## APPENDIX A

```
#include "Arduino.h"
#include "MPU6050.h"
#include "Wire.h"
#include "I2Cdev.h"
#include <Servo.h>
int16_t mpu6050Ax, mpu6050Ay, mpu6050Az;
int16_t mpu6050Gx, mpu6050Gy, mpu6050Gz;
MPU6050 mpu6050;
Servo myservo;
const int servoPin = 9;
const int16_t fallThreshold = -8000;
const int16_t riseThreshold = 8000;
bool isFalling = false;
bool isRising = false;
void setup() {
  Serial.begin(9600);
  while (!Serial) ; // Wait for serial port to connect
  Serial.println("Start");
  Wire.begin();
  mpu6050.initialize();
  myservo.attach(servoPin);
  myservo.write(90);
  delay(1500); // Wait for servo movement
}
```

```
void loop() {
  readMPU6050Data();
  detectFallAndRise();
  delay(100); // Adjust delay as needed
}



void readMPU6050Data() {
  mpu6050.getMotion6(&mpu6050Ax, &mpu6050Ay, &mpu6050Az, &mpu6050Gx,
&mpu6050Gy, &mpu6050Gz);
  double mpu6050Temp = ((double)mpu6050.getTemperature() + 12412.0) / 340.0;
  Serial.print("a/g-\t");
  Serial.print(mpu6050Ax); Serial.print("\t");
  Serial.print(mpu6050Ay); Serial.print("\t");
  Serial.print(mpu6050Az); Serial.print("\t");
  Serial.print(mpu6050Gx); Serial.print("\t");
  Serial.print(mpu6050Gy); Serial.print("\t");
  Serial.print(mpu6050Gz); Serial.print("\t");
  Serial.print(F("Temp- "));
  Serial.println(mpu6050Temp);
}
void detectFallAndRise() {
  if (mpu6050Ay < fallThreshold && !isFalling) {
    Serial.println("rise detected!");
    isFalling = true;
    isRising = false; // Reset rise detection
```

```
      myservo.write(0);
    delay(1500); // Wait for servo movement
  }
  if (mpu6050Ay > riseThreshold && !isRising) {
    Serial.println("Fall detected!");
    isRising = true;
    isFalling = false;
    myservo.write(90);
    delay(1500); // Wait for servo movement
  }
}
```

## APPENDIX B

```
#include <Wire.h>
#include <MPU6050.h>
#include <SoftwareSerial.h>
#include <MS5611.h>
#include <Servo.h>

MPU6050 mpu;
MS5611 barometer;
Servo servoMotor;


SoftwareSerial bluetoothSerial(2, 3); // RX, TX pins for SoftwareSerial

const int threshold = 18000; // Threshold for detecting rise or fall
const int stableThreshold = 14000; // Threshold for detecting stability
const int servoPin = 9; // Pin connected to servo motor
int fallCounter = 0; // Number of fall detection counter. It can be configured so that the parachute
can be opened when a certain number of fall detection signals are obtained.
const int fallCounterThreshold = 1; // Minimum number of fall signal detection

void setup() {
  Wire.begin();
  Serial.begin(9600);
  bluetoothSerial.begin(9600); // Initiate Bluetooth communication
  servoMotor.attach(servoPin); // Attach servo motor
  mpu.initialize();
  barometer.begin(); // Initialize the MS5611 sensor

  if (!barometer.begin()) {
    Serial.println("Could not find a valid MS5611 sensor, check wiring!");
    while (1); // Infinite loop
  }

  if (!mpu.testConnection()) {
    Serial.println("Could not connect to MPU6050, check wiring!");
    while(1); // Infinite loop
  }
  servoMotor.write(0); // Initialize the position of the servo as 0
}

void loop() {

  barometer.read(); // Read sensor data (temperature and pressure)
```

```
  float pressure = barometer.getPressure(); // Read Pressure
  float temperature = barometer.getTemperature(); // Read Temperature

  int16_t ax, ay, az;
  int16_t gx, gy, gz;

  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

  // Calculate total acceleration
  int16_t totalAcceleration = sqrt(pow(ax,2) + pow(ay,2) + pow(az,2));

  // Convert raw acceleration data into g
  float converted_ax = (float) ax / 16384.0;
  float converted_ay = (float) ay / 16384.0;
  float converted_az = (float) az / 16384.0;

  // Convert raw gyroscope data into degrees per second
  float converted_gx = (float) gx / 131.0;
  float converted_gy = (float) gy / 131.0;
  float converted_gz = (float) gz / 131.0;


  if(totalAcceleration > threshold) {

    String output = "Rise detected -> Total Acc:" + String(totalAcceleration) + "-[Accel:" +
String(converted_ax) + "g|" + String(converted_ay) + "g|" + String(converted_az) +"g]-[Gyro: " +
String(converted_gx) + "deg/s|" + String(converted_gy) + "deg/s|" + String(converted_gz)
+"deg/s]-[Pressure: " + String(pressure) + " hPa]-[Temperature: " + String(temperature) +
"Â°C]-[Parachute:Close]";
    sendBluetoothData(output);
    Serial.println(output);
    delay(10); // Delay to avoid multiple consecutive readings

  } else if (threshold > totalAcceleration and totalAcceleration > stableThreshold) {

    String output = "Stable detected -> Total Acc:" + String(totalAcceleration) + "-[Accel:" +
String(converted_ax) + "g|" + String(converted_ay) + "g|" + String(converted_az) +"g]-[Gyro: " +
String(converted_gx) + "deg/s|" + String(converted_gy) + "deg/s|" + String(converted_gz)
+"deg/s]-[Pressure: " + String(pressure) + " hPa]-[Temperature: " + String(temperature) +
"Â°C]-[Parachute:Close]";
    sendBluetoothData(output);
    Serial.println(output);
    delay(10); // Delay to avoid multiple consecutive readings

  } else {
```

```
    String output = "Fall detected -> Total Acc:" + String(totalAcceleration) + "-[Accel:" +
String(converted_ax) + "g|" + String(converted_ay) + "g|" + String(converted_az) +"g]-[Gyro: " +
String(converted_gx) + "deg/s|" + String(converted_gy) + "deg/s|" + String(converted_gz)
+"deg/s]-[Pressure: " + String(pressure) + " hPa]-[Temperature: " + String(temperature) +
"Â°C]";
    sendBluetoothData(output);
    Serial.print(output);
    fallCounter += 1;
    Serial.print("Fall Counter: " + String(fallCounter));
    //operateServo();


    delay(10); // Delay to avoid multiple consecutive readings
  }

  if (fallCounter >= fallCounterThreshold) {

    operateServo();

    delay(10);
    fallCounter = 0; // Reset the fallCounter to prevent opening the servo continuously
  }
  delay(100); // Adjust delay to observe the outputs clearly
}


void operateServo() {
  // Move servo motor to a specific angle
  servoMotor.write(90); // Adjust the angle as per your requirement
  String servoStatus = "Parachute is opened";
  Serial.println(servoStatus);
  sendBluetoothData(servoStatus);
  //servoMotor.write(0);
  delay(1500);

}

void sendBluetoothData(String data) {
  bluetoothSerial.println(data);
}
```

## APPENDIX C

```
#include <Wire.h>
#include <MPU6050.h>
#include <Servo.h>
#include <SoftwareSerial.h>

MPU6050 mpu6050;
Servo myServo;
SoftwareSerial bluetoothSerial(2, 3); // RX, TX

const int LOOP_DELAY = 10;
const int PITCH_FALL_ANGLE_TRIGGER = 15;
const int ROLL_FALL_ANGLE_TRIGGER = 15;
const int PITCH_FALLEN_ANGLE = 60;
const int ROLL_FALLEN_ANGLE = 60;
const int MAX_FALL_COUNT = 5;
const int MAX_LONG_FALL_DURATION = 3000;
const int TIMER_INTERVAL = 100;
const int SERVO_PIN = 9;

int timer = 0;
int pitch = 0;
int roll = 0;
int prevPitch = 0;
int prevRoll = 0;
int fallCount = 0;
int longFallCount = 0;
bool fallReported = false;
unsigned long fallStartTime = 0;

void getPitchAndRoll() {
  int16_t accX, accY, accZ, gyroX, gyroY, gyroZ;
  mpu6050.getMotion6(&accX, &accY, &accZ, &gyroX, &gyroY, &gyroZ);

  pitch = -(atan2(accX, sqrt(accY * accY + accZ * accZ)) * 180.0) / M_PI;
  roll = -(atan2(-accY, accZ) * 180.0) / M_PI;
}

void setup() {
  Serial.begin(9600);
  bluetoothSerial.begin(9600); // Bluetooth serial başlatılıyor
  Wire.begin();
  mpu6050.initialize();
  myServo.attach(SERVO_PIN);
```

```
  fallCount = 0;
  longFallCount = 0;
  fallReported = false;
  if (!mpu6050.testConnection()) {
    Serial.println("Could not connect to MPU6050, check wiring!");
    while (1);
  }
}

void loop() {
  unsigned long currentTime = millis();

  getPitchAndRoll();

  String output=("Pitch = "+ String(pitch)+ "      Roll = " + String(roll));
  Serial.println(output);
  sendBluetoothData(output);
  int pitchDelta = abs(pitch - prevPitch);
  int rollDelta = abs(roll - prevRoll);

  if (pitchDelta > PITCH_FALL_ANGLE_TRIGGER || rollDelta >
ROLL_FALL_ANGLE_TRIGGER) {
    fallCount++;

    if (fallCount >= MAX_FALL_COUNT && pitch<0) {
      if (!fallReported) {
        fallReported = true;
        fallStartTime = currentTime;

      }
    }
  } else {
    fallCount = 0;
    fallReported = false;
  }

  if (fallReported) {
    if (currentTime - fallStartTime >= MAX_LONG_FALL_DURATION) {
      sendBluetoothData("Uzun Süreli Düşme Algılandı!");
      Serial.println("Uzun Süreli Düşme Algılandı!");
      rotateServoForDuration(180, 1000);
      exit(0);
    }
  }

  if (currentTime % TIMER_INTERVAL == 0) {
    prevPitch = pitch;
    prevRoll = roll;
```

```
  }

  delay(LOOP_DELAY);
}

void rotateServoForDuration(int angle, unsigned long duration) {
  unsigned long startTime = millis();
  while (millis() - startTime <= duration) {
    myServo.write(angle);
    delay(20);
  }
}

void sendBluetoothData(String data) {
  bluetoothSerial.println(data);
}
```