P4 Deliverable 3: Manual

Overview: Our device is an actuated platform for enabling the directional control of a camera via remote control.

Function: A wireless remote-control system provides actuation signals to a microcontroller using either conventional buttons/joystick or an inertial measurement unit to provide angular input. (Here, we have chosen to test the feasibility of the latter.)

The remote control utilizes WLAN to transmit the desired angular motion to the actuated platform, which then compares the signals to its current state. Microcontrollers within the remote and the platform use Madgwick filtering to improve the precision of internal state estimation to reduce influence of sensor noise.

The platform's microcontroller computes required actuation signals via angular velocity feedback and uses a DRV 8833 motor driver to actuate the elevation and traverse motors. Power is supplied by 3.7V DC for the microcontrollers, and 6V DC for the motors.

Plans to include autonomous target tracking using inertial estimation were curtailed by excessive sensor noise. Future plans are to investigate use of GPS, RF direction-finding, or computer vision.

Mechanical Components: We show here the mechanical assembly of the actuated platform. Please refer to Appendix E for pictures of the remote control.

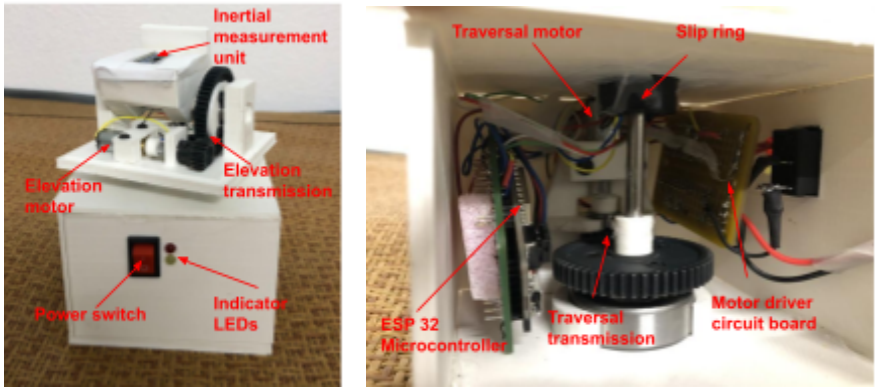

Figure MC1: Images showing the assembled device (left) and the internal components (right).

Load Specifications and Potential Future Improvements:

We approximate a maximum traversal speed requirement for the platform, assuming a 10ft distance to a moving target at 12 mph speed and 30 degrees angle:

$$\frac{dx}{dt} = \frac{d(5tan(\theta))}{dt} = 10\ sec^2(\theta) * \frac{d\theta}{dt} \rightarrow \frac{d\theta}{dt} = \frac{17.6ft/s}{10ft * sec^2(30)} = 1.32\ rad/s$$

The gear reduction of the gearset is 3.37:1, so the motor will need to be able to run at 43 rpm. The moment of inertia of the rotating assembly is at most $645\ kg\ mm^2$. From the datasheet, the traversal motor can output $24.71\ kg\ mm$ of torque at 43 rpm, and higher torque under this value, indicating that the motor is fully capable of accelerating to this speed. Practical usage of the device confirms this finding.

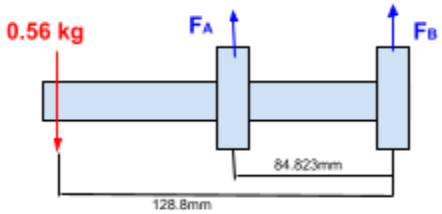

Figure LS1: Figure for the traversal bearing load

$0.56\,kg\ *\ (121.3\ -\ 84.823)mm\ =\ F_B\ *\ 84.823mm$

$F_B\ =\ 0.241\,kg\ =\ 2.36N,\ F_A\ =\ 0.56\,kg\ -\ 0.241\,kg\ =\ 0.3192\,kg\ =\ 3.13N$

Calculations assume that the entire assembly is on its side. Both the bearing loads and the rpm are well within the bearing's maximum specs. The 1st bearing is a slip-ring that acts as a bearing. The radial loads are within spec. The lower bearing handles the entire thrust load of the rotating assembly, 0.56 kgf and consists of 2 bearings, close enough together to count as one bearing.

When the motor is running at maximum power, the small gear will impart 25 kg * mm / 7.6mm = 3.3kgf to the bearing, with 3.091 kgf vertical and 1.125 kgf horizontal due to the 20 degree pitch angle. Due to the relatively low load and the fact that the bearing is about 1/5th the length of the shaft, we determined that one bearing was enough for the motor transmissions.

The elevation portion was shown to have a rotating mass of 0.33422kg , 52.54mm from the center of rotation. In the worst case scenario, this will require 17.56 kg*mm to move. 84.25 kg*mm torque will be applied to the turret, enough to elevate it.

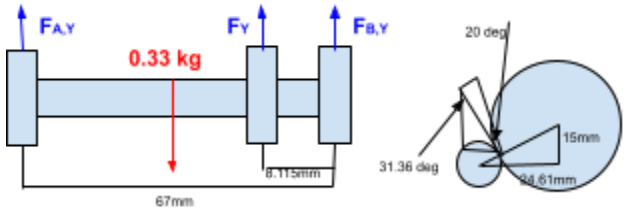

Figure LS2: Figures for the Turret Elevation gear load in the y-direction (left) and Elevation gear side view (right).

$F\ =\ \tau\ /(\ D_{small}/2)\ =\ 3.3kgf$

$F_x\ =\ F\ *\ (-\ cos(20)\ *\ sin(31.363)\ +\ sin(20)\ *\ cos(31.363))\ =-\ 0.65kgf$

$F_y\ =\ F\ *\ (cos(20)\ *\ cos(31.363)\ +\ sin(20)\ *\ sin(31.363))\ =\ 3.225kgf$

$F_{B,Y}\ *-\ 67mm\ =\ -\ 0.33kg\ *\ 33.5mm\ +\ 3.225kg\ *\ 58.885mm \rightarrow F_{B,Y}\ =\ -\ 2.7kgf$

$F_{A,Y}\ +\ F_{B,Y}\ +\ F_{gear,y}\ -\ M\ =\ 0$

$F_{A,Y}\ =\ M\ -\ F_{gear,y}\ -\ F_{B,Y}\ =-\ 0.195kg$

$F_{B,X}\ *\ 67mm\ =\ 0.65kgf\ *\ 58.885mm$

$F_{B,X}\ =\ 0.571\,kgf$

$F_{A,X}\ +\ F_{B,X}\ +\ F_{gear,x}\ =\ 0$

$F_{A,X}\ =\ 0.079kgf$

All of the calculated values are well within the bearings' specs.

Planned Improvements for Mechanical Design:

The transmission bearings use a cantilever configuration, which may expose the bearings to unnecessarily high radial load, so a second bearing could be added to the other side of the gear if this is an issue. A future design could also utilize belleville washers to maintain precise shaft alignment.

The 3D printed couplers also feature set screws that are "self-tapped" into the holes, contacting a flat on the shaft and fixing the component to the shaft. This is not optimal for durability, so a future prototype should use a machined aluminum piece with a pre-drilled and tapped hole for a set screw.

The remote control could also be improved, as it is currently far too large and bulky, and difficult to use. A possible solution to this problem would be to utilize a jog wheel and joystick to control the elevation and traversal functions of the turret, integrated into a much smaller controller that could be held out of sight of the camera.

Electronics and Software: The actuated camera platform schematics have been included here. For the schematics of the remote controller, please refer to the Appendix D.

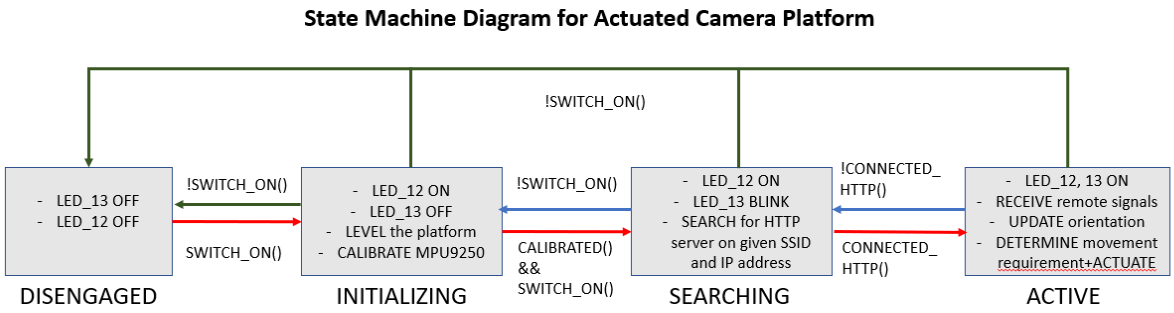

Figure ES1: State machine diagrams for the actuated platform.

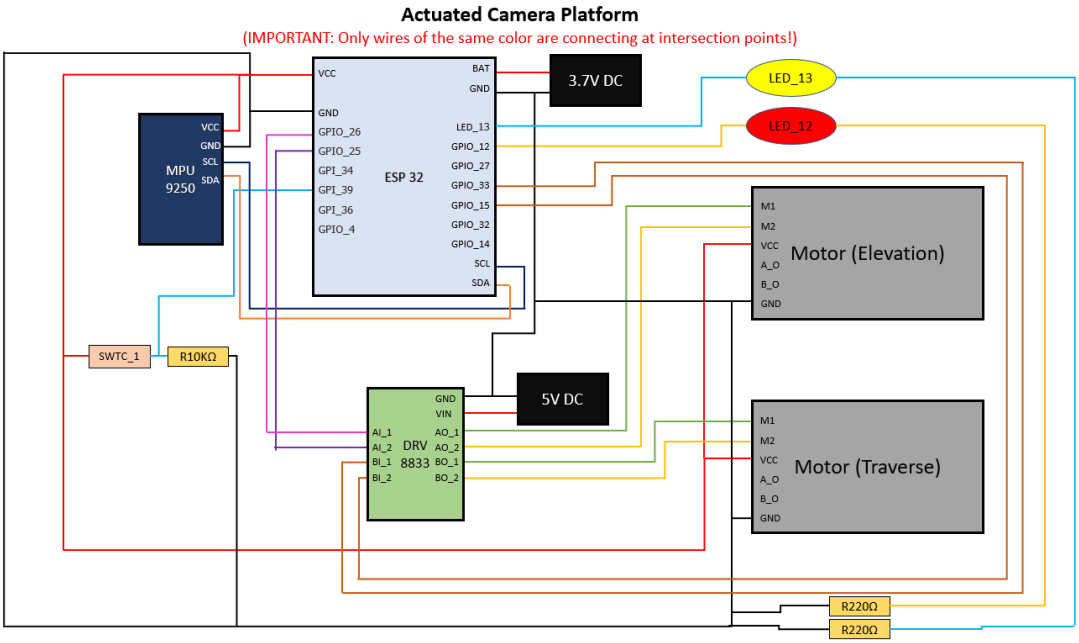

Figure ES2: Wiring schematic for the actuated camera platform.

Reflection: Effective communication allowed us to coordinate well and efficiently complete the project. Given that the group unexpectedly dropped to a size of only 2 people, it meant we needed to adapt our designs to fit our constraints. Despite this unexpected situation, we were able to communicate well and adapt the design to finish the project on time.

We do wish we would have had another member of the team to add more complex functionality to the project. Additionally, a word of advice to future students is to pay attention to wiring in the planning stages; electrical wires do add a degree of complexity to the design, like when we found out we had to buy a slip ring to enable wires to go between the stationary microcontroller and sensors and motor on rotating turret of the platform.

Appendix A - Bill of Materials

Actuated Camera Platform's Purchase Portfolio							**Total (Projected):**	**$ 268.37**
Item Name	**Description**	**Purchase Justification**	**Serial Number / SKU**	**Price (ea.)**	**#**	**Vendor**	**Link to Item**	**Subtotal**
ElectroCookie Mini PCB Prototype Board 6-pack	Solderable Breadboard for DIY Electronics, Compatible for Mini Arduino Soldering Projects, Gold-Plated (6 Pack, Multicolor)	Circuit boards	B081MSKJJX	$ 9.99	1	Amazon (ElectroCookie)	h	$ 9.99
M3 screws, various lengths	DYWISHKEY 360 Pieces M3 x 6mm/8mm/10mm/12mm/16mm/20mm, 10.9 Grade Alloy Steel Hex Button Head Cap Bolts Screws Nuts Kit with Hex Wrench	Fastener	B07VRC7ZL5	$ 9.99	1	Amazon (DYWISHKEY)	h	$ 9.99
F685ZZ Flanged Ball Bearing	5x11x5mm Shielded Chrome Steel Bearings 10pcs	Shaft bearings for reducing radial load on motor components	a19091100ux0578	$ 9.99	1	Amazon (uxCell)	h	$ 9.99
Round Steel Rod, 16mm diameter	16mm HSS Lathe Bar Stock Tool 150mm Long	Shaft for attaching platform to elevation mechanism	a19050800ux0857	$ 15.79	1	Amazon (uxCell)	h	$ 15.79
5mm Stainless Steel Straight Solid Metal Round Rod	356mm length, 2 pieces (712mm of material total)	Will be used to mount gears onto shafts	B07KJ8DGF3	$ 9.99	1	Amazon (Glarks)	h	$ 9.99
Steel gear set (Pinions and spur gear)	54T 0.8 32 Pitch Metal Steel 3956 Spur Gear with 15T/17T/19T Pinions Gear Sets	Gears for elevation mechanism	B088QGFRH7	$ 14.67	1	Amazon (Globact)	h	$ 14.67
6202 Ball Bearings,Mokell 6pcs Roller Bearing	6 pieces (16mmx35mmx11mm) Steel and Double Rubber Sealed Miniature Deep Groove Ball Bearings	Bearings for 16mm shaft used in elevation mechanism	B08KWC7FSY	$ 7.99	1	Amazon (Mlxkell)	h	$ 7.99
4mm to 5mm Flexible Shaft Couplings	WEIJ 5Pcs 4mm to 5mm Shaft Coupling Flexible Coupler Motor Connector Joint L25xD19 Silver	Connecting the motor shaft to the shafts used to drive gears	B07MHKH8LR	$ 13.49	1	Amazon (WEIJ)	h	$ 13.49
Fielect 2 Position ON/Off Switch	ON/Off Black Boat Rocker Switch with 2 Wire AC 250V/6A 125V/10A 1Pcs	On-off switch for camera platform and beacon	B083FR8N11	$ 3.89	1	Amazon (Fielect)	h	$ 3.89
Adafruit (PID 3591) HUZZAH32 – ESP32 Feather Board (pre-soldered)	Micro controller	Micro-controllers for camera platform and for beacon		$ 26.15	2	Amazon (Adafruit)	h	$ 52.30

EEMB 3.7V 1800mAh Lipo Battery	1800mAh Lipo Battery 963450 Li-ion Battery Rechargeable Lithium Polymer ion Battery Pack with JST Connector	Power source for beacon and camera platform	YDL2018032916	$ 12.99	2	Amazon (YDL)	h	$ 25.98
ALMOCN 3PCS GY-521 MPU-6050	MPU6050 Module 3 Axis Accelerometer 6 DOF 6-axis Gyroscope Sensor Module 16 Bit AD Converter Data Output IIC I2C for Arduino	The inertial sensors for the camera platform and beacon	B08GKRFMNH	$ 7.56	1	Amazon (ALMOCN)	h	$ 7.56
Taidacent Hollow Shaft Slip Ring (6 wire Inner hole 5mm X OD 22mm)	Taidacent Through Bore Electrical Slip Ring Rotate Conductive Energization Collector Ring Brushes Hollow Shaft Slip Ring (6 wire Inner hole 5mm X OD 22mm)	To pass electrical wires through rotating platform		$ 19.34	1	Amazon (Taidacent)	h	$ 19.34
Steel gear set (Pinions and spur gear)	54T 0.8 32 Pitch Metal Steel 3956 Spur Gear with 15T/17T/19T Pinions Gear Sets	Gears for traversal mechanism	B088QGFRH7	$ 14.67	1	Amazon (Globact)	h	$ 14.67
GPS NEO-6M	GPS Module GPS NEO-6M (Arduino GPS, Drone Microcontroller GPS Receiver) Compatible with 51 Microcontroller STM32 Arduino UNO R3 with IPEX Antenna High Sensitivity for Navigation Satellite Positioning	Position sensing for beacon	B07P8YMVNT	$ 11.59	1	Amazon (MakerFocus)	h	$ 11.59
50 RPM High Torque Motor	DC 6V Brush 50RPM	Motor for elevation mechanism	TOPINCNbgvec1tak0	$ 8.59	1	Amazon (TOPINCN)	h	$ 8.59
60 RPM High Torque Motor	DC 6V 60 RPM	Motor for traverse mechanism	a12120600ux0191	$ 9.88	1	Amazon (uxcell)	h	$ 9.88
Gikfun J Female & Male Plug Connector Wire Cables for Arduino	Gikfun JST SM 2-Pins 2P Female & Male Plug Connector Wire Cables for Arduino (Pack of 10 Pairs) AE1045	Connectors for 6V battery	B01EJI8TQK	$ 7.68	1	Amazon (Gikfun)	h	$ 7.68
Blomiky 6V 2200mAh Ni-MH 5 AA Rechargeable Battery Pack	Blomiky 6V 2200mAh Ni-MH 5 AA Rechargeable Battery Pack with SM-2P Black 2 Pin Connector Plug and USB Charger Cable for RC Truck Cars Vehicles 6V NiMh	6V Power source for motors	B07PY8R3F4	$ 14.99	1	Amazon (Blomiky)	h	$ 14.99

Appendix B - CAD Images

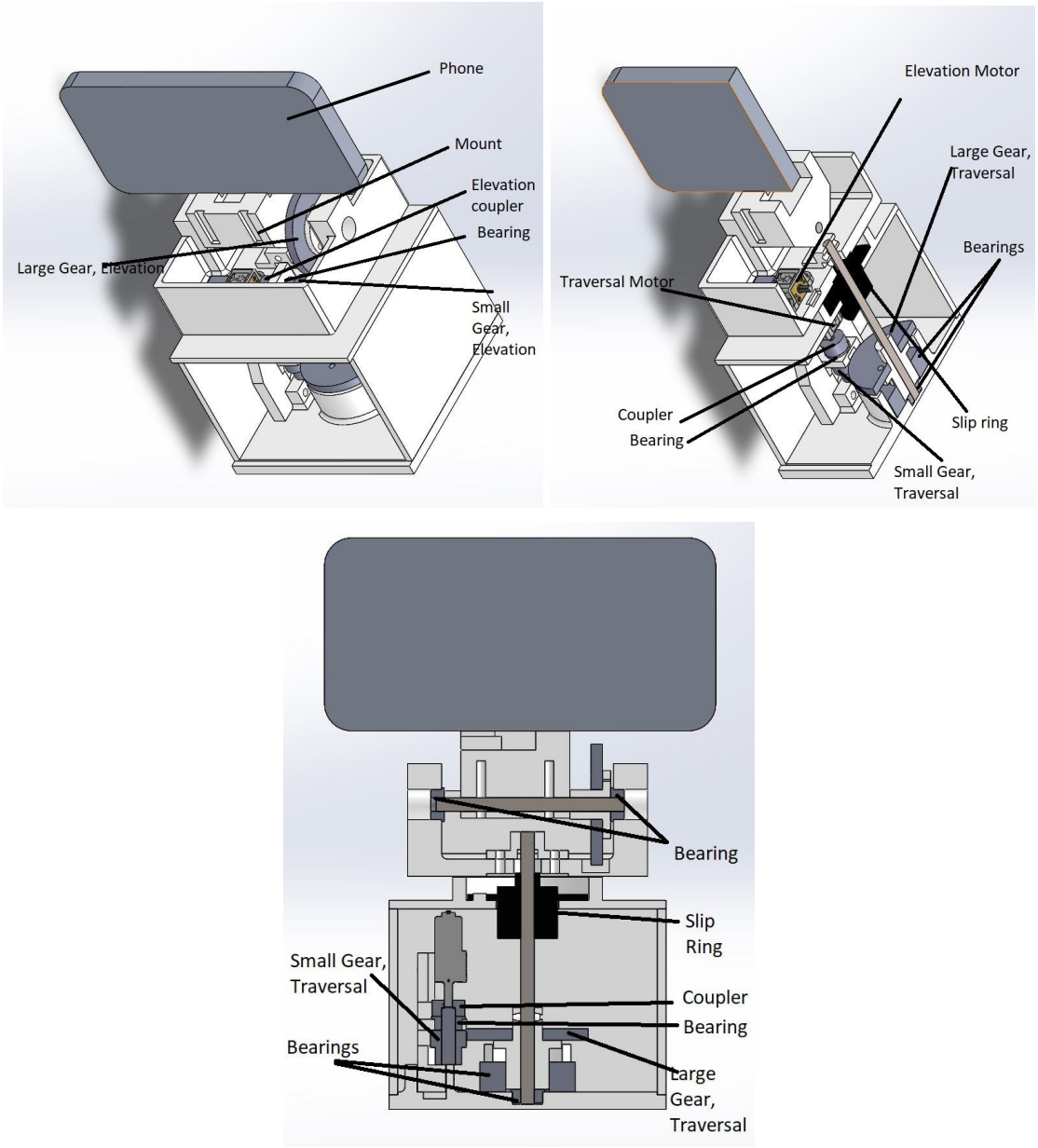

Figure B1: The labeled 3D CAD models of the actuated camera platform, excluding the electronic components.

Appendix C - Software Code

Actuated Platform:

```
#include <MPU9250.h>

#include <WiFi.h>
#include <HTTPClient.h>
#include <Wire.h>

#include "SensorFusion.h"

// HTTP antenna variables --------------------
const char* ssid = "FS-Beacon";
const char* password = "888888888";
const char* statusChannel = "http://192.168.4.1/status";
const char* positionChannel = "http://192.168.4.1/position";
const char* orientationChannel = "http://192.168.4.1/orientation";
bool httpConnecting = false; // Is connection attempt being made?
bool httpConnected = false; // Is connection done?

TaskHandle_t HTTPSENDRECEIVE;

// I/O variables -----------------------------
#define SWTC_1 39  // On-Off switch
#define EMD_1 26  // Traverse motor driver - direction 1
#define EMD_2 25  // Traverse motor driver - direction 2
#define TMD_1 33  // Elevator motor driver - direction 2
#define TMD_2 15  // Elevator motor driver - direction 2
#define LED_13 13  // YELLOW LED
#define LED_12 12  // RED LED

SF fusion;

// Measurement and control variables ---------------------------------
float gx, gy, gz, ax, ay, az, mx, my, mz, temp;
float pitch, roll, yaw; // The pitch, roll, and yaw of the camera platform
float pitch_rate, yaw_rate; // The rates of pitch and yaw (roll should not change for the device)
float deltat;
float pitch_des, yaw_des; // 2 axis camera platform can have pitch and yaw be controlled
float pitch_rate_des, yaw_rate_des; // 2 axis camera platform can have pitch and yaw be controlled

float pitch_rate_error = 0;
float yaw_rate_error = 0;

float p_e = 0;
float Kp_e = 7.0f; // Elevation feedback constant
float Kp_d = 4.0f; // Depression feedback constant
float K_i = 0.0f; // For integral term

float prev_pitch, prev_yaw = 0; // previous pitch and yaw

int state = 0;

// Set up IMU sensor -------------------------------------
MPU9250 IMU(Wire, 0x68);
int status;

// IMU cclibration variables ---------------------------------
volatile bool calibrated = false; // Calibration completed?
volatile bool calibrating = false; // Calibration in process?
int calibrationStage = 0; // 0 = find z-offset, 1 = level z, 2 = find x,y offset, 3 = calibrate gyros, 4
= calibrate mag sensor
int calibrationCounter = 0; // Tracks number of times calibration measurements have been taken

// Platform calibration variables
short directionTowardsLevel = 0; // Indicates the direction of elevation offset from a level surface; -1
for below horizon and 1 for above
float prevZMeas = 0.0f;
float maxGravZ = 0.0f; // The maximum value of gravity detected along z-axis; this is the position where
```

```
the platform is fully level
const float g = 9.81f; // The gravitational acceleration on Earth (we will subtract this from maxGravZ
to find the accelerometer z-offset, which we then use to auto-level the platform)

// Remote state variables ----------------------------
float b_X, b_Y, b_Z = 0; // x,y,z positions (currently unused due to lack of good estimator)
int b_Pitch, b_Roll, b_Yaw = 0; // Angular orientation (degrees)
int b_State = 0; // Current state of the remote

int commaIndex = 0; // Intermediate variable for parsing the HTTP message
String val = ""; // Intermediate variable for parsing the HTTP message

const long interval = 200; // Will check for updates every 1/5 second
unsigned long previousMillis = 0;

// PWM Characteristics -----------------------------
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;

const int ledChannel_3 = 3;
const int ledChannel_4 = 4;

const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE = 150;

// Motor PWM signals ---------------------------
int TP = 0; // Traversal motor PWM signal
int EP = 0; // Elevator motor PWM signal

//Timer interrupt variables ----------------------------
volatile bool interruptCounter0 = false;    // check timer interrupt 0
volatile bool interruptCounter1 = false;     // check timer interrupt 1

volatile bool switch_on = false;

int totalInterrupts0 = 0;   // counts the number of triggering of the alarm
int totalInterrupts1 = 0;   // counts the number of triggering of the alarm

hw_timer_t * timer0 = NULL;
hw_timer_t * timer1 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

//Initialization for interrupts ----------------------------------
void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
  interruptCounter0 = true; // the function to be called when timer interrupt is triggered
  totalInterrupts0++;
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  interruptCounter1 = true; // the function to be called when timer interrupt is triggered
  totalInterrupts1++;
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);

  pinMode(SWTC_1, INPUT);  // configures the specified pin to behave either as an input or an output
  pinMode(LED_12, OUTPUT);
  pinMode(LED_13, OUTPUT);
  pinMode(TMD_1, OUTPUT);
  pinMode(TMD_2, OUTPUT);
  pinMode(EMD_1, OUTPUT);
```

```
  pinMode(EMD_2, OUTPUT);

  // configure LED PWM functionalitites
  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);

  // configure LED PWM functionalitites
  ledcSetup(ledChannel_3, freq, resolution);
  ledcSetup(ledChannel_4, freq, resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(TMD_2, ledChannel_2);
  ledcAttachPin(TMD_1, ledChannel_1);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(EMD_2, ledChannel_4);
  ledcAttachPin(EMD_1, ledChannel_3);

  timer0 = timerBegin(0, 80, true);  // timer 0, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE
-> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer0, &onTime0, true); // edge (not level) triggered
  timerAlarmWrite(timer0, 1000000, true); // 1000000 * 1 us = 0.5 s, autoreload true

  timer1 = timerBegin(1, 80, true);  // timer 1, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE
-> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggered
  timerAlarmWrite(timer1, 20000, true); // 20000 * 1 us = 20 ms, autoreload true

  // at least enable the timer alarms
  timerAlarmEnable(timer0); // enable
  timerAlarmEnable(timer1); // enable

  status = IMU.begin();
  if (status < 0) {
    Serial.println("IMU initialization unsuccessful");
    Serial.println("Check IMU wiring or try cycling power");
    Serial.print("Status: ");
    Serial.println(status);
    state = -1; // If the IMU could not initialize, then go into the error state
    return;
  }

  // setting the accelerometer full scale range to +/-8G
  IMU.setAccelRange(MPU9250::ACCEL_RANGE_2G);
  // setting the gyroscope full scale range to +/-500 deg/s
  IMU.setGyroRange(MPU9250::GYRO_RANGE_250DPS);
}

void loop()
{
  switch (state)
  {
    case 0: // Switch is off (Do nothing at all)
      digitalWrite(LED_12, LOW);
      digitalWrite(LED_13, LOW);
      STOP_ELEVATOR();  // Stop all motors
      STOP_TRAVERSE();
      if (SWITCH_ON())
      {
        state = 1;
      }
      break;
    case 1: // Calibrate our platform
      digitalWrite(LED_12, HIGH);
      digitalWrite(LED_13, LOW);

      //STOP_ELEVATOR();
      //STOP_TRAVERSE();

      POSITIONAL_CALIBRATE(); // Calibrate the IMU and auto-level the platform
```

```
      if (CALIBRATED() && SWITCH_ON())
      {
        state = 2; // If calibration is complete
      }

      if (!SWITCH_ON())
      {
        state = 0;
        digitalWrite(LED_12, LOW);
        digitalWrite(LED_13, LOW);
        STOP_ELEVATOR(); // Stop all motors
        STOP_TRAVERSE();
        ESP.restart(); // Restart the micro controller to clear any variables in RAM
      }
      break;
    case 2: // Try to connect to HTTP channel
      if (HTTPSENDRECEIVE == NULL)
        xTaskCreatePinnedToCore(HTTPCoroutine, "HTTPSENDRECEIVE", 10000, NULL, 0, &HTTPSENDRECEIVE, 0);
// Start WLAN on core 0 of processor, to make sure motor is not blocked if there is connection
interruptions

      STOP_ELEVATOR(); // Stop all motors while connecting
      STOP_TRAVERSE();

      BLINK_LED_1S(LED_13);
      digitalWrite(LED_12, HIGH);
      if (!SWITCH_ON())
      {
        state = 0;
        digitalWrite(LED_12, LOW);
        digitalWrite(LED_13, LOW);
        STOP_ELEVATOR(); // Stop all motors
        STOP_TRAVERSE();
        ESP.restart(); // Restart the microcontroller to clear any variables in RAM
      }
      else if (CONNECTED_HTTP())
      {
        state = 3; // Connected? Go to state 3
      }
      break;

    case 3: // Receiving data (actuating as necessary)
      digitalWrite(LED_12, HIGH);
      digitalWrite(LED_13, HIGH);

      POSITIONAL_UPDATE(); // Update own position from IMU readings

      DETERMINE_MOVEMENT_REQ(); // Calculate required motor PWM

      ACTUATE_TRAVERSE(); // Actuate traverse
      ACTUATE_ELEVATOR(); // Actuate elevator

      if (!CONNECTED_HTTP())
      {
        state = 2;
      }
      if (!SWITCH_ON())
      {
        state = 0;
        digitalWrite(LED_12, LOW);
        digitalWrite(LED_13, LOW);
        STOP_ELEVATOR(); // Stop all motors
        STOP_TRAVERSE();
        ESP.restart(); // Restart the microcontroller to clear any variables in RAM
      }
      break;
    default:
      BLINK_LED_1S(LED_13); // All lights flash when there is an error
      BLINK_LED_1S(LED_12);
```

```
      if (!SWITCH_ON())
      {
        state = 0; // If the switch is off, move to state 0
        digitalWrite(LED_12, LOW);
        digitalWrite(LED_13, LOW);
        STOP_ELEVATOR(); // Stop all motors
        STOP_TRAVERSE();
        ESP.restart(); // Restart the microcontroller to clear any variables in RAM
      }
      break;
  }
}

/*
   Run HTTP protocol on second core, so that internet lag does not interrupt the function of the
actuator
*/
void HTTPCoroutine( void* pvParameters)
{
  for (;;) {
    switch (state)
    {
      case 2:
        HTTP_TRY_CONNECTION();
        break;
      case 3:
        HTTP_RECEIVE_MSG(); // Receive data from the remote
        break;
    }
    delay(25); // Required to prevent this core from overloading the CPU
  }
}

// SERVICES ----------------------------------

// Stop motors
void STOP_ELEVATOR() {
  ledcWrite(ledChannel_4, LOW);
  ledcWrite(ledChannel_3, LOW);
}
void STOP_TRAVERSE() {
  ledcWrite(ledChannel_2, LOW);
  ledcWrite(ledChannel_1, LOW);
}

// Actuate motors
void ACTUATE_ELEVATOR() {
  if (EP > 0)
  {
    ledcWrite(ledChannel_3, LOW);
    ledcWrite(ledChannel_4, abs(EP));
  }
  else if (EP < 0)
  {
    ledcWrite(ledChannel_3, abs(EP));
    ledcWrite(ledChannel_4, LOW);
  }
  else
  {
    STOP_ELEVATOR();
  }
}

void ACTUATE_TRAVERSE() {
  if (TP > 0)
  {
    ledcWrite(ledChannel_1, LOW);
    ledcWrite(ledChannel_2, abs(TP));
  }
  else if (TP < 0)
```

```
  {
    ledcWrite(ledChannel_1, abs(TP));
    ledcWrite(ledChannel_2, LOW);
  }
  else
  {
    STOP_TRAVERSE();
  }
}

/**
   Use MPU data to determine the state of this device
  Store data as global variable
*/
void POSITIONAL_UPDATE()
{
  IMU.readSensor(); // Update IMU

  ax = IMU.getAccelX_mss();
  ay = IMU.getAccelY_mss();
  az = IMU.getAccelZ_mss();
  gx = IMU.getGyroX_rads();
  gy = IMU.getGyroY_rads();
  gz = IMU.getGyroZ_rads();
  mx = IMU.getMagX_uT();
  my = IMU.getMagY_uT();
  mz = IMU.getMagZ_uT();
  temp = IMU.getTemperature_C();

  deltat = fusion.deltatUpdate(); // Update time
  fusion.MadgwickUpdate(gx, gy, gz, ax, ay, az, mx, my, mz, deltat);  //Estimate the orientation of the
platform using the filtering algorithm

// Update roll, pitch, yaw
  roll = fusion.getRoll();
  pitch = fusion.getPitch();
  yaw = fusion.getYaw();

// Estimate pitch and yaw rate
  pitch_rate = (pitch - prev_pitch)/deltat;
  yaw_rate = (yaw - prev_yaw)/deltat;

  prev_pitch = pitch;
  prev_yaw = yaw;
}

// For calibration of the sensor
void POSITIONAL_CALIBRATE()
{
  switch (calibrationStage)
  {
    case 0: // Begin calibration
      calibrating = true;
      Serial.println("Calibration started...");
      calibrationCounter = 0; // This will be used as a "timer" for calibration

      IMU.readSensor();

      ax = IMU.getAccelX_mss();
      ay = IMU.getAccelY_mss();
      az = IMU.getAccelZ_mss();
      gx = IMU.getGyroX_rads();
      gy = IMU.getGyroY_rads();
      gz = IMU.getGyroZ_rads();

      directionTowardsLevel = 1;
      EP = 75 * directionTowardsLevel; // Set motor to spin a certain direction
      prevZMeas = az;
      Serial.println(String(calibrationStage) + "\t" + String(az) + "\t" + String(prevZMeas));
      Serial.println("Actuating...");
```

```
      ACTUATE_ELEVATOR();
      calibrationStage = 1;
      totalInterrupts1 = 0;
      break;
    case 1: // Find the first edge
      if (totalInterrupts1 > 45)
      {
        IMU.readSensor();

        ax = IMU.getAccelX_mss();
        ay = IMU.getAccelY_mss();
        az = IMU.getAccelZ_mss();
        gx = IMU.getGyroX_rads();
        gy = IMU.getGyroY_rads();
        gz = IMU.getGyroZ_rads();

        if (abs(prevZMeas) - abs(az) < 0.005)
        {
          directionTowardsLevel *= -1;
          EP = 75 * directionTowardsLevel;
          Serial.println("Switching direction!");
          calibrationStage = 2;
        }
        if (abs(az) > abs(maxGravZ))
        {
          maxGravZ = az;
        }

        prevZMeas = az;

        ACTUATE_ELEVATOR();
        totalInterrupts1 = 0;
      }
      break;
    case 2: // Find the second edge
      if (totalInterrupts1 > 100)
      {
        IMU.readSensor();

        ax = IMU.getAccelX_mss();
        ay = IMU.getAccelY_mss();
        az = IMU.getAccelZ_mss();
        gx = IMU.getGyroX_rads();
        gy = IMU.getGyroY_rads();
        gz = IMU.getGyroZ_rads();

        if (abs(prevZMeas) - abs(az) < 0.005)
        {
          directionTowardsLevel *= -1;
          EP = 75 * directionTowardsLevel;
          Serial.println("Switching direction!");
          calibrationStage = 3;
        }
        if (abs(az) > abs(maxGravZ))
        {
          maxGravZ = az;
        }
        prevZMeas = az;

        ACTUATE_ELEVATOR();
        totalInterrupts1 = 0;
      }
      break;
    case 3: // Slowly move back to find the center
      if (totalInterrupts1 > 20)
      {
        if (abs(az) - abs(maxGravZ) < 0.0095 || abs(az) > abs(maxGravZ))
        {
          STOP_ELEVATOR();
          calibrationStage = 4;
```

```
          Serial.println("Successfully leveled!");
        }
        else
        {
          prevZMeas = az;
          ACTUATE_ELEVATOR();
        }
      }
      break;
    case 4: // Calibrate the sensors after levelling complete
      STOP_ELEVATOR();
      EP = 0;
      IMU.calibrateGyro();
      IMU.calibrateAccel();
      calibrated = true;
      calibrating = false;
      break;
    default: // If some error occurs, restart calibration
      calibrationStage = 0;
      break;
  }
}

void HTTP_RECEIVE_MSG()
{
  // put your main code here, to run repeatedly:
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval)
  {
    // Check WiFi connection status
    if (WiFi.status() == WL_CONNECTED)
    {
      PARSE_ORIENTATION(httpGET(orientationChannel));
      previousMillis = currentMillis;
    }
    else
    {
      Serial.println("Disconnected");
    }
  }
}

void HTTP_TRY_CONNECTION()
{
  // put your main code here, to run repeatedly:
  unsigned long currentMillis = millis();
  if (!httpConnecting)
  {
    WiFi.begin(ssid, password);
    Serial.println("Initiated connection attempt");
    httpConnecting = true;
  }
  else if (currentMillis - previousMillis >= interval * 10) // Every 2.5 seconds, if no connection,
restart connection
  {
    WiFi.begin(ssid, password);
    Serial.println("Retrying connection...");
    httpConnecting = true;
    previousMillis = millis();
  }
  if (WiFi.status() == WL_CONNECTED)
  {
    Serial.println("");
    Serial.print("Connected to remote with IP Address: ");
    Serial.println(WiFi.localIP());
    httpConnecting = false;
  }
}
```

```
void DETERMINE_MOVEMENT_REQ()
{
  //pitch_des = 0;
  //yaw_des = 0;

  // Open loop control for the traverse
  if (b_Pitch > 10)
  {
    TP = min(6 * b_Pitch, NOM_PWM_VOLTAGE);
  }
  else if (b_Pitch < -10)
  {
    TP = max(6 * b_Pitch, -NOM_PWM_VOLTAGE);
  }
  else
  {
    TP = LOW;
  }

  // We use a PI control method for the elevation
  // Set desired pitch rate based on the tilt of the remote
  if (b_Roll > 15)
  {
    pitch_rate_des = min(b_Roll / 2, 30);
  }
  else if (b_Roll < -15)
  {
    pitch_rate_des = max(b_Roll / 2, -30)  ;
  }
  else
  {
    pitch_rate_des = 0;
  }

  // Set desired pitch rate
  // Here we use a PI controller to limit the pitch rate
  p_e = (pitch_rate_des) - (PITCH_RATE());

  pitch_rate_error += p_e;

  if (b_Roll > 0)
  {
    EP = - Kp_e * (p_e + (pitch_rate_error * K_i)); // More aggressive controller configuration because
of the extra torque needed to elevate the phone
  }
  else
  {
    EP = - Kp_d * (p_e + (pitch_rate_error * K_i)); // Less aggressive controller for depression of
mechanism, since the torque of the phone is biased in that direction
  }
  // Anti-windup
  if (EP > NOM_PWM_VOLTAGE)
  {
    EP = NOM_PWM_VOLTAGE;
    pitch_rate_error -= p_e;
  }
  else if (EP < -NOM_PWM_VOLTAGE)
  {
    EP = -NOM_PWM_VOLTAGE;
    pitch_rate_error -= p_e;
  }

  Serial.print(pitch);
  Serial.print("\t");
  Serial.print(pitch_des);
  Serial.print("\t");
  Serial.print(PITCH_RATE());
  Serial.print("\t");
  Serial.print(pitch_rate_des);
  Serial.print("\t");
```

```
  Serial.println(EP);
}

//EVENT CHECKERS---------------------------------
bool CONNECTED_HTTP()
{
  return WiFi.status() == WL_CONNECTED;
}

bool SWITCH_ON()
{
  int swtc = digitalRead(SWTC_1);

  if (swtc == 0)
    return false;
  else
    return true;
}

// The process of finishing calibration
bool CALIBRATED()
{
  return calibrated;
}

// The process of calibrating
bool CALIBRATING()
{
  return calibrating;
}

// UTILITY FUNCTIONS -------------------------
/*
   This is used for acquiring the data from the http address
*/
String httpGET(const char* httpAddress) {
  WiFiClient client;
  HTTPClient httpClient;

  // Examines the information at given http address
  httpClient.begin(client, httpAddress);

  // Get http message (useful for debugging error messages etc)
  int httpResponseCode = httpClient.GET();

  String payload = "--";

  if (httpResponseCode > 0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    payload = httpClient.getString();
  }
  else {
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
    httpClient.end();
    return "N";
  }
  // Free resources
  httpClient.end();

  return payload;
}

/**
   This is used for reading the input from the remote control and converting it into useful data
*/
void PARSE_ORIENTATION(String orientationMessage)
{
  if (orientationMessage != "N" && orientationMessage != "--")
```

```
  {
    //Using Strings
    int prevInd = 0;

    for (int i = 0; i <= 3; i++) {
      //Serial.println(input.indexOf(',',1));
      prevInd = commaIndex;
      if (i == 3) {
        commaIndex = orientationMessage.length();
      }
      else {
        commaIndex = orientationMessage.indexOf(',', commaIndex + 1);
      }
      if (i != 0) {
        val = orientationMessage.substring(prevInd + 1, commaIndex);
      }
      else {
        val = orientationMessage.substring(prevInd, commaIndex);
      }
      //vars[i] = val.toInt();
      //Serial.println(commaIndex);
      //Serial.println(val);
      switch (i)
      {
        case 0:
          b_State = val.toInt();
          break;
        case 1:
          b_Yaw = val.toFloat();
          break;
        case 2:
          b_Pitch = val.toFloat();
          if (b_Pitch > 180)
          {
            b_Pitch -= 360;
          }
          break;
        case 3:
          b_Roll = val.toFloat();
          if (b_Roll > 180)
          {
            b_Roll -= 360;
          }
          break;
      }
    }


    Serial.print("Received data");
    Serial.print("\t");
    Serial.print(b_State);
    Serial.print("\t");
    Serial.print(b_Yaw);
    Serial.print("\t");
    Serial.print(b_Pitch);
    Serial.print("\t");
    Serial.print(b_Roll);
    Serial.print("\t");
    prevInd = 0;
    commaIndex = 0;
  }
}

/*
   The estimation of rate of change of pitch
   (in degrees)
*/
float PITCH_RATE ()
{
  if (!CALIBRATED())
```

```
    return gx * RAD_TO_DEG;
  else
    return gx * RAD_TO_DEG;
}

/*
   The estimated rate of change of yaw, it seems we may need to account for the pitch of the device...
   (in degrees)
*/
float YAW_RATE ()
{
  if (!CALIBRATED())
    return -gz * RAD_TO_DEG;
  else
    return -gz * cos(float(pitch) * DEG_TO_RAD) * RAD_TO_DEG;
}

/*
   Blink the LED every second
*/
void BLINK_LED_1S(int LED_PIN_NUM)
{
  if (totalInterrupts0 % 2 == 0)
  {
    digitalWrite(LED_PIN_NUM, HIGH);
  }
  else
  {
    digitalWrite(LED_PIN_NUM, LOW);
  }
}
```

Remote Control:

```
#include <ETH.h>
#include <WiFi.h>

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

#include "SensorFusion.h"
#include "ESPAsyncWebServer.h"

#include <TinyGPS++.h>
#include <SoftwareSerial.h>

// HTTP broadcaster variables --------------------
const char* ssid = "FS-Beacon";
const char* password = "888888888";
AsyncWebServer server(80);
volatile bool broadcasting = false;

// I/O variables -----------------------------
#define SWTC_1 39  // On-Off switch
#define LED_13 13  // RED LED
#define LED_12 12  // GREEN LED

// State machine variables
int state = 0;
TaskHandle_t Calibration;

// MPU status ------------------------------
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus = -1;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
```

```
uint8_t fifoBuffer[64]; // FIFO storage buffer

// Position estimator variables ----------------------
SF fusion;
float x, y, z = 0; // Estimated position in world space (E frame)
float vx, vy, vz = 0; // Estimated position in world space (E frame)
float t_prev = 0; // Last measured time
float t_now = 0; // Current measured time
float dt = 0; // Amount of time since last update

int roll, pitch, yaw = 0;

// IMU calibration variables -------------------------
volatile bool calibrated = false; // Calibration completed?
volatile bool calibrating = false; // Calibration in process?

int calibrationCounter = 0; // Tracks number of times calibration measurements have been taken

int a_tolerance = 8;   //Acceptable margin of accelerometer error
int g_tolerance = 1;   //Acceptable margin of gyroscope error

float pitch_offset, roll_offset, yaw_offset = 0; // Angular offsets

// IMU variables -----------------------------
Adafruit_MPU6050 mpu;

//Timer interrupt variables -----------------------------
volatile bool interruptCounter0 = false;    // check timer interrupt 0
volatile bool interruptCounter1 = false;     // check timer interrupt 1

volatile bool switch_on = false;

int totalInterrupts0 = 0;   // counts the number of triggering of the alarm
int totalInterrupts1 = 0;   // counts the number of triggering of the alarm

hw_timer_t * timer0 = NULL;
hw_timer_t * timer1 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

//Initialization for interrupts ---------------------------------
void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
  interruptCounter0 = true; // the function to be called when timer interrupt is triggered
  totalInterrupts0++;
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  interruptCounter1 = true; // the function to be called when timer interrupt is triggered
  totalInterrupts1++;
  portEXIT_CRITICAL_ISR(&timerMux1);
}

// setting PWM properties ----------------------------
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
int MAX_PWM_VOLTAGE = 255;

#define EULER_DATA
//#define RAW_DATA
//#define PROCESSING
//#define SERIAL_PLOTER

void setup() {
  // serial to display data
  Serial.begin(115200);
```

```
  pinMode(SWTC_1, INPUT);  // configures the specified pin to behave either as an input or an output
  pinMode(LED_12, OUTPUT);
  pinMode(LED_13, OUTPUT);

  Wire.begin();
  Wire.setClock(400000);

  timer0 = timerBegin(0, 80, true);  // timer 0, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE
-> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer0, &onTime0, true); // edge (not level) triggered
  timerAlarmWrite(timer0, 1000000, true); // 2000000 * 1 us = 1 s, autoreload true

  timer1 = timerBegin(1, 80, true);  // timer 1, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE
-> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggered
  timerAlarmWrite(timer1, 20000, true); // 20000 * 1 us = 20 ms, autoreload true

  // at least enable the timer alarms
  timerAlarmEnable(timer0); // enable
  timerAlarmEnable(timer1); // enable
}

void loop() {
  // put your main code here, to run repeatedly:
  switch (state)
  {
    case 0: // Switch is off
      digitalWrite(LED_12, LOW);
      digitalWrite(LED_13, LOW);

      if (SWITCH_ON())
      {
        state = 1; // If the switch is on, move to state 1
      }
      break;
    case 1: // Switch is on, and beacon is calibrating
      BLINK_LED_1S(LED_13);
      digitalWrite(LED_12, LOW);

      if (!SWITCH_ON())
      {
        state = 0; // If the switch is off, move to state 0
        ESP.restart(); // Restart the microcontroller to clear any variables in RAM
      }
      else
      {
        if (!CALIBRATING() && !CALIBRATED())
        {
          INIT_MPU();
        }

        POSITIONAL_CALIBRATE();

        if (!BROADCASTING_HTTP()) // Start broadcasting if it is not already done so
        {
          INIT_BROADCASTER();
        }
        if (CALIBRATED() && BROADCASTING_HTTP())
        {
          state = 2;
        }
      }
      break;
    case 2: // Beacon is fully functional, but GPS is off
      BLINK_LED_1S(LED_12);
      digitalWrite(LED_13, HIGH);

      POSITIONAL_UPDATE_M();
```

```
      if (!SWITCH_ON())
      {
        state = 0; // If the switch is off, move to state 0
        ESP.restart(); // Restart the micro controller to clear any variables in RAM
      }
      break;
    default:
      BLINK_LED_1S(LED_13); // All lights flash when there is an error
      BLINK_LED_1S(LED_12);
      if (!SWITCH_ON())
      {
        state = 0; // If the switch is off, move to state 0
        ESP.restart(); // Restart the micro controller to clear any variables in RAM
      }
      break;
  }
}

// SERVICES ----------------------------------
/**
   Calibration of position data
   Calibration routine is designed to take measurements and average them to get a DC offset value.
   It is expected that calibration is done when the beacon is on the mount of the device.
*/
void POSITIONAL_CALIBRATE()
{
  if (!calibrating)
  {
    Serial.println("Begin calibration");
    totalInterrupts0 = 0;
    calibrating = true;
    calibrationCounter = 0;
  }
  else if (totalInterrupts0 >= 6 && totalInterrupts0 < 12) // wait 10 seconds before calibrating starts,
so that MPU readings will stabilize
  {
    calibrationCounter++;
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    dt = fusion.deltatUpdate();
    fusion.MadgwickUpdate(g.gyro.x, g.gyro.y, g.gyro.z, a.acceleration.x, a.acceleration.y,
a.acceleration.z, dt);  // Sensor fusion

    roll_offset += fusion.getRoll();
    pitch_offset += fusion.getPitch();
    yaw_offset += fusion.getYaw();
  }
  else if (totalInterrupts0 == 12)
  {
    roll_offset /= calibrationCounter;
    pitch_offset /= calibrationCounter;
    yaw_offset /= calibrationCounter;

    calibrated = true;
    calibrating = false;
  }
}
/*
   Use MPU data to determine the state of this device
   Store data as global variable
   This does not use DMP, but external madgwick filter to do so
*/
void POSITIONAL_UPDATE_M()
{
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  dt = fusion.deltatUpdate();
  fusion.MadgwickUpdate(g.gyro.x, g.gyro.y, g.gyro.z, a.acceleration.x, a.acceleration.y,
a.acceleration.z, dt);  // Sensor fusion
```

```
  roll = fusion.getRoll();
  pitch = fusion.getPitch();
  yaw = fusion.getYaw();

  roll -= roll_offset;
  pitch -= pitch_offset;
  yaw -= yaw_offset;

  if (roll < 0)
    roll = 360 + (roll % 360);
  if (pitch < 0)
    pitch = 360 + (pitch % 360);
  if (yaw < 0)
    yaw = 360 + (yaw % 360);

  Serial.println(GetOrientation());
}

/*
   Use MPU data to determine the state of this device
   Store data as global variable
*/
void POSITIONAL_UPDATE_GPS()
{
  // Not implemented
}

void CALIBRATE_GPS()
{
  // Not implemented
}

/*
   Start the MPU unit
*/
void INIT_MPU()
{
  // Try to initialize!
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");

  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setGyroRange(MPU6050_RANGE_250_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
}

/*
   Start broadcast of coordinates
*/
void INIT_BROADCASTER()
{
  WiFi.softAP(ssid, password);

  IPAddress IP = WiFi.softAPIP();
  Serial.print("IP address: ");
  Serial.println(IP);

  server.on("/orientation", HTTP_GET, [](AsyncWebServerRequest * request) {
    request->send_P(200, "text/plain", GetOrientation().c_str());
  });
  server.begin();

  broadcasting = true;
}
```

```
/*
   Shut down broadcasting during a state of 0...
*/
void DISCONNECT_BROADCASTER()
{
  if (BROADCASTING_HTTP())
  {
    server.end();
    WiFi.disconnect();
    broadcasting = false;
  }
}

/*
   Blink the LED every second
*/
void BLINK_LED_1S(int LED_PIN_NUM)
{
  if (totalInterrupts0 % 2 == 0)
  {
    digitalWrite(LED_PIN_NUM, HIGH);
  }
  else
  {
    digitalWrite(LED_PIN_NUM, LOW);
  }
}

// EVENT CHECKERS -----------------------------------
// Is the switch on???
bool SWITCH_ON()
{
  int swtc = digitalRead(SWTC_1);

  if (swtc == 0)
    return false;
  else
    return true;
}

// The process of finishing calibration
bool CALIBRATED()
{
  return calibrated;
}

// The process of calibrating
bool CALIBRATING()
{
  return calibrating;
}

bool BROADCASTING_HTTP()
{
  return broadcasting;
}

bool GPS_AVAILABLE()
{
  if (gps.satellites.isValid())
  {
    return (gps.satellites.value() > 2);
  }
  return false;
}

// Utility functions
String GetStatus()
{
```

```
  //return "s:" + String(state) + "t:" + dt;
  return "";
}

String GetOrientation()
{
  //return "p:" + String(ypr[1]) + ";r:" + String(ypr[2]) + ";y:" + String(ypr[0]);
  return String(state) + ',' + String(yaw) + ',' + String(pitch) + ',' + String(roll);
}

String GetPosition()
{
  //return "x:" + String(x) + ";y:" + String(y) + ";z:" + String(z) + ";vx:" + String(vx) + ";vy:" +
String(vy) + ";vz:" + String(vz) + ";ax:" + String(aaReal.x) + ";ay:" + String(aaReal.y) + ";az:" +
String(aaReal.z);
  return "";
}
```

Appendix D - Remote Control Wiring and State Machine Diagrams

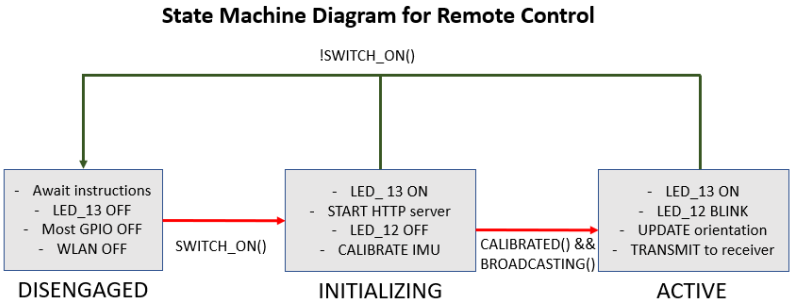

Figure AD-1: State machine diagram for the remote control.

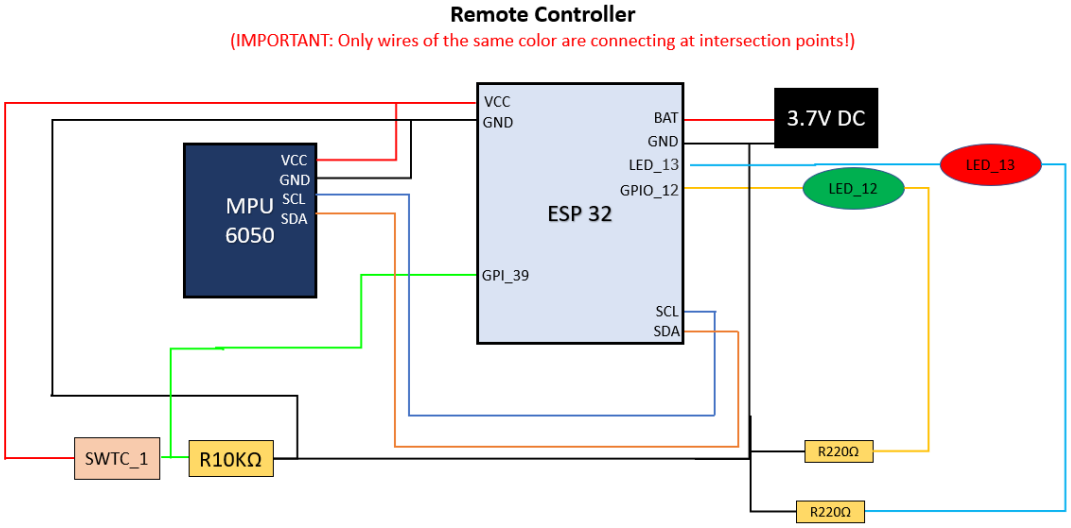

Figure AD-2: Wiring schematic for the remote-control device.

Appendix E - Remote Control Components

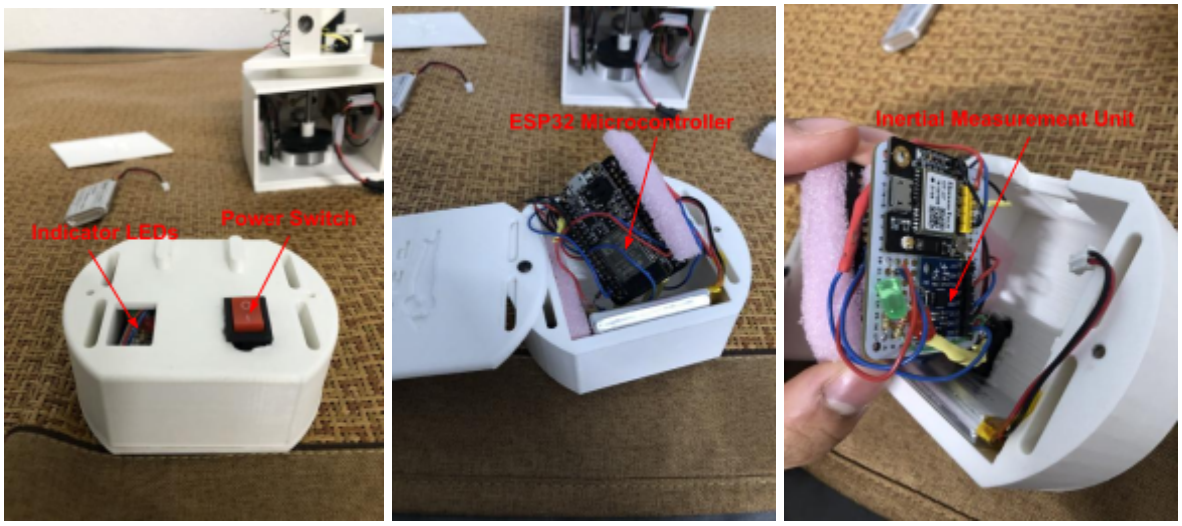

Figure AE1: The components of the remote control are illustrated here.