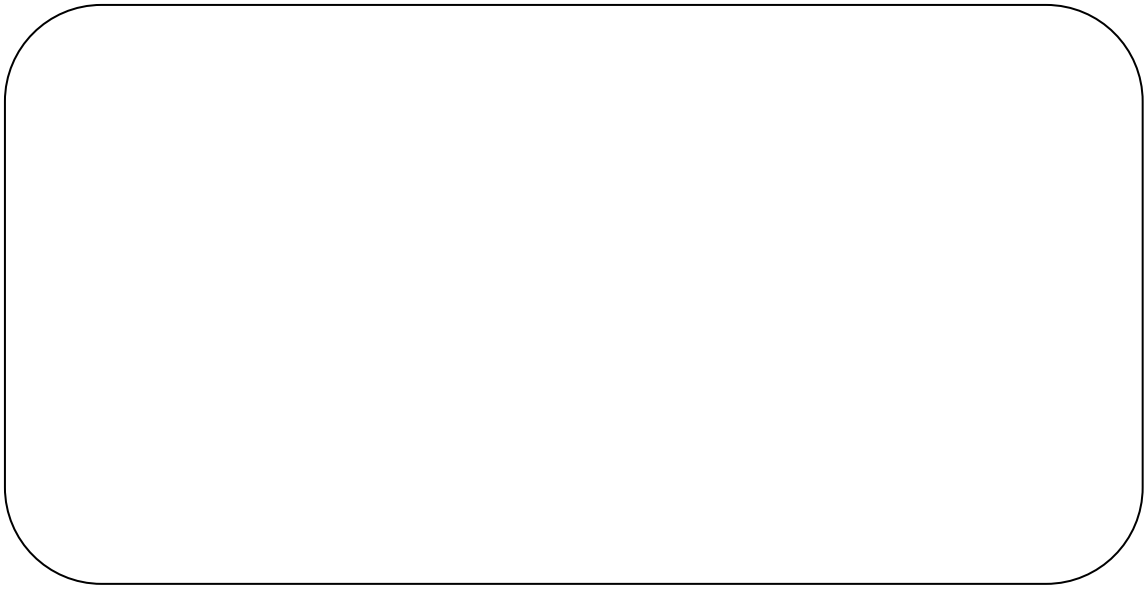
Student Name: Caleb Oviedo-Blomquist

Project Title: Canard-Based Rocket Roll Control System

Project Summary Sheet:



Description:

This system was designed as a proof of concept for a purely roll based control system for a high-powered rocket. There are issues with other control methods, such as have a stepper/servo on each canard, as they could be used for 2-axis control of a rocket, which is against ITAR regulations. The benefit of this solution is that all the canards are actuated simultaneously with a single stepper, and can also be used as an air-braking system without introducing a pitching moment. I simplified this system to only account for the angular velocity of the sensor using an open loop control scheme. A real canard roll control system would be a closed loop control scheme that measures the linear velocity of the rocket as well as the change in angular velocity to calculate the ideal angle of attack for the canards to produce a stable flight.

Inputs:

Adafruit 9-DOF Orientation IMU Fusion Breakout - BNO085 (BNO080) - STEMMA QT / Qwiic: This one Inertial Measurement Unit (IMU) houses both the sensors I am using for this project: A gyroscope (Gyro) to measure the angular velocity of the rocket, and an accelerometer to measure the gravity vector of the rocket. Both inputs are sent via the I^2C protocol and parsed with an Arduino package.

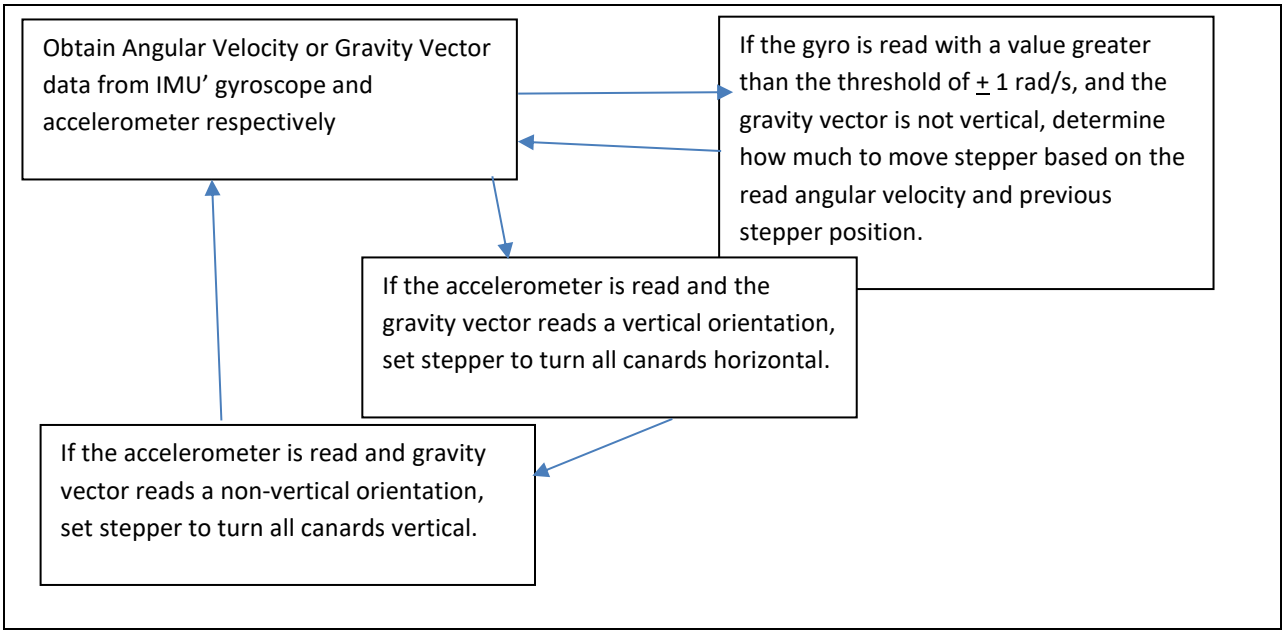
Outputs:

Canard Actuator: It is a stepper motor used to control the position of the canards to allow for roll control of a rocket in flight. It takes in 12 V DC from a stepper controller to allow for precise positioning control. The controller is controlled by an Arduino library that sends the data to the controller to move the stepper in the desired way

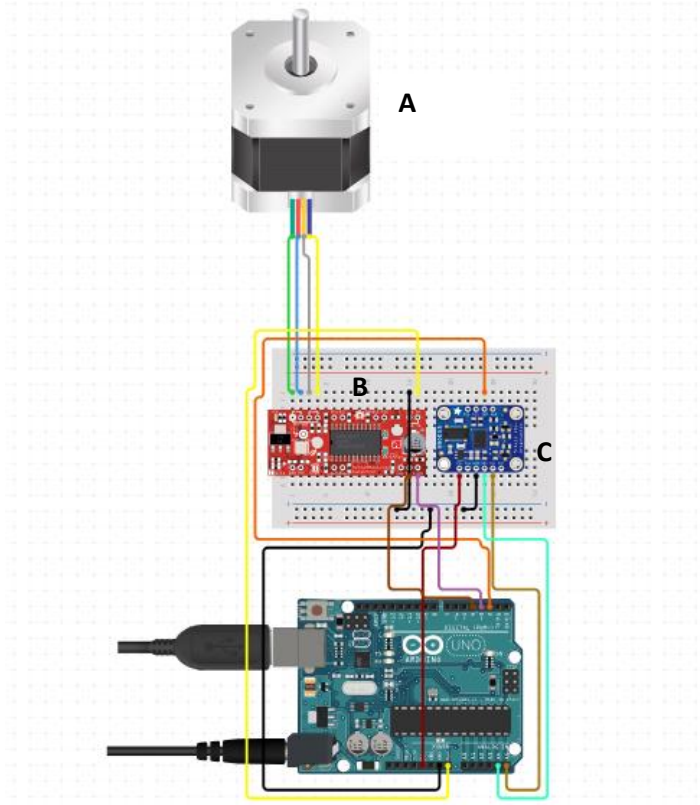
LED Matrix: It is the onboard LED matrix of the Arduino that turns on when the gravity vector of the accelerometer is pointing in the upwards direction. It is controlled by an Arduino library that writes the values of a given frame to the matrix when a function is called.

Functions:

1. Read a sensor input (gyro or accelerometer)
2. If the gyro is read, check if the sensor is vertical, if it is skip to next reading
    a. If the sensor is not vertical and it detects a value greater than the threshold of + 1 rad/s, determine how much to move stepper based on the read angular velocity and previous stepper position.
3. If the accelerometer/gravity vector is read, check to see if the gravity vector is vertical or not
    a. If the vector is read as vertical but the variable to keep the check through each loop is not set, change the variable and move the canards to be horizontal.
    b. If the vector is read as not vertical but the variable to keep the check through each loop is set, change the variable and move the canards to be vertical.
4. Repeat



Design:



A) 12 V Stepper Motor Connected to a gearing system for canard control
B) Stepper driver (One depicted not one to be used)
C) 9-DOF Inertial Measurement Unit

Measurements and Calculations:

No Tabulated Measurements. Only measured angular velocity and the gravity vector to inform what the stepper motor should do.

Results:

No results. The stepper worked as intended.

Discussion:.

The stepper works as I had envisioned, where based on the angular velocity of the sensor the stepper moves the canards to a corresponding position and returned to vertical when no velocity was detected. It also moved the stepper to a position where the canards were horizontal when the sensor was laid flat.

Improvements and Extensions:

The main area of improvement would be the design of the canards and housing themselves as they were not strong enough to handle the quick impulses from the stepper. This would require remodeling the retention system for the canards as well as a better mounting system for the stepper motor. For the control electronics there would have to be a closed loop system that would take into account the linear velocity of the rocket as well as the angular acceleration to adjust the canards to positions that would be

able to keep the rocket locked to a particular roll angle. This along with CFD of the canards themselves to understand the maximum angle of attack they could have before stalling.

References:

Example code from the library,Adafruit_BNO08x.h, used to parse data from the IMU.

Appendix:

```
#include <Arduino.h>
#include <Adafruit_BNO08x.h>
#include "SparkFun_ProDriver_TC78H670FTG_Arduino_Library.h"
#include "Arduino_LED_Matrix.h"

// For SPI mode, we need a CS pin
#define BNO08X_CS 10
#define BNO08X_INT 9

// For SPI mode, we also need a RESET
//#define BNO08X_RESET 5
// but not for I2C or UART
#define BNO08X_RESET -1

Adafruit_BNO08x bno08x(BNO08X_RESET);

sh2_SensorValue_t sensorValue;

// Stepper declaration
PRODRIVER Stepper;

// Variable initiation
int Gravity = 1; // Make sure stepper doesn't move on startup
int Step = 0; // Tracks steps from a vertical canard position
bool Direction = 0; // Direction stepper moves
int Steplimit = 22; // Max step limit for canard control, out of 34

// LED Matrix definitions
ArduinoLEDMatrix matrix;
// On Frame
unsigned long on[] = {
  0xFFFFFFFF,
  0xFFFFFFFF,
  0xFFFFFFFF
};

// Off Frame
```

```
unsigned long off[] = {
  0x0000000,
  0x0000000,
  0x0000000
};

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

  // IMU startup
  Serial.println("Adafruit BNO08x test!");

  // Try to initialize!
  if (!bno08x.begin_I2C()) {
    // if (!bno08x.begin_UART(&Serial1)) {  // Requires a device with > 300 byte
    // UART buffer! if (!bno08x.begin_SPI(BNO08X_CS, BNO08X_INT)) {
    Serial.println("Failed to find BNO08x chip");
    while (1) {
      delay(10);
    }
  }
  Wire.setClock(400000);
  Serial.println("BNO08x Found!");

  // Set what reports come from the IMU
  setReports();

  // Motor Setup
  Stepper.begin();

  // LED matrix Setup
  matrix.begin();
  matrix.loadFrame(off);
}

// IMU Sensor events to poll for, commented if statements are for other events I
didn't end up needing
void setReports(void) {
  Serial.println("Setting desired reports");
  if (!bno08x.enableReport(SH2_GYROSCOPE_CALIBRATED, 10000)) { // 10000 is the
time in milliseconds for this event to send a new reading
    Serial.println("Could not enable gyroscope");
  }
```

```
  if (!bno08x.enableReport(SH2_GRAVITY, 20000)) { // 20000 is the time in
milliseconds for this event to send a new reading, It's double the gyro because
it was overcrowding the report buffer and the gyro couldn't be read
    Serial.println("Could not enable gravity vector");
  }
}

void loop() {
  delay(10);
  // Check if the sensor was reset and reset the reports
  if (bno08x.wasReset()) {
    Serial.print("sensor was reset ");
    setReports();
  }

  // Check if the sensor reported an event
  if (!bno08x.getSensorEvent(&sensorValue)) {
    return;
  }
    // Read what event was triggered and then call the corresponding block of
code
    switch (sensorValue.sensorId) {

      case SH2_GYROSCOPE_CALIBRATED:
        // Prints out relevant variables
        Serial.println("Gyro");
        Serial.print("Gravity: "); Serial.println(Gravity);
        Serial.print("Step: "); Serial.println(Step);
        Serial.print("Gyro - x: ");
        Serial.println(sensorValue.un.gyroscope.x);
        // Checks if the rocket is not vertical and if the gyro has a large enoug
reading to warrant a canard movement
        if ((Gravity == 0) && (sensorValue.un.gyroscope.x > 1)) { // Change gyro
axis and direction of rotation depending on final orientation of the IMU
          Direction = 0;
          int Step2move2 = map(sensorValue.un.gyroscope.x, 1, 20, 1, Steplimit);
// Maps the output of the gyro to the number of steps within a range
          // The steps are mapped to a positive and negative limit from a
vertical canard position, which is where the next if statements determine the
direction of how the motor should turn
          Serial.print("Step 2 move 2: "); Serial.print(Step2move2);
Serial.println();
          if (Step2move2 > Step) { // If the step we need to go to
            Stepper.step(abs(Step2move2-Step), Direction, 1);
```

```
            Step = Step2move2; // Sets the Step to where the canard is after the
movement
          }
          else if (Step2move2 < Step) {
            Stepper.step(abs(Step2move2-Step), !Direction, 1);
            Step = Step2move2;
          }
          else if (Step2move2 == Step) {}
        }

        else if ((Gravity == 0) && (sensorValue.un.gyroscope.x < -1)) {
          Direction = 1;
          int Step2move2 = map(sensorValue.un.gyroscope.x, -1, -20, -1, -
Steplimit);
          Serial.print("Step 2 move 2: "); Serial.print(Step2move2);
Serial.println();
          if (Step2move2 < Step) {
            Stepper.step(abs(Step2move2-Step), Direction, 1);
            Step = Step2move2;
          }
          else if (Step2move2 > Step) {
            Stepper.step(abs(Step2move2-Step), !Direction, 1);
            Step = Step2move2;
          }
          else if (Step2move2 == Step) {}
        }
        break;
      case SH2_GRAVITY:
        Serial.println("Gravity");
        Serial.print("Gravity: "); Serial.println(Gravity);
        Serial.print("Gravity - z: ");
        Serial.println(sensorValue.un.gravity.z);

        // Checks if the gravity vector is within range for it to be considered
upright and the gravity variable is not triggered
        if (((sensorValue.un.gravity.z > 9) || (sensorValue.un.gravity.z < -9))
&& (Gravity == 0)) { // Change gravity axis depending on final orientation of the
IMU
          // Sets the gravity variable to be triggered
          Gravity = 1;
          Serial.print("Step G->1: "); Serial.println(Step);
          // Steps the required amount to make the canards lay flat
          Stepper.step(abs(34-Step), Direction, 2);
          // Turns on the LED matrix
          matrix.loadFrame(on);
```

```
          // Sets the current step position to be flat
          Step = 34;
        }
        // Checks if we come out of vertical and the gravity conditions is not
turned off
        else if (((sensorValue.un.gravity.z < 9) && (sensorValue.un.gravity.z > -
9)) && (Gravity == 1)) {
          // Sets the gravity condition to be off and our step position to be
vertical
          Gravity = 0;
          Step = 0;
          Serial.print("Step G->0: "); Serial.println(Step);
          // Moves the required amount to be in a vertical position and turns the
LED matrix off
          Stepper.step(34, !Direction, 2);
          matrix.loadFrame(off);
        }
        break;
    }


}
```