```
#include <Wire.h>
#include <SPI.h>
#include <SparkFunLSM9DS1.h>

LSM9DS1 imu;

// SDO_XM and SDO_G are both pulled high, so our addresses are:
#define LSM9DS1_M 0x1E // Would be 0x1C if SDO_M is LOW
#define LSM9DS1_AG  0x6B // Would be 0x6A if SDO_AG is LOW

#define PRINT_CALCULATED
//#define PRINT_RAW

#define DECLINATION -8.58 // Declination (degrees) in Boulder, CO.

uint32_t lastWrite = 0;
uint32_t lastSensorCheck = 0;

boolean cycling = false;

int currentPrintSpeed = 500;

int BASE_PRINT_SPEED = 500;
int CYCLING_PRINT_SPEED = 100;
int SENSOR_CHECK_SPEED = 100;

// 2m = 120s = 120,000ms
uint32_t TIME_TO_BEGIN_IN_MILLIS = 120000;
// 3m = 180s = 180,000ms
uint32_t TIME_TO_STOP_IN_MILLIS = 180000;

void setup()
{
  Serial.begin(115200);

  imu.settings.device.commInterface = IMU_MODE_I2C;
  imu.settings.device.mAddress = LSM9DS1_M;
  imu.settings.device.agAddress = LSM9DS1_AG;

  if (!imu.begin())
  {
        Serial.println("Failed to communicate with LSM9DS1.");
        Serial.println("Double-check wiring.");
```

```
        Serial.println("Default settings in this sketch will " \
                "work for an out of the box LSM9DS1 " \
                "Breakout, but may need to be modified " \
                "if the board jumpers are.");
        while (1)
        ;
  }
  pinMode(3, OUTPUT); //CW Thruster
  pinMode(4, OUTPUT); //CCW Thruster
  Serial.println("Time, GyroX, GyroY, GyroZ, AccelX, AccelY, AccelZ, Pitch, Roll, Heading");
//header for data
}

void loop()
{
  readSensor();
  checkWrite();
  checkCycleChange();
  setThrusters();
}

void readSensor() {
  if ((lastSensorCheck + SENSOR_CHECK_SPEED) < millis()) {
        lastSensorCheck = millis();
        if ( imu.gyroAvailable() )
        {
        imu.readGyro();
        }
        if ( imu.accelAvailable() )
        {
        imu.readAccel();
        }
        if ( imu.magAvailable() )
        {
        imu.readMag();
        }
  }
}

void checkWrite() {
  if ((lastWrite + currentPrintSpeed) < millis()) {
        lastWrite = millis();
        Serial.print(lastWrite);
```

```
        Serial.print(", ");

        printGyro();  // Print gx, gy, gz"
        printAccel(); // Print ax, ay, az"
        printAttitude(imu.ax, imu.ay, imu.az,
                -imu.my, -imu.mx, imu.mz);
  }
}

void checkCycleChange() {
  if (isTimeToStabilize() && !cycling)
        {
        cycling = true;

        Serial.println("\n*** Stabilization Cycle Started ***\n");
        Serial.println("Time, GyroX, GyroY, GyroZ, AccelX, AccelY, AccelZ, Pitch, Roll,
Heading"); //header for data
        }
        if (!isTimeToStabilize() && cycling)
        {
        cycling = false;
        Serial.println("\n*** Stabilization Cycle Ended ***\n");
        Serial.println("Time, GyroX, GyroY, GyroZ, AccelX, AccelY, AccelZ, Pitch, Roll,
Heading"); //header for data
        }
}

void setThrusters() {
  if (cycling){
        double g = imu.calcGyro(imu.gz); //save current gyro reading to g (rad/s); + is CCW, - is
CW
        if (g > 10) //number changes how sensitive the attitude control system is to the gz
readings
        digitalWrite(3,HIGH); //rotational motion is CCW, fire thrusters CW
        else if (g < -10)
        digitalWrite(4,HIGH); //rotational motion is CW, fire thrusters CCW
        else //if gz reading is within acceptable range, turn off appropriate valve
        {
        if (digitalRead(3) == HIGH)
        digitalWrite(3,LOW);
        if (digitalRead(4) == HIGH)
        digitalWrite(4,LOW);
        }
```

```
        }
}

boolean isTimeToStabilize()
{
  if (millis() > TIME_TO_BEGIN_IN_MILLIS && millis() < TIME_TO_STOP_IN_MILLIS)
  {
        currentPrintSpeed = CYCLING_PRINT_SPEED;
        return true;
  }
  else
  {
        currentPrintSpeed = BASE_PRINT_SPEED;
        return false;
  }
}

void printGyro()
{
#ifdef PRINT_CALCULATED
  Serial.print(imu.calcGyro(imu.gx), 2);
  Serial.print(", ");
  Serial.print(imu.calcGyro(imu.gy), 2);
  Serial.print(", ");
  Serial.print(imu.calcGyro(imu.gz), 2);
  Serial.print(", ");
  //Serial.println(" deg/s");
#elif defined PRINT_RAW
  Serial.print(imu.gx);
  Serial.print(", ");
  Serial.print(imu.gy);
  Serial.print(", ");
  Serial.print(imu.gz);
  Serial.print(", ");
#endif
}

void printAccel()
{
#ifdef PRINT_CALCULATED
  Serial.print(imu.calcAccel(imu.ax), 2);
  Serial.print(", ");
  Serial.print(imu.calcAccel(imu.ay), 2);
```

```
  Serial.print(", ");
  Serial.print(imu.calcAccel(imu.az), 2);
  Serial.print(", ");
  //Serial.println(" g");
#elif defined PRINT_RAW
  Serial.print(imu.ax);
  Serial.print(", ");
  Serial.print(imu.ay);
  Serial.print(", ");
  Serial.print(imu.az);
  Serial.print(", ");
#endif

}

void printMag()
{
#ifdef PRINT_CALCULATED
  Serial.print(imu.calcMag(imu.mx), 2);
  Serial.print(", ");
  Serial.print(imu.calcMag(imu.my), 2);
  Serial.print(", ");
  Serial.println(imu.calcMag(imu.mz), 2);
  //Serial.println(" gauss");
#elif defined PRINT_RAW
  Serial.print(imu.mx);
  Serial.print(", ");
  Serial.print(imu.my);
  Serial.print(", ");
  Serial.println(imu.mz);
  //Serial.print(", ");
#endif
}

void printAttitude(float ax, float ay, float az, float mx, float my, float mz)
{
  float roll = atan2(ay, az);
  float pitch = atan2(-ax, sqrt(ay * ay + az * az));

  float heading;
  if (my == 0)
	heading = (mx < 0) ? PI : 0;
  else
```

```
      heading = atan2(mx, my);

  heading -= DECLINATION * PI / 180;

  if (heading > PI) heading -= (2 * PI);
  else if (heading < -PI) heading += (2 * PI);
  else if (heading < 0) heading += 2 * PI;

  // Convert everything from radians to degrees:
  heading *= 180.0 / PI;
  pitch *= 180.0 / PI;
  roll  *= 180.0 / PI;

  Serial.print(pitch, 2);
  Serial.print(", ");
  Serial.print(roll, 2);
  Serial.print(", ");
  Serial.println(heading, 2);
}
```