Mechatronics Final Project Report

Michelle Klein, Johnathan Lao, Aidan Sussman, Roy Zhang

Opportunity:

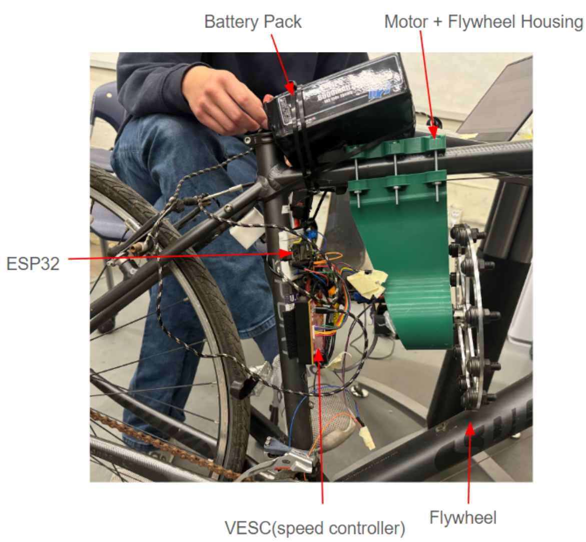
For our final project, we wanted to create a smart environmentally friendly vehicle that is safe to ride around a bustling city. We created a self stabilizing bicycle, a device that can assist with keeping the rider upright, even when biking at low speeds. This technology would help reduce carbon emissions by taking an already existing alternative mode of transportation to driving, which is biking, and making it more safe and stable which would further encourage people to switch to biking instead of driving cars. Our device does not emit any pollutants, which means it is not actively harming the environment.

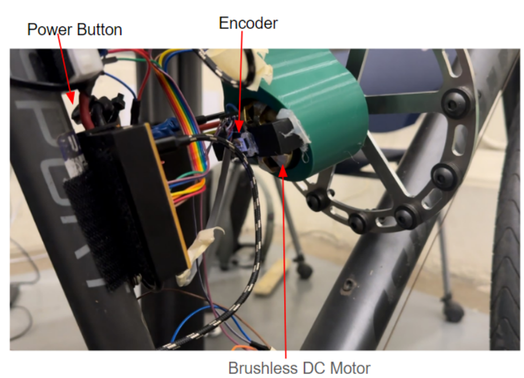
High Level Strategy:

The self-stabilizing bicycle consists of a flywheel inverted pendulum that stores energy in the form of rotational momentum to offset the tilt of the bike when riding. It is equipped with an encoder to measure the tilt/lean angle of the bicycle frame in the lateral direction, which provides real-time feedback on the position of the DC motor. The sensor feedback is then processed on an ESP32 microcontroller to calculate the tilt angle and determine the direction and magnitude of the adjustment needed to keep the bike upright. The generated control signals are then sent to the motor to adjust the position of the flywheel, generating a counteracting torque to bring the bicycle back to an upright position. The use of these real-time corrections counteracts disturbances that are caused by external factors such as wind gusts, uneven terrain, or rider movements.

Our initial desired functionality was to achieve lateral stability with the use of a DC motor encoder and VESC speed controller. Through testing and fine-tuning, the integrated inverted pendulum flywheel system should actively counteract disturbances and keep the bicycle upright.

Photo of Assembly:





Function-Critical Decisions:

One of the main functional-critical decisions we had to make was the selection of a brushless DC motor over a brushed motor. It yields a higher power density and energy efficiency compared to a brushed motor, where energy is partially lost from the friction of the brushes. The main load in the transmission system is the dynamic load experienced during the motion of the bicycle. This is a combination of various factors that can be represented by the tipping bike model, depicted in the figure below.

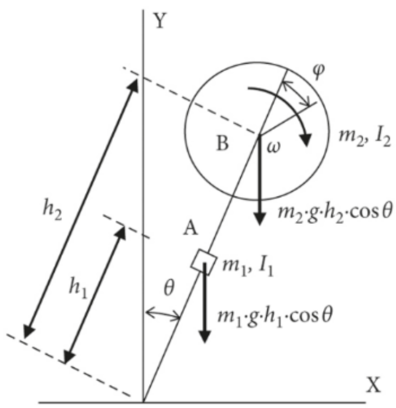


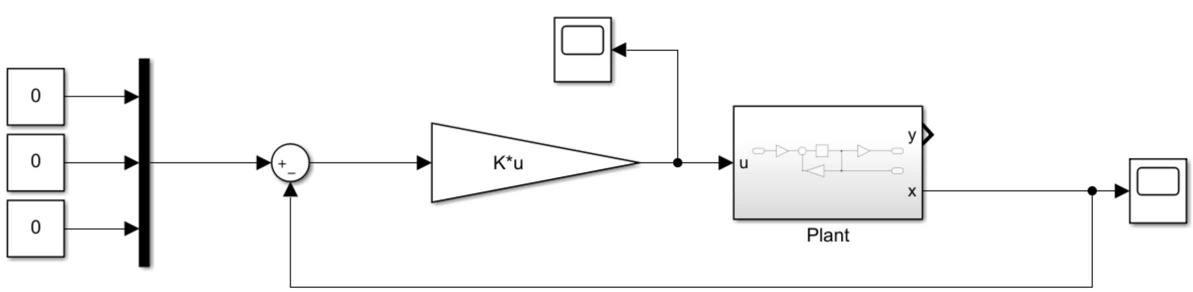
Figure 1: Inverted pendulum diagram

The rig follows the model represented by the Lagrangian formula: **L = T - U**, where **T** is the rotational kinetic energy and **U** is the potential energy. The T is the summation of three components: the kinetic energy of the bike due to tipping, $T_1 = \frac{1}{2} I_1 \theta dot^2$, where θ is the lean angle and I_1 is the moment of inertia of the center of mass; The rotational KE due to spinning of the flywheel, $T_2 = \frac{1}{2} I_2 \varphi dot^2$, where I_2 is the moment of inertia of the flywheel and φ is the rotational speed of the flywheel; The translational KE due to tipping, $T_3 = \frac{1}{2}(m_2 + m_{bike})(h_2 \theta dot)^2$, with m_2 and m_{bike} being the mass of the flywheel and the mass of the bicycle, and h_2 being the length of the pendulum. The formula for the potential energy is defined by the equation $U = (m_1 h_1 + m_2 h_2 + m_{rider} h_{rider}) g cos\theta$, where m_1 and h_1 are the mass of the center of the pendulum's gravity and the distance between the center of rotation and the pendulum's center of gravity, and m_{rider} and h_{rider} being the mass and height of the rider's center of gravity.

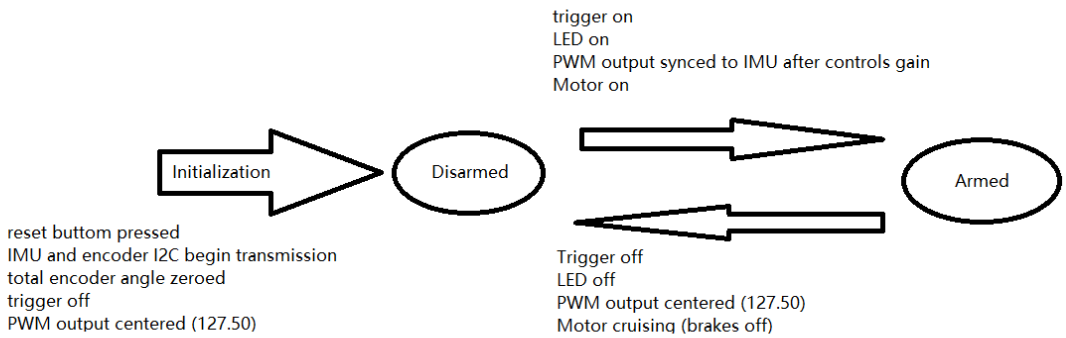
The mass of the flywheel is $m_2 \approx 5.74\ kg$ and the moment of inertia is $I_2 \approx 0.0629\ kg \cdot m^2$. *These full calculations can be found in the appendix.*

Diagrams:

Circuit Diagram



State Transition Diagram



Reflection:

Overall, we are pretty proud of the outcome of our project. In reflection, having group members assigned their general share of tasks corresponding to their strengths and skills was a strategy that worked well for us. One of the biggest challenges with this project was to deal with brushless motor control. We utilized a current based control system using state variables φ, angle of bike relative to calibration, and φ', change in angle of bike. There were a few issues within the control system that created moments of delay and lag that stalled the system's reaction, causing it to reach an unrecoverable state. More research into better brushless motor control techniques along with better overall equipment with higher resolution would allow for a smoother more reliable system. There were also a few reach goals that given the time and resources we'd like to have accomplished. The long term goal and purpose of this project was to create a product that was easy to assemble and use, so streamlining the design, compacting all the components into the main mount body, and creating a more light-weight model that can be uploaded into the product. Additionally we wanted to have the balancing effect scale down its power as the user increases speed. This effect was demonstrated by abstracting the speed input as a potentiometer signal which scaled the balancing effects of the system. This would be beneficial because when in motion the bike creates a natural balancing stabilization. Additionally we wanted to explore how this flywheel could assist when turning, which would require using IMU data to identify a turn and then seeing if the flywheel could be used to assist this action.

Appendix

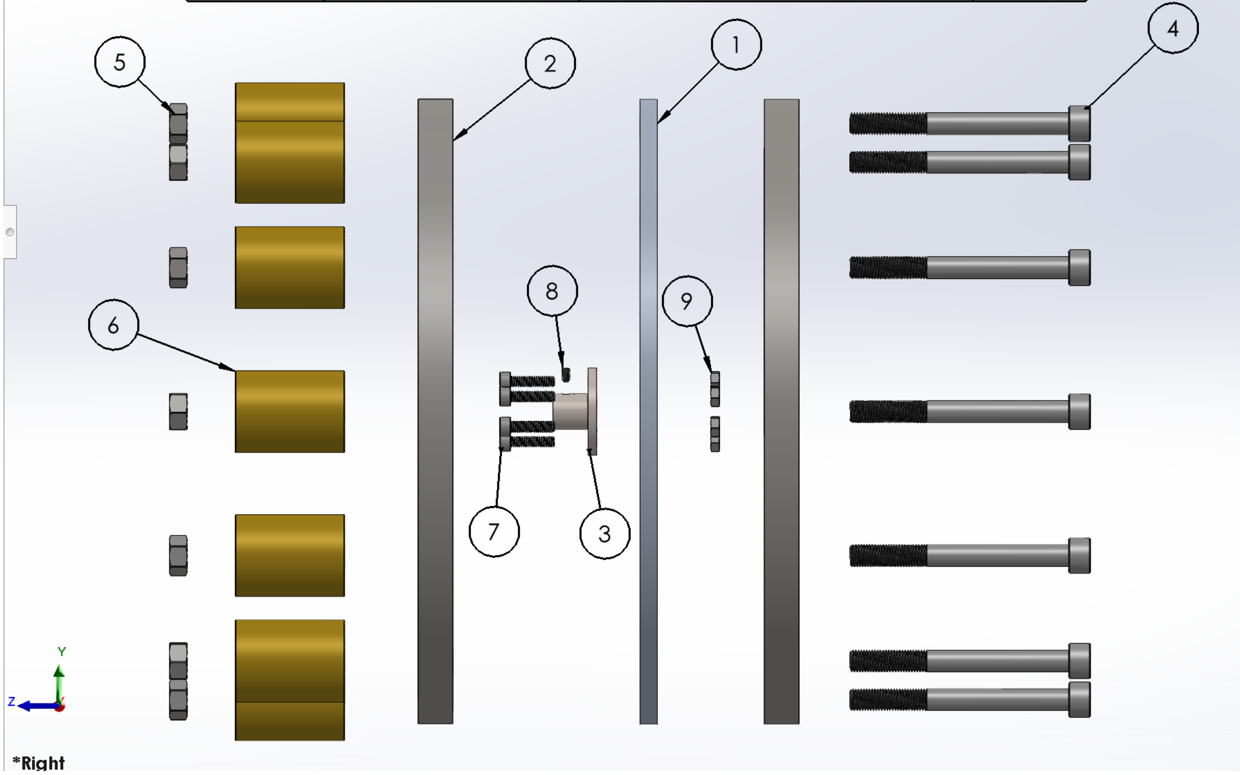
Complete Bill of Materials:

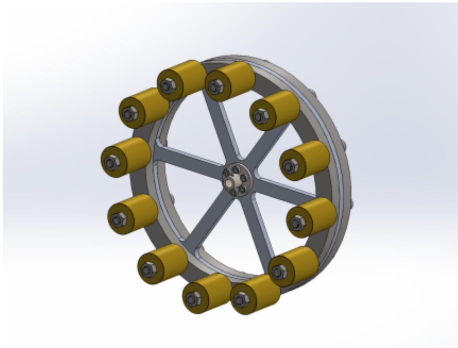
Listing Name	Part No.	Quantity	Cost	Link
¼” 6061 Aluminum Alloy Sheet	9246K11	1	$19.69	https://www.mcmaster.com/9246K11/
8” diameter ½” Low Carbon Steel Disks	7786T271	2	$36.44 each	https://www.mcmaster.com/7786t271/
5mm Flange Shaft Couplers	B07L1FMBBC	1	$9.49	www.amazon.com/Coupling-Support
M8 Fine-Thread Alloy Steel Socket Head Screws	96144A239	12	$8.54	https://www.mcmaster.com/96144A239/
M8 Steel Hex Nut	90592A022	12	$9.67	https://www.mcmaster.com/90592A022/
M8, 16 mm OD Brass Washers	91635A260	12	$15.21	https://www.mcmaster.com/91635A260/
M4 Alloy Steel Socket Head Screw	91290A154	6	$12.91	https://www.mcmaster.com/91290A154/
M3 Alloy Steel Cup-Point Set Screw	91390A099	1	$6.50	https://www.mcmaster.com/91390A099/
M4 Steel Hex Nut	90592A090	6	$3.81	https://www.mcmaster.com/90592A090/
Signwise encoder		1	$17.99	https://www.amazon.com/Signswise-Incremental-Optical-Encoder-Quadrature/dp/B085ZLCYS1?th=1
M8 Fine-Thread Alloy Steel Socket Head Screw	96144A219	1	$9.20	https://www.mcmaster.com/96144A219/
M8 High-Strength Steel Nylon-Insert Locknut	97260A102	2	$8.57	https://www.mcmaster.com/97260A102/

Maytech 100A VESC		1	$237.00	https://electricboardsolutions.com/products/maytech-100a-vesc
Motor: Scorpion SII-4035-450KV	SII-4035-450KV	1	$229.99	https://www.scorpionsystem.com/catalog/aeroplane/motors_1/sii-40/SII-4035-450/
LiPo 8000 10S2P 37v Battery Pack		1	$599.99	https://maxamps.com/collections/10s-lipo-battery-37v/products/lipo-8000-10s2p-37v-battery-pack
M3 Black-Oxide Alloy Steel Hex Drive Flat Head Screw	91294A130	1	$6.36	https://www.mcmaster.com/91294A130/
Ball Bearing, ⅜" Shaft Diameter	60355K165	2	$13.10	https://www.mcmaster.com/60355K165/
High-Strength Steel Threaded Rod, ⅜"-24 Thread, 6" Long	90322A342	1	$11.06	https://www.mcmaster.com/90322A342/
Medium-Strength Steel Hex Nut, Grade 5, 3/8"-24 Thread Size	95505A613	4	$8.58	https://www.mcmaster.com/95505A613/
6061 Aluminum Rod, ½" Diameter	2655N15	1	$10.31	https://www.mcmaster.com/2655N15/
Alloy Steel Socket Head Screw, Black-Oxide, M8 x 1.25 mm Thread, 140 mm Long	91290A476	1	$6.11	https://www.mcmaster.com/catalog/129/3495/91290A476

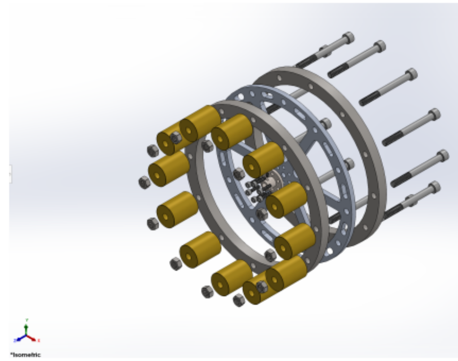
CAD:

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Flywheel_Skeleton	0.25" 6061 Aluminium Alloy	1
2	Flywheel_Steel_Disks	0.5" Low Carbon Steel	2
3	Flanged Shaft Coupling		1
4	96144A239	Fine-Thread Alloy Steel Socket Head Screw	12
5	90592A022	Steel Hex Nut	12
6	91635A260	Brass Washer	12
7	91290A154	Alloy Steel Socket Head Screw	6
8	91390A099	Alloy Steel Cup-Point Set Screw	1
9	90592A090	Steel Hex Nut	6

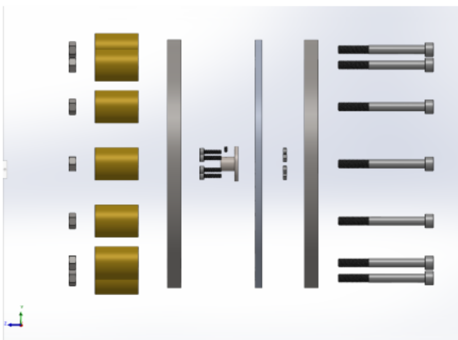




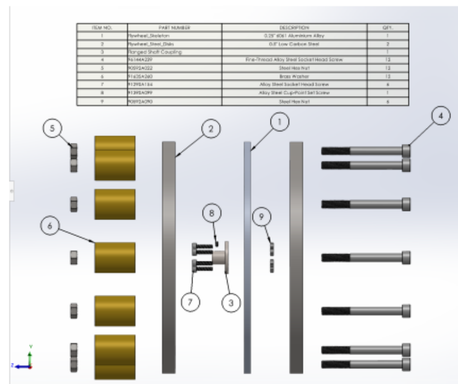
(a) Isometric View



(b) Exploded View



(c) Exploded Side View



(d) BOM

Figure 3: Flywheel CAD views.

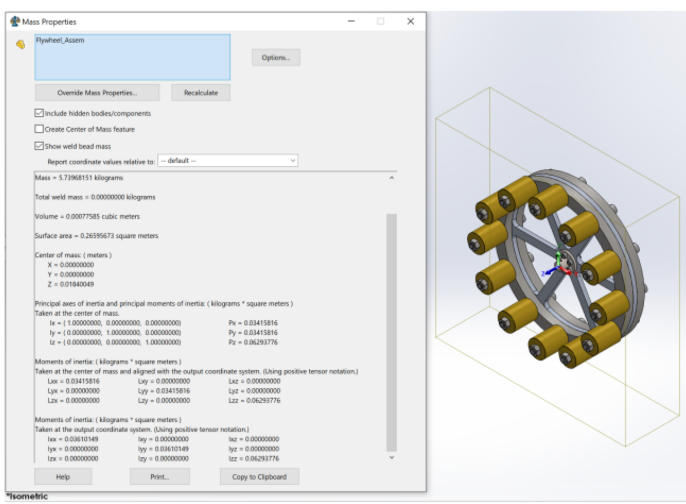
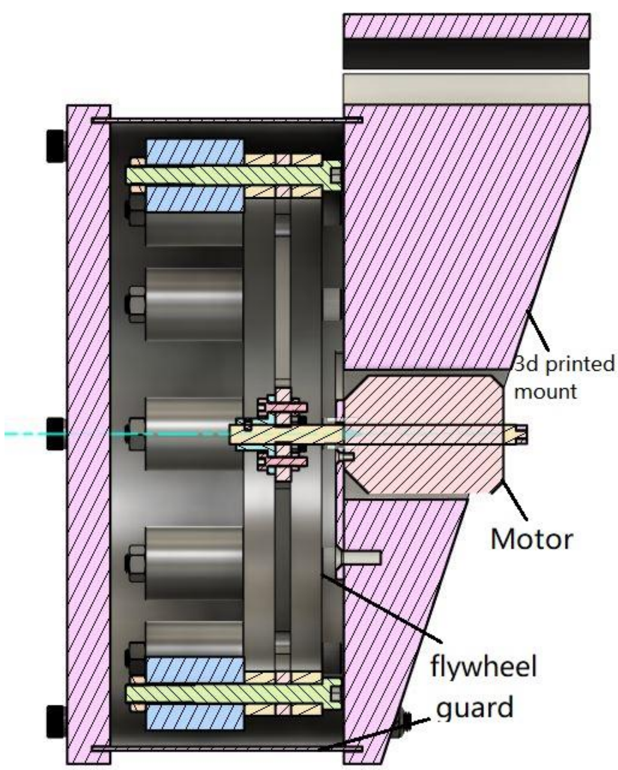
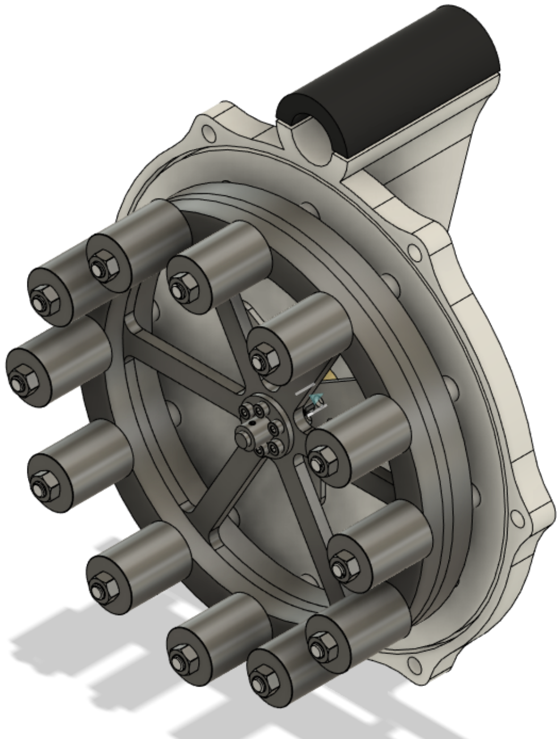


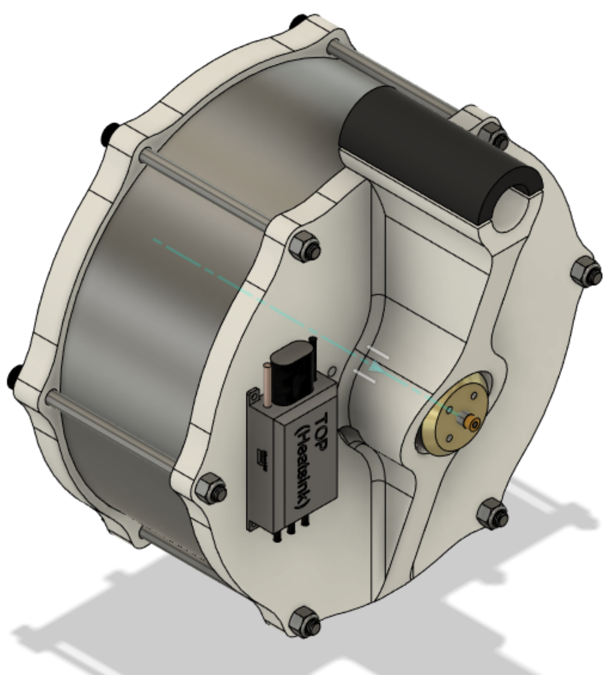
Figure 4: Isometric view with mass properties. Mass: 5.73968151kg and Moment of Inertia: $0.06293776\text{kg} \cdot \text{m}^2$.


GIANT


3d printed
mount
Motor
flywheel
guard



Open view of the flywheel inside the mount: the flywheel is directly mounted to the motor shaft and shrouded with a bent piece of 1/16 Aluminum sheet metal.



Back view of flywheel mount: an encoder is attached to the orange magnet protruding from the back of the motor and a magnetic encoder to detect motor position.

Code:

main_102b.ino

main_102b.ino | IMU6050.cpp | IMU6050.h | controls.cpp | controls.h

```
#include "Wire.h" // This library allows you to communicate with I2C devices.
#include <ezButton.h>
#include <VescUart.h>
#include "IMU6050.h" // abstracted all IMU related code
//#include "encoder.h" // abstracted all encoder related code
#include "controls.h" // abstracted all dynamic controls related code
#define MTR 25 // motor PWM output
#define POT 26 // trigger pin (potentiometer as placeholder)
#define LED_PIN 13 // declare the builtin LED pin number
//hw_timer_t *My_timer = NULL;

ezButton toggleSwitch(34);  // create ezButton object that attach to pin 7;

// Accelerometer variables
const int MPU_ADDR = 0x68; // I2C address of the MPU-6050. If AD0 pin is set to HIGH, the I2C address will be 0x69.
float FIPstate[3] = {0.0,0.0,0.0}; // phi, phi dot, theta dot
float FIPstate_filt[3] = {0.0,0.0,0.0}; // phi, phi dot, theta dot
float disarmstate[3] = {0.0,0.0,0.0};


// Controller variables
float inputVal = 0;
int state = 1;

float AngVel_r = 0;
float AngVel_f = 0;
float AngVel_r1 = 0;
float AngVel_f1 = 0;
float previousTime,previousAngVel;

```

```
VescUart UART;
/*
void IRAM_ATTR onTimer() { // interrupt for rpm calculation
  //RPM = as5600.readAngle();
}*/
template <int order> // order is 1 or 2
class LowPass
{
  private:
    float a[order];
    float b[order+1];
    float omega0;
    float dt;
    bool adapt;
    float tn1 = 0;
    float x[order+1]; // Raw values
    float y[order+1]; // Filtered values

  public:
    LowPass(float f0, float fs, bool adaptive){
      // f0: cutoff frequency (Hz)
      // fs: sample frequency (Hz)
      // adaptive: boolean flag, if set to 1, the code will automatically set
      // the sample frequency based on the time history.

      omega0 = 6.28318530718*f0;
      dt = 1.0/fs;
      adapt = adaptive;
      tn1 = -dt;
      for(int k = 0; k < order+1; k++){
```

```
        x[k] = 0;
        y[k] = 0;
      }
      setCoef();
    }

    void setCoef(){
      if(adapt){
        float t = micros()/1.0e6;
        dt = t - tn1;
        tn1 = t;
      }

      float alpha = omega0*dt;
      if(order==1){
        a[0] = -(alpha - 2.0)/(alpha+2.0);
        b[0] = alpha/(alpha+2.0);
        b[1] = alpha/(alpha+2.0);
      }
      if(order==2){
        float alphaSq = alpha*alpha;
        float beta[] = {1, sqrt(2), 1};
        float D = alphaSq*beta[0] + 2*alpha*beta[1] + 4*beta[2];
        b[0] = alphaSq/D;
        b[1] = 2*b[0];
        b[2] = b[0];
        a[0] = -(2*alphaSq*beta[0] - 8*beta[2])/D;
        a[1] = -(beta[0]*alphaSq - 2*beta[1]*alpha + 4*beta[2])/D;
      }
    }
```

```

    float filt(float xn){
      // Provide me with the current raw value: x
      // I will give you the current filtered value: y
      if(adapt){
        setCoef(); // Update coefficients if necessary
      }
      y[0] = 0;
      x[0] = xn;
      // Compute the filtered values
      for(int k = 0; k < order; k++){
        y[0] += a[k]*y[k+1] + b[k]*x[k];
      }
      y[0] += b[order]*x[order];

      // Save the historical values
      for(int k = order; k > 0; k--){
        y[k] = y[k-1];
        x[k] = x[k-1];
      }

      // Return the filtered value
      return y[0];
    }
};

LowPass<1> lp0(3,5000,true);
LowPass<1> lp1(3,5000,true);

void setup() {
```

```
  Serial.begin(115200);
  // VESC UART serial
  Serial2.begin(115200, SERIAL_8N1, 16, 17);
  UART.setSerialPort(&Serial2);
  Wire.begin();
  Wire.setClock(800000L); //fast clock
  IMU_setup(MPU_ADDR);

  pinMode(MTR, OUTPUT);
  pinMode(POT, INPUT);
  pinMode(LED_PIN, OUTPUT);

  toggleSwitch.setDebounceTime(50); // set debounce time to 50 milliseconds


  delay(1000);
}
void loop() {

  toggleSwitch.loop(); // MUST call the loop() function first
  int toggleState = toggleSwitch.getState();
  IMU_measure(MPU_ADDR,FIPstate); // gather accelerometer and gyroscope data

  unsigned long currentTime = millis();
  if ( UART.getVescValues() ) {
    //float rpm_r = UART.data.rpm;
    AngVel_r = UART.data.rpm * 0.10472;
    //float rpm_f = 0.9*rpm_f1 + 0.05*rpm_r + 0.05*rpm_r1;
    AngVel_f = 0.9*AngVel_f1 + 0.05*AngVel_r + 0.05*AngVel_r1;
    float AngVel_f2 = 0.95*AngVel_f1 + 0.025*AngVel_r + 0.025*AngVel_r1;
```

```
    AngVel_r1 = AngVel_r;
    AngVel_f1 = AngVel_f;

    float dt = currentTime - previousTime;
    FIPstate[2] = (AngVel_f2 - previousAngVel)/(dt/1000);
    previousTime = currentTime;
    previousAngVel = AngVel_f;
  }
  else
  {
    Serial.println(" Failed to get data!");
  }
  FIPstate_filt[0] = lp0.filt(FIPstate[0]);
  Serial.print("PHI:"); Serial.print(FIPstate[0],3);Serial.print(",");
  FIPstate_filt[1] = lp1.filt(FIPstate[1]);
  FIPstate_filt[2] = FIPstate[2];
  Serial.print("PHI:"); Serial.print(FIPstate_filt[0],3);Serial.print(",");
  Serial.print("PHI_dot:"); Serial.print(FIPstate_filt[1],3);Serial.print(",");
  Serial.print("AngAcc:"); Serial.print(FIPstate[2],3);Serial.println(",");

  float scale = floatMap(analogRead(POT), 0, 4096, 0, 1000)/1000;
  //Serial.print("scale:"); Serial.print(scale);Serial.print(",");

  switch (state) {
    case 1: // disarmed
      inputVal = 0;
      UART.setCurrent(scale*inputVal);
      //Serial.print("PWM:"); Serial.print(inputVal); Serial.println("OFF");
      if (toggleState != HIGH) { // trigger event checker (potentiometer as placeholder)
        digitalWrite(LED_PIN, HIGH); // LED toggle for ON OFF indication
```

```
      state = 2;
    }
    break;

  case 2: // armed static
    inputVal = static_input(FIPstate_filt);
    UART.setCurrent(scale*inputVal);
    //Serial.print("PWM:"); Serial.print(inputVal); Serial.println("ON");
    if (toggleState == HIGH) { // trigger event checker (potentiometer as placeholder)
      digitalWrite(LED_PIN, LOW); // LED toggle for ON OFF indication
      state = 1;
    }/*
    else if (analogRead(POT) > 2000) {
      state = 3;
    }*/
    break;
  case 3: // armed rolling
    digitalWrite(LED_PIN, (millis() / 500) % 2);
    inputVal = static_input(FIPstate);
    //analogWrite(MTR,inputVal); // set motor PWM according to controls calculation
    Serial.print("PWM:"); Serial.print(inputVal); Serial.println("ONNNN");
    if (toggleState == HIGH) { // trigger event checker (potentiometer as placeholder)
      digitalWrite(LED_PIN, LOW); // LED toggle for ON OFF indication
      state = 1;
    }
    else if (analogRead(POT) < 2000) {
      digitalWrite(LED_PIN, HIGH); // LED toggle for ON OFF indication
      state = 2;
    }
    break;
  }
}

float floatMap(float x, float in_min, float in_max, float out_min, float out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

IMU6050.cpp

main_102b.ino | IMU6050.cpp | IMU6050.h | controls.cpp | controls.h

```
#include "IMU6050.h"
#include <Kalman.h>
Kalman kalman;

float AccX, AccY, AccZ;
float GyroX, GyroY, GyroZ;
float accAngleX, accAngleY, accAngleZ, gyroAngleX, gyroAngleY, gyroAngleZ;
double pitch;
double zeroValues[3] = { 0 };


unsigned long timer;
Adafruit_MPU6050 mpu;

void IMU_setup(int MPU){
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");
  delay(500);
  calculate_IMU_offset();

  mpu.setAccelerometerRange(MPU6050_RANGE_2_G); // set accelerometer range to +-2G
  mpu.setGyroRange(MPU6050_RANGE_250_DEG); // set gyro range to +- 250 deg/s
  mpu.setFilterBandwidth(MPU6050_BAND_10_HZ); // set filter bandwidth to 5 Hz

  delay(100);
```

```
}

void IMU_measure(int MPU_ADDR,float FIPstate[]){
  IMU_update();
  FIPstate[0] = pitch - zeroValues[0];
  FIPstate[1] = GyroX - zeroValues[1];
  /*
  double accXval = ((double)AccX - AccX_cal);
  double accYval = ((double)AccY - AccY_cal);
  double accZval = ((double)AccZ - AccZ_cal);
  accAngleX =  (atan2(-accYval,-accZval)+PI)*RAD_TO_DEG;
  float gyroRate = GyroX - gyro_cal;
  float pitch = kalman.getAngle(accAngleX, gyroRate, (double)(micros() - timer)/1000000);
  timer = micros();
  */
  /*
  if (a.acceleration.y > 0) {AccX = abs(pitch);}
  else {AccX = -1*abs(pitch);}
  GyroX = g.gyro.x;
  */
}

void calculate_IMU_offset() {
  int const points = 200;
  for (uint8_t i = 0; i < points; i++) {
    IMU_update();
    zeroValues[0] += pitch;
    zeroValues[1] += GyroX;
  }

  zeroValues[0] = zeroValues[0] / points;
  zeroValues[1] = zeroValues[1] / points;
  Serial.print(zeroValues[0]);
  Serial.print("///");
  Serial.println(zeroValues[1]);
}

void IMU_update() {
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  accAngleX = atan(a.acceleration.y / sqrt(pow(a.acceleration.x, 2) + pow(a.acceleration.z, 2)));
  gyroAngleX = gyroAngleX + g.gyro.x * (double)(micros() - timer)/1000000; // deg/s * s = deg
  gyroAngleX = 0.5 * gyroAngleX + 0.5 * accAngleX;
  timer = micros();

  if (a.acceleration.y > 0) {pitch = abs(gyroAngleX);}
  else {pitch = -1*abs(gyroAngleX);}

  GyroX = g.gyro.x;
}
```

IMU6050.h

main_102b.ino | IMU6050.cpp | IMU6050.h | controls.cpp | controls.h

```cpp
#ifndef IMU6050_H
#define IMU6050_H

#include <Arduino.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include "Wire.h" // This library allows you to communicate with I2C devices.


void IMU_setup(int MPU);
void IMU_measure(int MPU_ADDR,float FIPstate[]);
void calculate_IMU_offset();
void IMU_update();


#endif
```

control.cpp

main_102b.ino | IMU6050.cpp | IMU6050.h | controls.cpp | controls.h

```cpp
#include "controls.h"
//double K[3] = {-647.08258,-169.64748,-0.10000};
double K[3] = {-647.08258,-269.64748,-0.10000};
float const max_u = 50;

float static_input(float FIPstate[]) {
  float u = (FIPstate[0]*K[0] + FIPstate[1]*K[1] + FIPstate[2]*K[2]);
  if (u > max_u) {
    return max_u;
  }
  else if (u < -1*max_u) {
    return -1*max_u;
  }
  else {
    return u;
  }
}

float rolling_input(float POT, double FIPstate[]) {

  float scale = (4095 - POT)/4095;
  float u = -0.65*scale*(FIPstate[0]*K[0] + FIPstate[1]*K[1] + FIPstate[2]*K[1]);
  if (PWM_neutral + u > 255) {
    return 255;
  }
  else if (PWM_neutral + u < 0) {
    return 0;
  }
  else {
    return PWM_neutral + u;
  }
};
```

controls.h

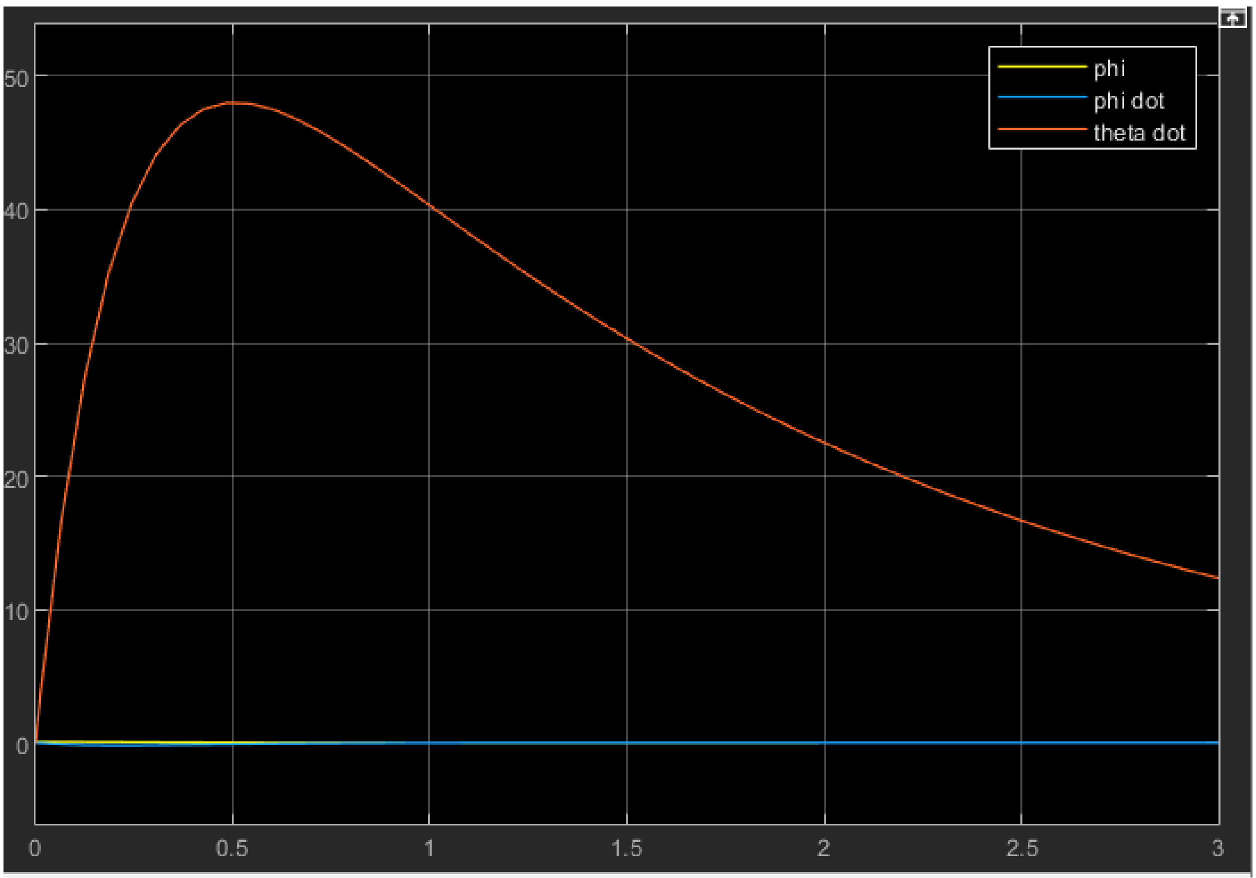
main_102b.ino | IMU6050.cpp | IMU6050.h | controls.cpp | controls.h

```
#ifndef CONTROLS_H
#define CONTROLS_H

#include <Arduino.h>

float static_input(float FIPstate[]);
float rolling_input(float POT, double FIPstate[]);

#endif
```



Dynamic System Analysis & Calculations:

Tipping Bike Model:

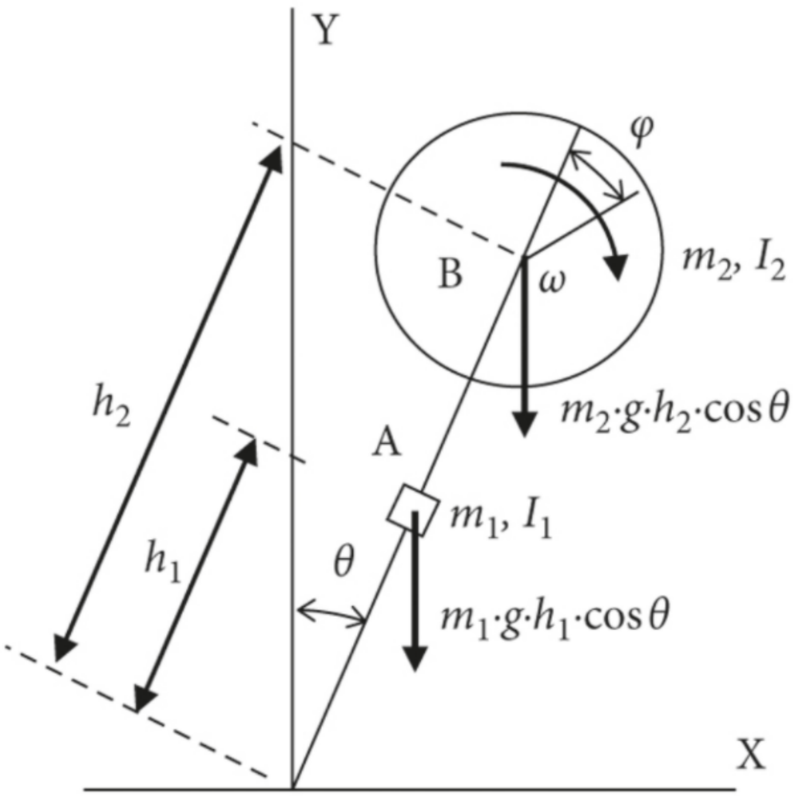


Figure 1: Inverted pendulum diagram

Lagrangian:

$L = T - U$, where T is the kinetic energy and U is the potential energy.

Kinetic Energy: $\frac{1}{2}I\omega^2$

$$\text{Bike: } T_1 = \frac{1}{2}I_1\dot{\theta}^2 \text{ (due to tipping)}$$

$$\text{Flywheel: } T_2 = \frac{1}{2}I_2\dot{\phi}^2 \text{ (rotational KE due to spinning)}$$

$$T_3 = \frac{1}{2}m_2(h_2\dot{\theta})^2 \text{ (translational KE due to tipping)}$$

$$\text{Total: } T = T_1 + T_2 + T_3 = \frac{1}{2}I_1\dot{\theta}^2 + \frac{1}{2}I_2\dot{\phi}^2 + \frac{1}{2}m_2(h_2\dot{\theta})^2$$

Potential Energy:
Bike: $U = (m_1h_1 + m_2h_2)g\cos\theta$

$$L = T_1 + T_2 - U = \frac{1}{2}I_1\dot{\theta}^2 + \frac{1}{2}I_2\dot{\phi}^2 + \frac{1}{2}m_2(h_2\dot{\theta})^2 - (m_1h_1 + m_2h_2)g\cos\theta$$

$$L = \frac{1}{2}I_1\dot{\theta}^2 + \frac{1}{2}I_2\dot{\phi}^2 + \frac{1}{2}m_2h_2^2\dot{\theta}^2 - (m_1h_1 + m_2h_2)g\cos\theta$$

Lagrangian Method:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = Q, \text{ where } Q \text{ represents external forces.}$$

$$\frac{\partial L}{\partial \dot{\theta}} = I_1\dot{\theta} + m_2h_2^2\dot{\theta}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) = I_1\ddot{\theta} + m_2h_2^2\ddot{\theta}$$

$$\frac{\partial L}{\partial \theta} = (m_1h_1 + m_2h_2)g\sin\theta$$

$$I_1\ddot{\theta} + m_2h_2^2\ddot{\theta} - (m_1h_1 + m_2h_2)g\sin\theta = Q$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\phi}}\right) - \frac{\partial L}{\partial \phi} = Q$$

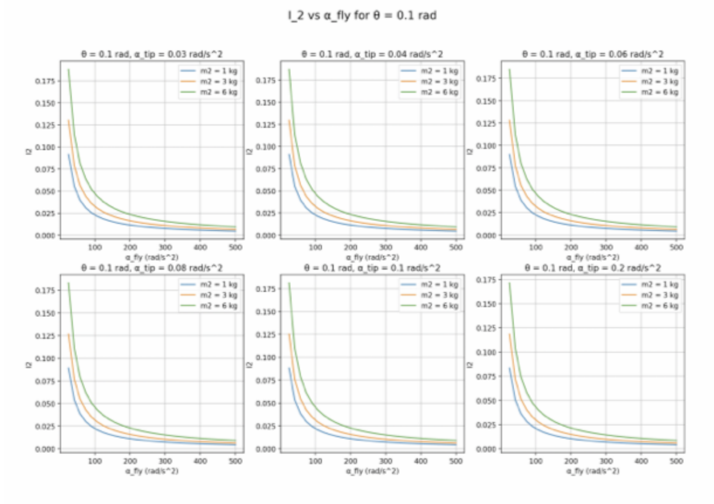
$$\frac{\partial L}{\partial \dot{\phi}} = I_2\dot{\phi}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\phi}}\right) = I_2\ddot{\phi}$$

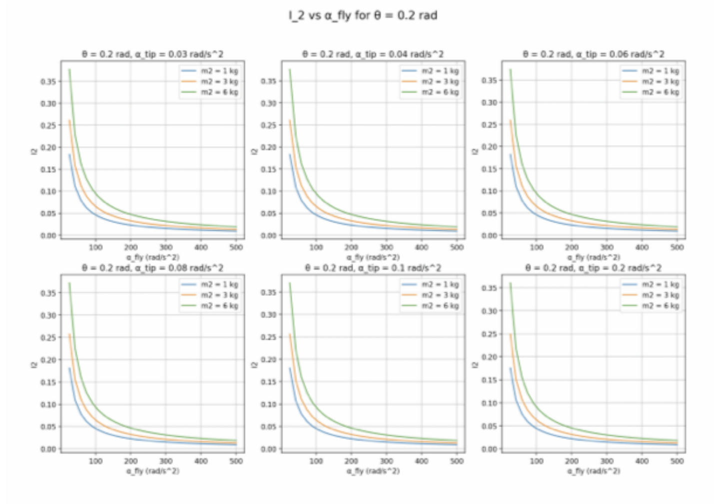
$$\frac{\partial L}{\partial \phi} = 0$$

$$I_2\ddot{\phi} = Q$$

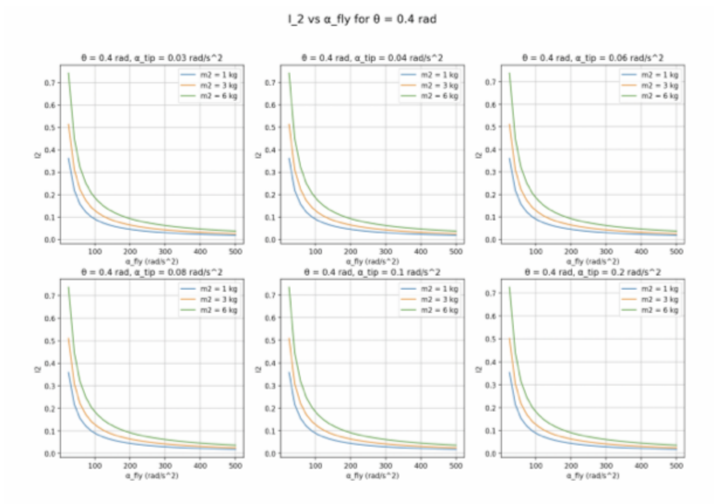
$$I_2\ddot{\phi} = I_1\ddot{\theta} + m_2h_2^2\ddot{\theta} - (m_1h_1 + m_2h_2)g\sin\theta$$



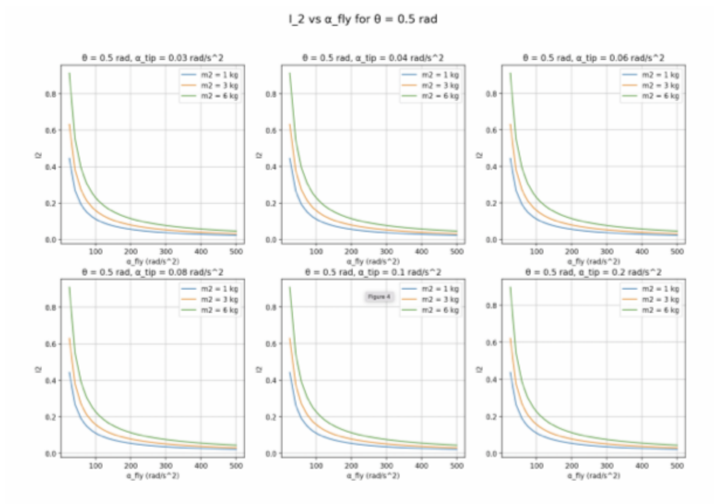
(a) $\theta = 0.1$ rad



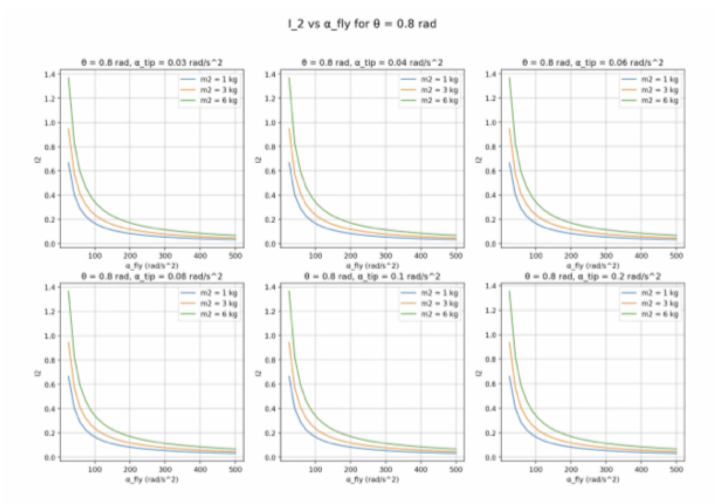
(b) $\theta = 0.2$ rad



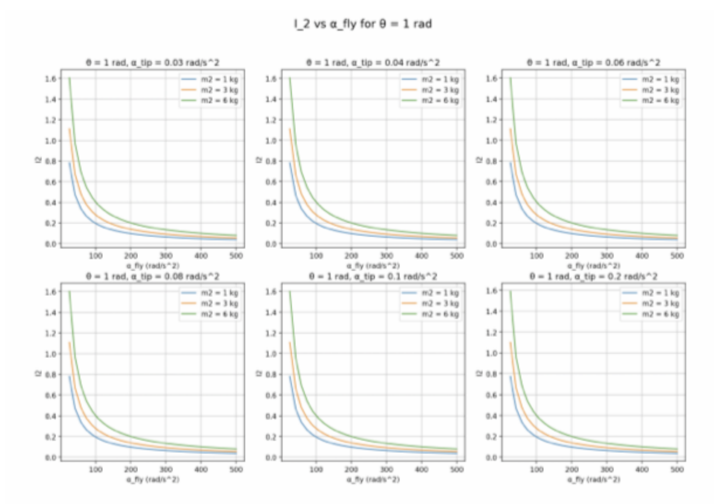
(c) $\theta = 0.4$ rad



(d) $\theta = 0.5$ rad



(e) $\theta = 0.8$ rad



(f) $\theta = 1$ rad

Figure 2: Relationship between the moment of inertia I_2 and the angular acceleration of the flywheel a_{fly} for various values of the angle θ. Each graph showcases the effects of different flywheel masses $m_2 = 1, 3,$ and 6 kg and angular acceleration of tipping values a_{tip}. These plots provide insights into the intricate interplay of system parameters and their impact on a_{fly}, highlighting the significance of our derived mechanics equations.

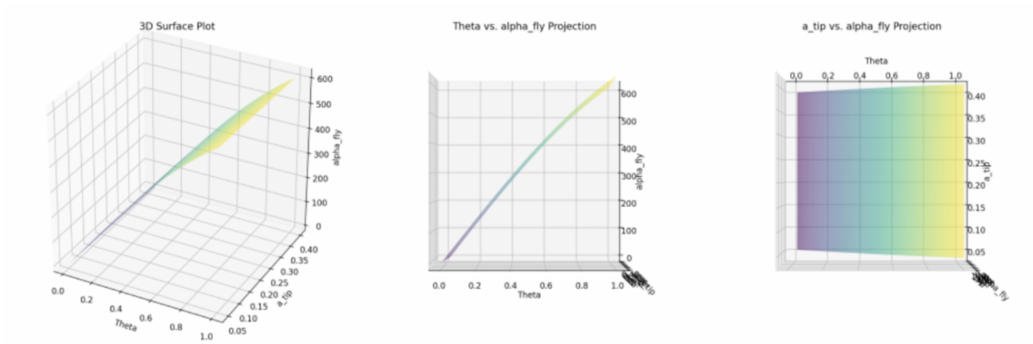


Figure 5: 3D plot of the angular acceleration of the flywheel (a_{fly}) against the angle θ and the angular acceleration of tipping (a_{tip}), alongside its 2D projection showcasing the individual effects of changing θ and a_{tip} on a_{fly}.

Parameters:

- **Mass of Bike (m_1):** 5kg
- **Height of Bike (h_1):** 0.75m (height from ground to flywheel)
- **Height from Ground to Flywheel (h_2):** 0.5m
- **Weight of Flywheel (m_2):** 5.73968151kg
- **Moment of Inertia of Flywheel (I_2):** 0.06293776kg·m^2
- $a_{\textbf{tip}}$ **Range:** [0.05, 0.4] rad/s^2
- θ **Range:** [0.01, 1] rad

Through this investigation, we aim to understand the influence of varying the angle θ and the angular acceleration of tipping a_{tip} on the angular acceleration of the flywheel a_{fly}. The 2D projection offers insights into the individual impacts of each parameter on the system's dynamics.