Design and Testing of a 2-Axis Thrust Vector Control and Roll Control System for Rockets

Saroj Gaire

Undergraduate Student, Thapathali Campus, IOE, Tribhuvan University, Nepal

E-mail: sarojgaire4@gmail.com

Abstract

This project focuses on development and testing thrust vector control (TVC) and roll control (RC) systems specifically designed for small solid-propellant rockets. This control system ensures the stability and maneuverability of the rocket during flight. The key components used in this system include a microcontroller (ESP32), an Inertial Measurement Unit (IMU)-MPU-6050, and servo motors (SG90), which dynamically adjust the thrust direction opposite to the rocket's trajectory to maintain stability. Additionally, other basic components include the battery to power the electrical components, the PVC pipe to form the rocket's body, and various electronic components. The software used in this project includes Arduino IDE, CAD (Onshape, AutoCAD), etc.

Keywords: Thrust Vector Control (TVC) - Roll Control (RC) – Inertial Measurement Unit (IMU)

1. INTRODUCTION

Thrust vector control (TVC) is the method of adjusting the direction of the exhaust gas flow to maintain and manipulate the orientation of rocket. TVC is crucial for trajectory correction during launch and orbital maneuvers. It counteracts unwanted forces or torques keeping the rocket stable. In the vacuum of space or at high speeds where aerodynamic control is ineffective, TVC becomes primary method for controlling rocket orientation. Thrust vectoring can be achieved by different methods like gimbaled engine(s) or nozzle(s), auxiliary (vernier) thrusters, exhaust vanes, reactive fluid injection etc. In this project TVC is achieved by two axes gimbal. The schematic diagram of TVC is given in figure 1.

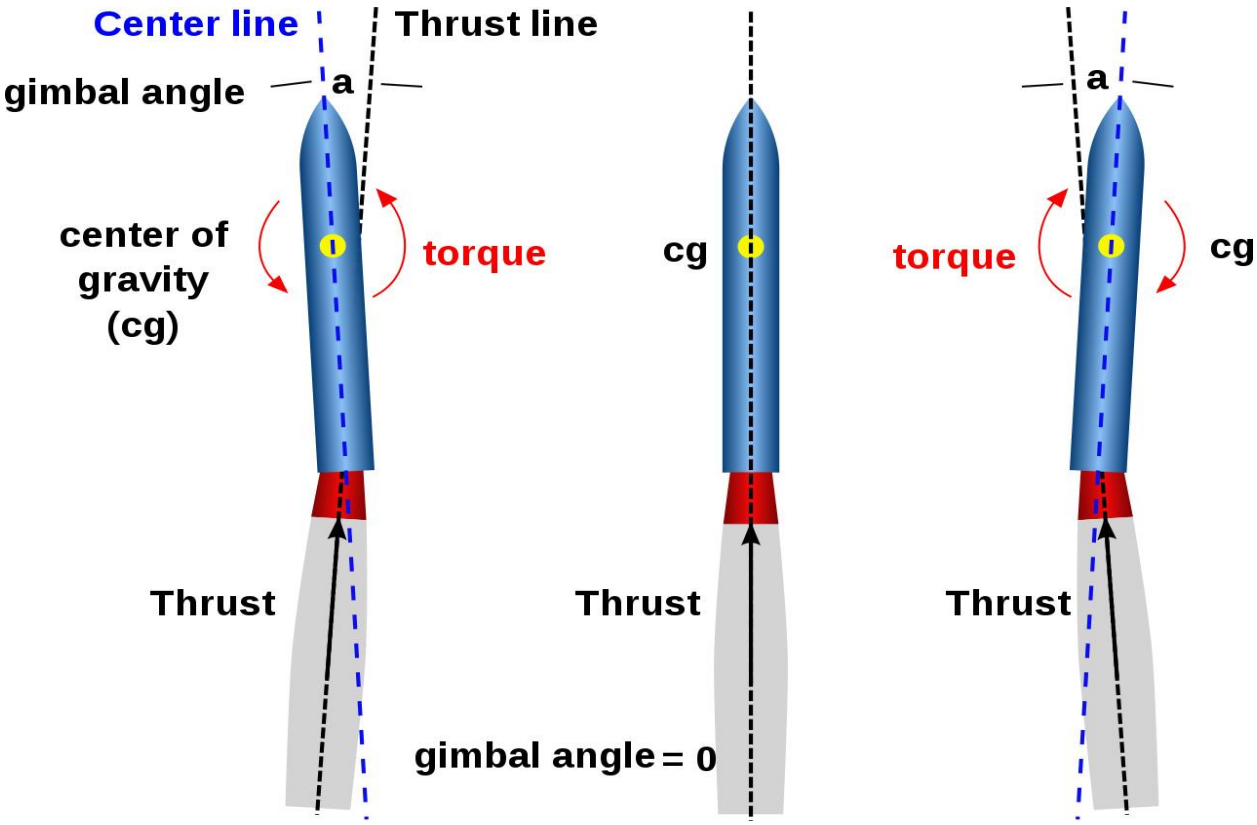

Figure 1: Thrust vectoring in rockets

Roll control is an aerodynamic control system used in rocket fins to create torque about its vertical axis. It controls the unwanted roll in rocket due to manufacturing imperfections so that it didn't tear itself apart. During launch, rockets need to roll to align with their intended flight path or azimuth. This ensures that the rocket is travelling in the correct direction for orbital insertion or reaching its target. By controlling the roll, engineers can ensure that rocket engines are properly oriented for optimal thrust and maneuverability. This can also help in aligning antennas for communication with ground control. To change the aerodynamic shape of fins end part of fin is changed by servo actuators to create torque about vertical axis as in figure 2.

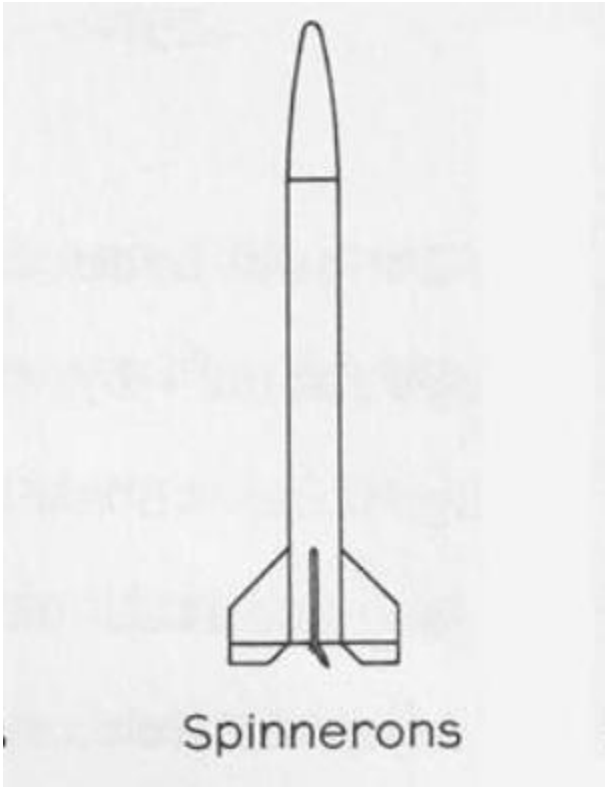

Figure 2: Roll control using aerodynamic control

2. METHOLOGY

In this project, TVC is simply two two-axis gimbals, each axis controlled by a servo motor as an actuator to tilt the nozzle of rocket exhaust, which creates torque about the center of gravity in the horizontal axis, which is used to counteract the tilt or to make a tilt. The roll control is based on aerodynamic control, i.e., changing the aerodynamic shape of rocket fins to create torque in the vertical axis, which is used to counteract the roll or to make roll. Each servo is controlled by a programmed microcontroller based on the orientation of the rocket measured by the IMU sensor. The 3D model of the thrust vector control and roll control have been generated using onshape CAD.

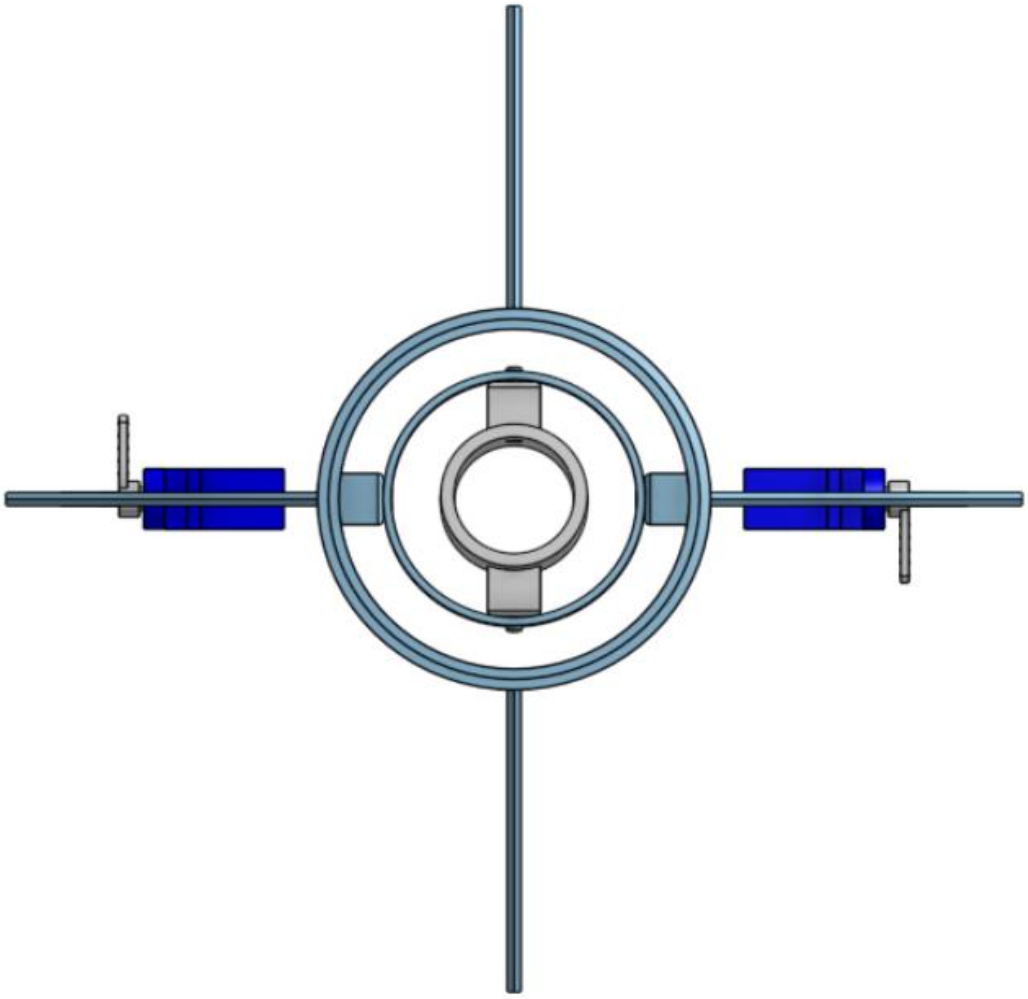

Figure 3: TVC + RC top view (3D model) using onshape

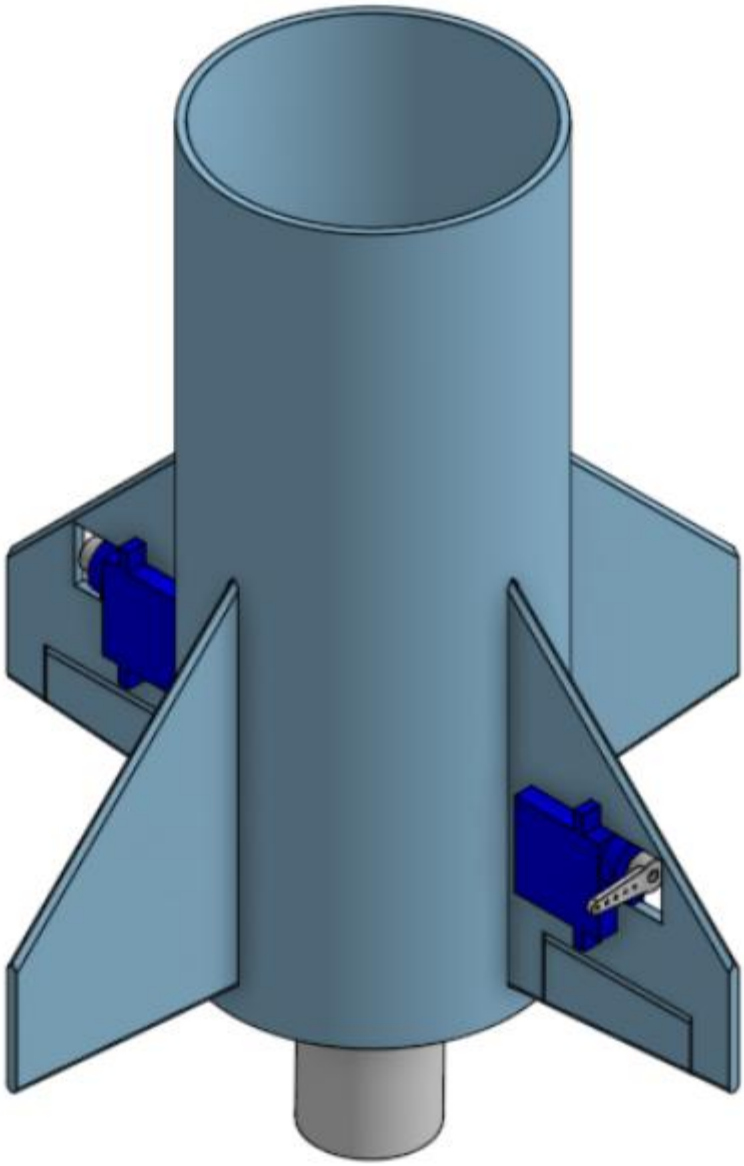

Figure 4: TVC + RC isometric view (3D model) using onshape

This 3D model has been made using pvc pipe by making sheet. And the gimbals are made using circular rings of pipe and wire. The fins used in this model are trapezoidal fins as shown in figure 5.

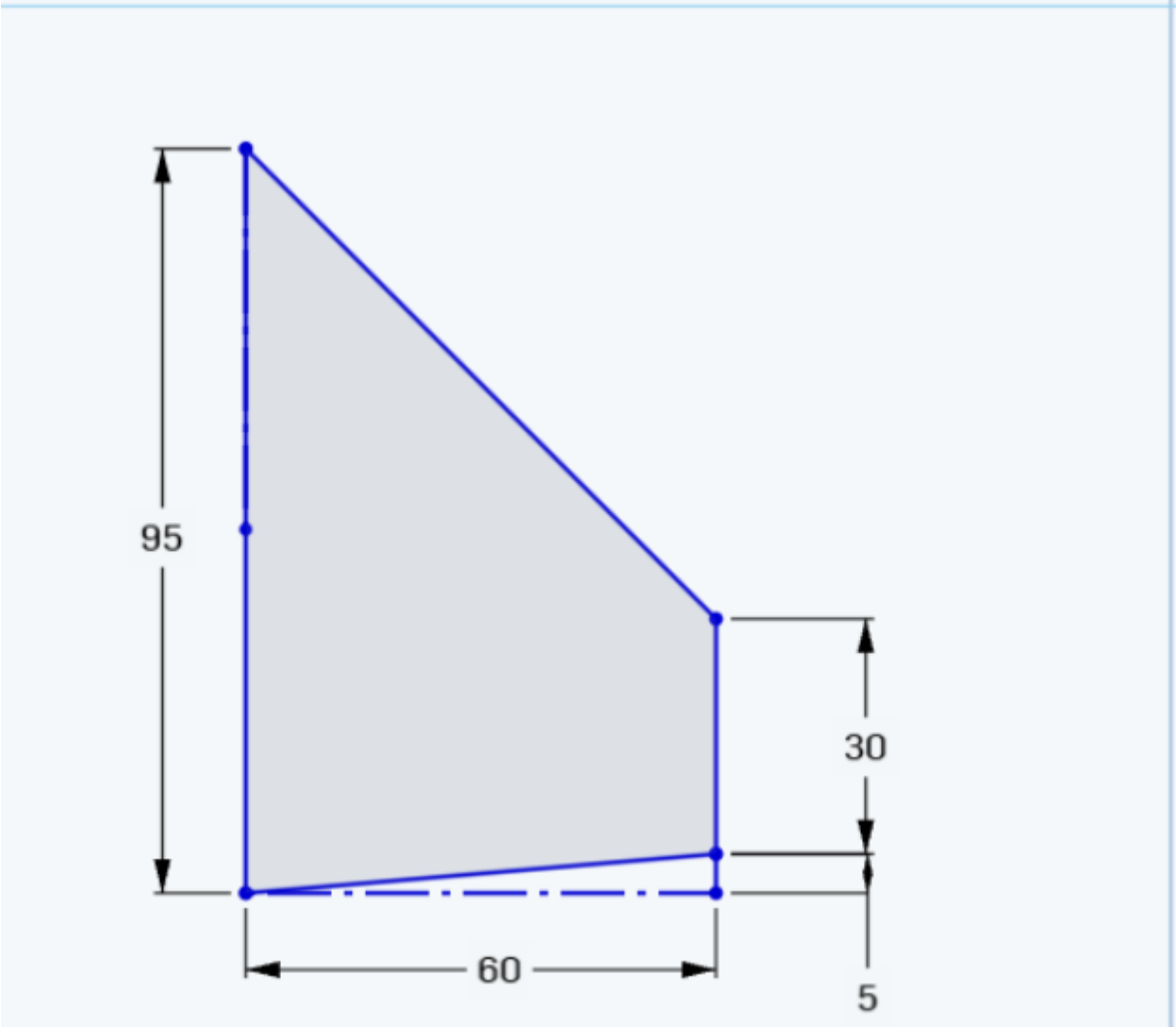

Figure 5: Trapezoidal fin (2D drawing) using onshape

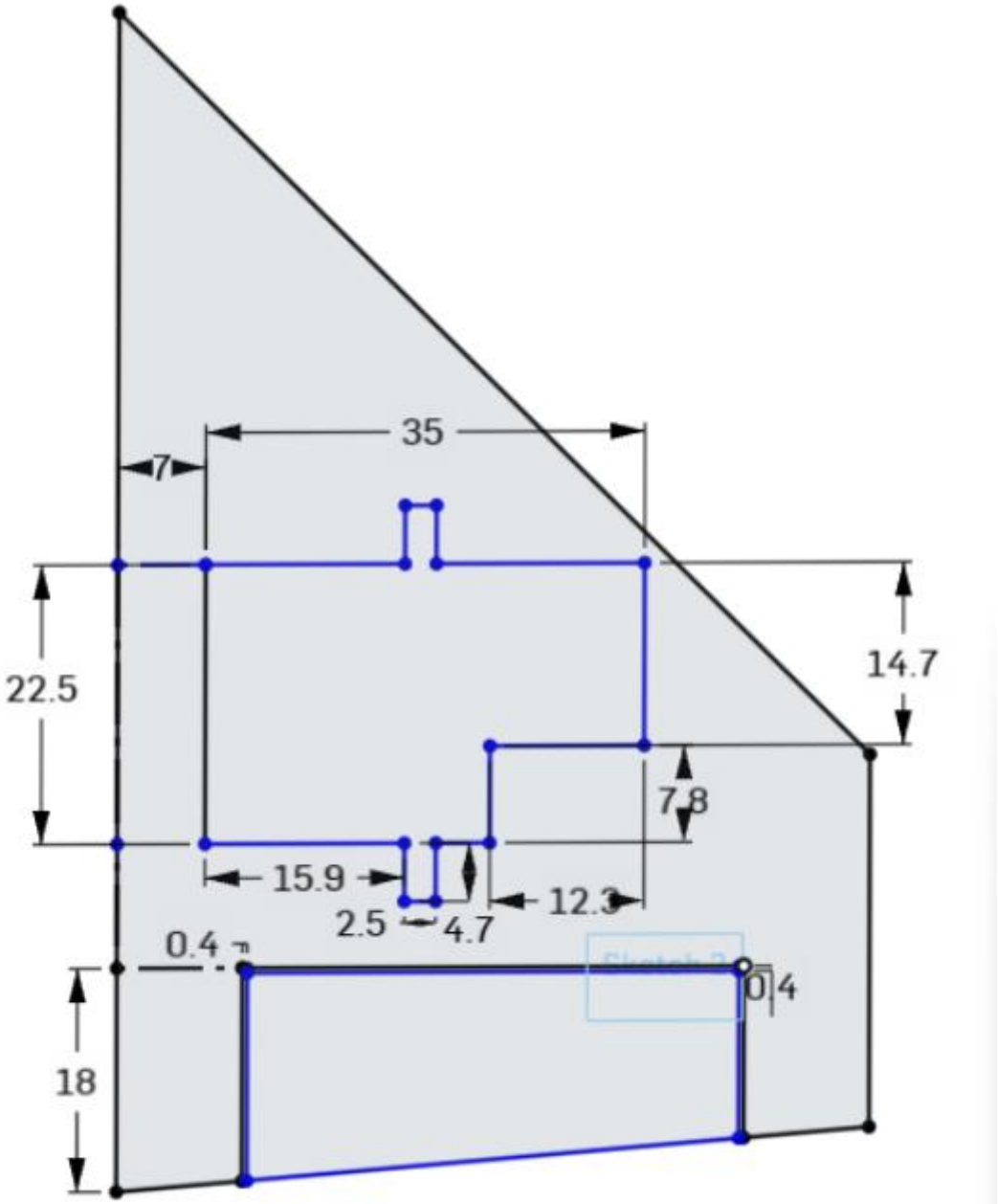

Figure 6: Proper cutout for servo mount and spinnerons

The microcontroller (ESP32), IMU (MPU-6050) and servos (SG90) are the main components in the circuit to make the thrust vectoring autonomously in loop as shown in figure 7. And the main circuit diagram is in the figure 8.

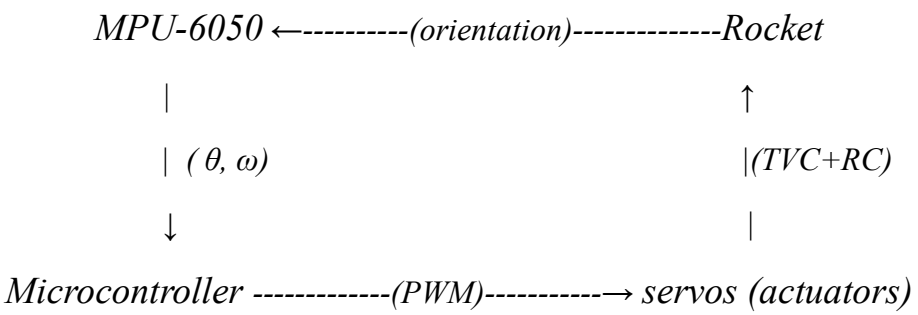

Figure 7: control loop

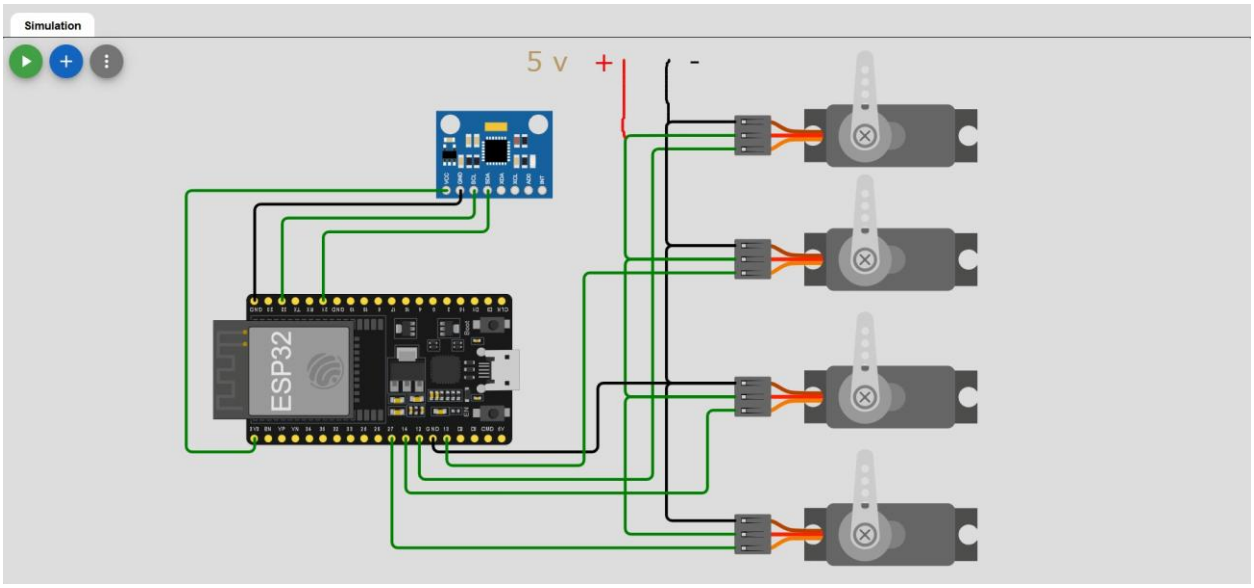

Figure 8: Circuit diagram using Wokwi

3. Results and conclusion

The thrust vector control and roll control developed in this project was working fine after several iterations. The development was implemented using an ESP32 microcontroller, an MPU6050 and four servos. Two servos control the yaw and pitch of rocket via two gimbals and two servos combinedly manage roll. While mimicking the roll and tilts in rocket, the servo actions were found correct and are quite responsive for small solid-propellent rockets with some adjustments like high ampere power supply for servos for quicker response.

The test was carried out with both PID control loop and filtered data loop (without any control algorithms). Both systems are fine while PID loop is accurate and more fused with hardware due to calibration and good choice to use in rockets. But the filtered data loop was good for demonstrating the logic behind TVC and RC systems as it gives good visual concept while mimicking the roll and tilt in rocket. As filtered data loop only uses complementary filter to merge the gyroscope and accelerometer data to use directly in control logic it is not much useable in real rocket tests.

Figure 9: Circuit assembly

Figure 10: Final assembly

4. APPENDIX

- Code with filtered data loop

```
#include <Wire.h>
#include <ESP32Servo.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>

Servo servoX;
Servo servoY;
Servo roll1;
Servo roll2;
```

```
const int pinX = 13;
const int pinY = 12;
const int pinRoll1 = 14;
const int pinRoll2 = 27;
const int ledPin = 2;

Adafruit_MPU6050 mpu;

const int center = 90;
const int radius = 58;
const int rollAmplitude = 38;
const float tiltRange = 15.0;
const float gimbalDeadzone = 3.0;

int lastX = -1, lastY = -1, lastRoll = -1;

float gyroZoffset = 0;
float compPitch = 0, compRoll = 0;
unsigned long lastUpdate = 0;
unsigned long startTime;
bool demoDone = false;
bool centerPauseDone = false;

const float alpha = 0.98; // Complementary filter value

void calibrateGyro() {
  Serial.println("Calibrating gyro... Keep MPU6050 still.");
  float sum = 0;
  int samples = 500;
  for (int i = 0; i < samples; i++) {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    sum += g.gyro.z;
    delay(2);
  }
  gyroZoffset = sum / samples;
  Serial.print("Gyro Z offset: ");
  Serial.println(gyroZoffset * 57.2958);
}

//function for demo motion
void runDemoMotion(float elapsed) {
  int demoX = center + radius * sin(elapsed);
  int demoY = center + radius * cos(elapsed);
  int demoRoll = center + rollAmplitude * sin(elapsed * 2);
```

```
  servoX.write(demoX);
  servoY.write(demoY);
  roll1.write(demoRoll);
  roll2.write(demoRoll);

  digitalWrite(ledPin, (millis() / 500) % 2 == 0 ? HIGH : LOW);
}

void setup() {
  Serial.begin(115200);
  while (!Serial) delay(10);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  if (!mpu.begin()) {
    Serial.println("MPU6050 not found!");
    while (1) delay(10);
  }

  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

  servoX.setPeriodHertz(50);
  servoY.setPeriodHertz(50);
  roll1.setPeriodHertz(50);
  roll2.setPeriodHertz(50);

  servoX.attach(pinX);
  servoY.attach(pinY);
  roll1.attach(pinRoll1);
  roll2.attach(pinRoll2);

  servoX.write(center);
  servoY.write(center);
  roll1.write(center);
  roll2.write(center);

  delay(2000);
  startTime = millis();
  lastUpdate = millis();
}

void loop() {
```

```
  unsigned long now = millis();
  float elapsed = (now - startTime) / 1000.0;

  // === Run demo once ===
  if (!demoDone) {
    runDemoMotion(elapsed);

    if (elapsed > 10) {
      demoDone = true;
      digitalWrite(ledPin, LOW);
      servoX.write(center);
      servoY.write(center);
      roll1.write(center);
      roll2.write(center);
      startTime = millis();
      lastUpdate = millis();
    }

    delay(10);
    return;
  }

  // === Pause before control ===
  if (!centerPauseDone) {
    if (elapsed > 2) {
      centerPauseDone = true;
      calibrateGyro();
    }
    return;
  }

  // === Sensor reading ===
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  float dt = (millis() - lastUpdate) / 1000.0;
  lastUpdate = millis();

  float accelPitch = atan2(-a.acceleration.x, sqrt(a.acceleration.y *
a.acceleration.y + a.acceleration.z * a.acceleration.z)) * 180.0 / PI;
  float accelRoll = atan2(a.acceleration.y, a.acceleration.z) * 180.0 / PI;
  float gyroPitchRate = g.gyro.y * 57.2958;
  float gyroRollRate  = g.gyro.x * 57.2958;
  float gyroZrate = (g.gyro.z - gyroZoffset) * 57.2958;
```

```
  if (abs(gyroZrate) < 0.8) gyroZrate = 0;

  compPitch = alpha * (compPitch + gyroPitchRate * dt) + (1 - alpha) *
accelPitch;
  compRoll  = alpha * (compRoll  + gyroRollRate  * dt) + (1 - alpha) * accelRoll;

  compPitch = constrain(compPitch, -tiltRange, tiltRange);
  compRoll = constrain(compRoll, -tiltRange, tiltRange);

  int servoPitch = abs(compPitch) < gimbalDeadzone ? center : map(compPitch, -
tiltRange, tiltRange, center - radius, center + radius);
  int servoRoll = abs(compRoll) < gimbalDeadzone ? center : map(compRoll, -
tiltRange, tiltRange, center - radius, center + radius);
  int servoRollControl = gyroZrate == 0 ? center :
                         map(constrain(gyroZrate, -100, 100), -100, 100, center -
rollAmplitude, center + rollAmplitude);

  if (servoPitch != lastX) {
    servoX.write(servoPitch);
    lastX = servoPitch;
  }
  if (servoRoll != lastY) {
    servoY.write(servoRoll);
    lastY = servoRoll;
  }
  if (servoRollControl != lastRoll) {
    roll1.write(servoRollControl);
    roll2.write(servoRollControl);
    lastRoll = servoRollControl;
  }

  Serial.print("Pitch: ");
  Serial.print(compPitch);
  Serial.print(" | yaw: ");
  Serial.print(compRoll);
  Serial.print(" | GyroZ: ");
  Serial.print(gyroZrate);
  Serial.print(" | Roll Servo: ");
  Serial.println(servoRollControl);

  delay(5);
}
```

- Code with PID loop

```
#include <Wire.h>
#include <ESP32Servo.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <PID_v1.h>

Servo servoX, servoY, roll1, roll2;

const int pinX = 13, pinY = 12, pinRoll1 = 14, pinRoll2 = 27;
const int ledPin = 2;

Adafruit_MPU6050 mpu;

const int center = 90;
const int radius = 58;
const int rollAmplitude = 38;
const float tiltRange = 15.0;
const float gimbalDeadzone = 3.0;
const float alpha = 0.98;

int lastX = -1, lastY = -1, lastRoll = -1;

float gyroZoffset = 0;
float compPitch = 0, compRoll = 0;

unsigned long lastUpdate = 0;
unsigned long startTime;
bool demoDone = false, centerPauseDone = false;

// PID constants (easily changeable)
double Kp = 4.0, Ki = 0.5, Kd = 0.05;

// PID variables
double pitchSetpoint = 0, pitchInput = 0, pitchOutput = 0;
```

```
double rollSetpoint = 0, rollInput = 0, rollOutput = 0;
double yawSetpoint = 0, yawInput = 0, yawOutput = 0;

PID pidPitch(&pitchInput, &pitchOutput, &pitchSetpoint, Kp, Ki, Kd, DIRECT);
PID pidRoll(&rollInput, &rollOutput, &rollSetpoint, Kp, Ki, Kd, DIRECT);
PID pidYaw(&yawInput, &yawOutput, &yawSetpoint, Kp, Ki, Kd, DIRECT);

void calibrateGyro() {
  Serial.println("Calibrating gyro... Keep MPU6050 still.");
  float sum = 0;
  for (int i = 0; i < 500; i++) {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    sum += g.gyro.z;
    delay(2);
  }
  gyroZoffset = sum / 500;
  Serial.print("Gyro Z offset: ");
  Serial.println(gyroZoffset * 57.2958);
}

void runDemoMotion(float elapsed) {
  int demoX = center + radius * sin(elapsed);
  int demoY = center + radius * cos(elapsed);
  int demoRoll = center + rollAmplitude * sin(elapsed * 2);

  servoX.write(demoX);
  servoY.write(demoY);
  roll1.write(demoRoll);
  roll2.write(demoRoll);

  digitalWrite(ledPin, (millis() / 500) % 2 == 0 ? HIGH : LOW);
}

void setup() {
  Serial.begin(115200);
  while (!Serial) delay(10);

  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  if (!mpu.begin()) {
    Serial.println("MPU6050 not found!");
    while (1) delay(10);
  }
```

```
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

  servoX.setPeriodHertz(50);
  servoY.setPeriodHertz(50);
  roll1.setPeriodHertz(50);
  roll2.setPeriodHertz(50);

  servoX.attach(pinX);
  servoY.attach(pinY);
  roll1.attach(pinRoll1);
  roll2.attach(pinRoll2);

  servoX.write(center);
  servoY.write(center);
  roll1.write(center);
  roll2.write(center);

  pidPitch.SetMode(AUTOMATIC);
  pidRoll.SetMode(AUTOMATIC);
  pidYaw.SetMode(AUTOMATIC);

  pidPitch.SetOutputLimits(-radius, radius);
  pidRoll.SetOutputLimits(-radius, radius);
  pidYaw.SetOutputLimits(-rollAmplitude, rollAmplitude);

  pidPitch.SetTunings(Kp, Ki, Kd);
  pidRoll.SetTunings(Kp, Ki, Kd);
  pidYaw.SetTunings(Kp, Ki, Kd);

  delay(2000);
  startTime = millis();
  lastUpdate = millis();
}

void loop() {
  unsigned long now = millis();
  float elapsed = (now - startTime) / 1000.0;

  if (!demoDone) {
    runDemoMotion(elapsed);
    if (elapsed > 10) {
      demoDone = true;
```

```
      digitalWrite(ledPin, LOW);
      servoX.write(center);
      servoY.write(center);
      roll1.write(center);
      roll2.write(center);
      startTime = millis();
      lastUpdate = millis();
    }
    delay(10);
    return;
  }

  if (!centerPauseDone) {
    if (elapsed > 2) {
      centerPauseDone = true;
      calibrateGyro();
    }
    return;
  }

  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  float dt = (millis() - lastUpdate) / 1000.0;
  lastUpdate = millis();

  float accelPitch = atan2(-a.acceleration.x, sqrt(a.acceleration.y *
a.acceleration.y + a.acceleration.z * a.acceleration.z)) * 180.0 / PI;
  float accelRoll = atan2(a.acceleration.y, a.acceleration.z) * 180.0 / PI;
  float gyroPitchRate = g.gyro.y * 57.2958;
  float gyroRollRate  = g.gyro.x * 57.2958;
  float gyroZrate     = (g.gyro.z - gyroZoffset) * 57.2958;
  if (abs(gyroZrate) < 0.8) gyroZrate = 0;

  compPitch = alpha * (compPitch + gyroPitchRate * dt) + (1 - alpha) *
accelPitch;
  compRoll  = alpha * (compRoll  + gyroRollRate  * dt) + (1 - alpha) * accelRoll;

  compPitch = constrain(compPitch, -tiltRange, tiltRange);
  compRoll  = constrain(compRoll, -tiltRange, tiltRange);

  // PID inputs and compute
  pitchInput = compPitch;
  rollInput = compRoll;
  yawInput = gyroZrate;
```

```
  pidPitch.Compute();
  pidRoll.Compute();
  pidYaw.Compute();

  int servoPitch = center - (int)pitchOutput;
  int servoRoll  = center - (int)rollOutput;
  int servoYaw   = center - (int)yawOutput;

  if (servoPitch != lastX) {
    servoX.write(servoPitch);
    lastX = servoPitch;
  }
  if (servoRoll != lastY) {
    servoY.write(servoRoll);
    lastY = servoRoll;
  }
  if (servoYaw != lastRoll) {
    roll1.write(servoYaw);
    roll2.write(servoYaw);
    lastRoll = servoYaw;
  }

  Serial.print("Pitch Angle: "); Serial.print(compPitch);
  Serial.print(" | Roll Angle: "); Serial.print(compRoll);
  Serial.print(" | Yaw (Gyro Z): "); Serial.println(gyroZrate);

  delay(5);
}
```


Source: NASA
STS-124 // May 31, 2008
EVERYDAY
ASTRONAUT