculating differentialcontrol
 // displaying data on screen
 lcd.setCursor(0,0);
 lcd.print("ANGLE:");
 lcd.setCursor(7,0);
 lcd.print(roll);
// button to switch on/off control
  if (digitalRead(control) == HIGH) {
      lcd.setCursor(0,1);
      lcd.print("CONTROL:ON");
      u=pid_p+pid_d;
  }


  if (digitalRead(control )== LOW) {
    lcd.setCursor(0,1);
    lcd.print("CONTROL:OFF");
    u=0;
  }
  // setting servo angle
  myservo.write(u+90);
  Prevroll=roll;
 delay(30);
 lcd.clear();
}
```

Arduino code (with multi-tasking):

```
#include <Arduino_FreeRTOS.h>
#include <Servo.h>
#include <LiquidCrystal.h>
#include <Wire.h>
#include <MPU6050.h>
#include <semphr.h>
#include <queue.h>
MPU6050 mpu;
int roll;
int ctrl = 13;
QueueHandle_t structQueue;
SemaphoreHandle_t xSerialSemaphore;
// define two Tasks for DigitalRead & AnalogRead
void getRoll( void *pvParameters );
void LCD( void *pvParameters );
void ESC( void *pvParameters );
void control( void *pvParameters );
```

```
// the setup function runs once when you press reset or power the board
void setup() {
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB, on LEONARDO,
MICRO, YUN, and other 32u4 based boards.
  }
structQueue = xQueueCreate(10, // Queue length
                              sizeof(roll)); // Queue item size
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  if (structQueue != NULL) {
  xTaskCreate(
    getRoll
    ,  "ROLL"  // A name just for humans
    ,  128  // This stack size can be checked & adjusted by reading the Stack
Highwater
    ,  NULL //Parameters for the task
    ,  2 // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0 being
the lowest.
    ,  NULL ); //Task Handle
    xTaskCreate(
    control
    ,  "control"  // A name just for humans
    ,  128  // This stack size can be checked & adjusted by reading the Stack
Highwater
    ,  NULL //Parameters for the task
    ,  3 // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0 being
the lowest.
    ,  NULL ); //Task Handle
  }
  xTaskCreate(
    LCD
    ,  "LCD"  // A name just for humans
    ,  128  // This stack size can be checked & adjusted by reading the Stack
Highwater
    ,  NULL //Parameters for the task
    ,  0  // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0
being the lowest.
    ,  NULL ); //Task Handle
     xTaskCreate(
    ESC
    ,  "esc"  // A name just for humans
    ,  128  // This stack size can be checked & adjusted by reading the Stack
Highwater
    ,  NULL //Parameters for the task
    ,  1  // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0
being the lowest.
    ,  NULL ); //Task Handle
  // Now the Task scheduler, which takes over control of scheduling individual
Tasks, is automatically started.
}
void loop()
```

```
{
  // Empty. Things are done in Tasks.
}


/*--------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*--------------------------------------------------*/
void getRoll( void *pvParameters __attribute__((unused)) )  // This is a Task.
{
  while(!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G))
  {
    Serial.println("Could not find a valid MPU6050 sensor, check wiring!");
    delay(500);
  }
  for (;;) // A Task shall never return or exit.
  {
    Vector normAccel = mpu.readNormalizeAccel();
     int roll = (atan2(normAccel.YAxis, normAccel.ZAxis)*180.0)/M_PI;
     roll=roll-1;
     Serial.print(" roll = ");
     Serial.print(roll);
     Serial.println();
    xQueueSend(structQueue, &roll, portMAX_DELAY);
    //vTaskDelay(1);  // one tick delay (15ms) in between reads for stability
  }
}
void control( void *pvParameters __attribute__((unused)) )  // This is a Task.
{
  int Kp=0.3;
  int Kd=0.1;
  pinMode(13, INPUT);
  int time=millis();
  bool  currentState=digitalRead(ctrl);
  bool  prevState=currentState;
  int Prevroll;
  int pid_p;
  int pid_d;
  for (;;) // A Task shall never return or exit.
  {
    xQueueSend(structQueue, &ctrl, portMAX_DELAY);
   int timePrev=time;
   time=millis();
   int elapsedTime=(time-timePrev)/1000;
    int u;
    if (xQueueReceive(structQueue, &roll, portMAX_DELAY) == pdPASS) {
      int pid_p=Kp*roll;
      int pid_d=Kd*((roll-Prevroll)/elapsedTime);
    }
```

```
      if(currentState!=prevState){
      vTaskDelay(6.66);
        if(currentState!=prevState){
         u=u=pid_p+pid_d;
         //Serial.print(" u = ");
         //Serial.print(u);
         //Serial.println();
        }
        Prevroll=roll;
    }
    xQueueSend(structQueue, &u, portMAX_DELAY);


    //vTaskDelay(1);  // one tick delay (15ms) in between reads for stability
  }
}
void LCD( void *pvParameters __attribute__((unused)) )  // This is a Task.
{
    const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
    LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
    lcd.begin(16, 2);
  for (;;) // A Task shall never return or exit.
  {
   if ( xQueueReceive(structQueue, &roll, portMAX_DELAY) == pdPASS )
    {
```

```
    lcd.setCursor(0,0);
    lcd.print("ANGLE:");
    lcd.setCursor(6,0);
    lcd.print(roll);
   vTaskDelay(3.33);
   }
      // one tick delay (15ms) in between reads for stability
    lcd.clear();

  }
}
  void ESC( void *pvParameters __attribute__((unused)) )  // This is a Task.
{
  int potValue;
  Servo ESC;
  ESC.attach(9,1000,2000);
  potValue = analogRead(A0);
  for (;;) // A Task shall never return or exit.
  {
  potValue = map(potValue, 0, 1023, 0, 180);
  ESC.write(potValue);
  }
  }
```