

MECENG 102B Final Report

Group 21

Samuel Leo Mankoff, Puneet Bhaskar, Suk Young Jang, Un U Li

1. Opportunity:

Our device was designed as a solution to help individuals who are lost in the wilderness without any electronic devices to get back to safety by retracing their steps. Given that a plethora of solutions from traditional compass and maps to advanced GPS location systems already exist, we wanted to focus on providing a low-tech solution that could work even in the absence of any network connection or access to the open skies.

2. Strategy

The original strategy envisioned a device that would have 2 modes: tracking, and pointing. In the tracking mode, an onboard gyroscope would log the position changes taken by the user. In the pointing mode it would direct the user in the way they came from using an arrow with 2 degrees of freedom. Another major consideration was compactness of design and portability.

Facing budget and difficulty constraints we scaled the scope of the project down in our implementation. As such, the pointing mode was changed to react to a set of external remote controllers(Fig 1), while the tracking mode was changed to hold an absolute position set in the pointing mode regardless of the user's orientation. We were able to satisfy these goals, ensuring that our pointing needle reacted in real time to external controls and the tracking mode maintained the absolute position set by the controls. We were also reasonably successful in minimising the overall size of the product, reducing it to a device that can be held with 2 hands.(Fig 2).

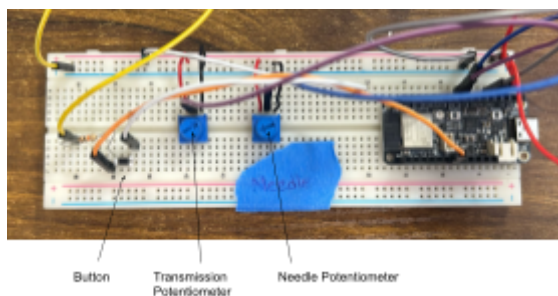


Fig 1: Remote Controller

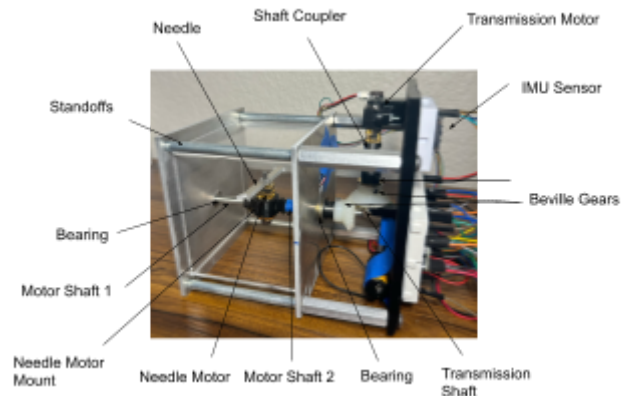


Fig 2: Complete assembly

3. Function-Critical Decisions

Choice of motors

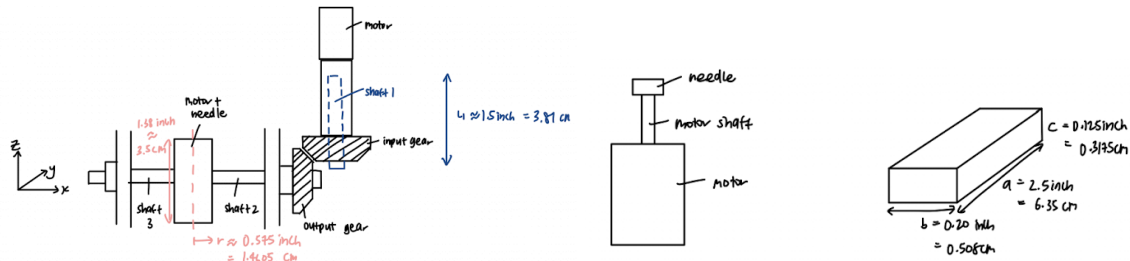


Fig. 3.4: Transmission Shaft System (Left). Needle System (Right)

We decided on the Polulu #2215 dc motors from the lab kits due to its size and the low torque required for our application. Since that motor gear has a 1:1 gear ratio, input and output gear rotate at the same speed, but in different directions, the input and output torques are the same.

$$T_{input} = T_{output}$$

Since the motor and needle system is free to rotate about the x-axis, and there is minimal external force acting on it, we only consider torque due to inertia of the load. For a rotational load, this is given by:

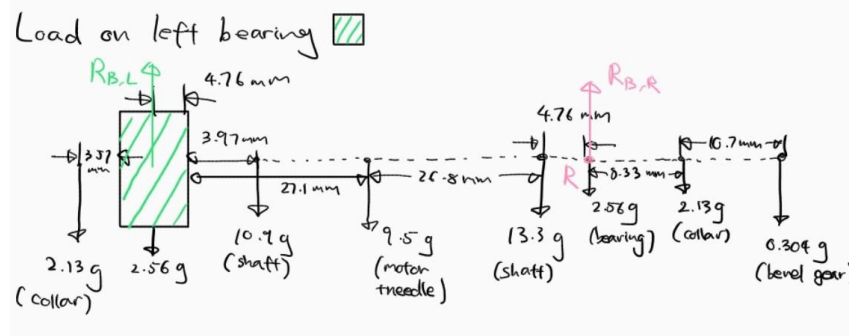
$$T_{output} = I\alpha$$

where I is the moment of inertia of motor and needle load and α is the angular acceleration. I can be calculated by approximating the motor and needle as a solid cylinder and α can be calculated by setting the system to reach its desired position in 1 second. The same principle is applied to calculate torque on the needle, but now approximating the needle as a cuboid. This reveals that

$$T_{stall,motor} > 1.6T_{input}$$

for both cases. Hence, our chosen motor was suitable.

Choice of bearings



We opted for the 1/4 " McMaster stainless steel ball bearing due to its cost and size fit. The selected ball bearing has a maximum radial static load of 90lb. To calculate the force on each bearing, we measured the moments caused by each component with respect to the left bearing. Using the fact that $\Sigma M = 0$, we were able to calculate the force exerted on the bearings as

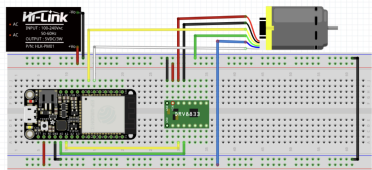
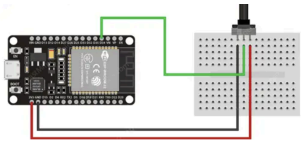
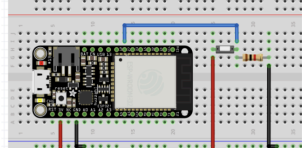
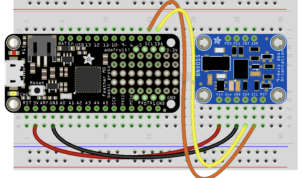
$$R_{B,Left} = 0.209 \text{ N (Equivalent to 21.3g mass)}$$

$$R_{B,Right} = 0.217 \text{ N (Equivalent to 22.1g mass)}$$

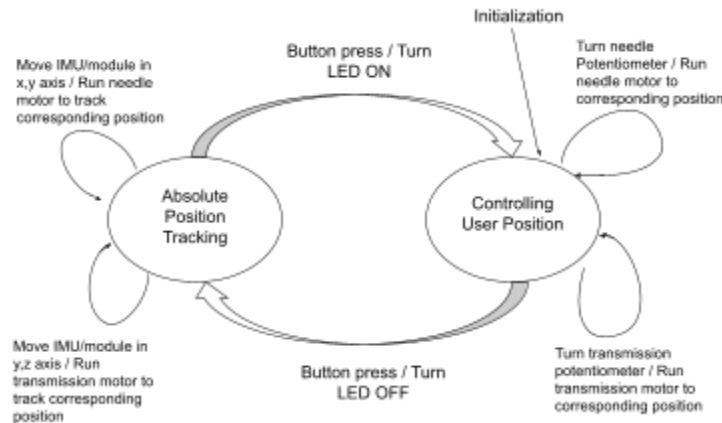
Therefore, $R_{B,Left}$ and $R_{B,Right} \ll 90\text{lbs}$ maximum static load of selected bearings

4. Circuit and State Transition Diagrams

We used two Pololu #2215 dc motors. We wired one of them exactly as shown below and the other we used the second set of inputs and outputs (AIN1,AIN2,AOUT1,AOUT2) on the motor driver and other available GPIO pins on the ESP.

 <p>Figure 6: Connecting the encoder</p>			
<p><u>Wiring for IMU to ESP</u>(pins match exactly):</p>	<p><u>Potentiometer wiring:</u></p>	<p><u>Push Button Wiring:</u></p>	<p><u>IMU Wiring:</u></p>

State Transition Diagram



5. Reflection

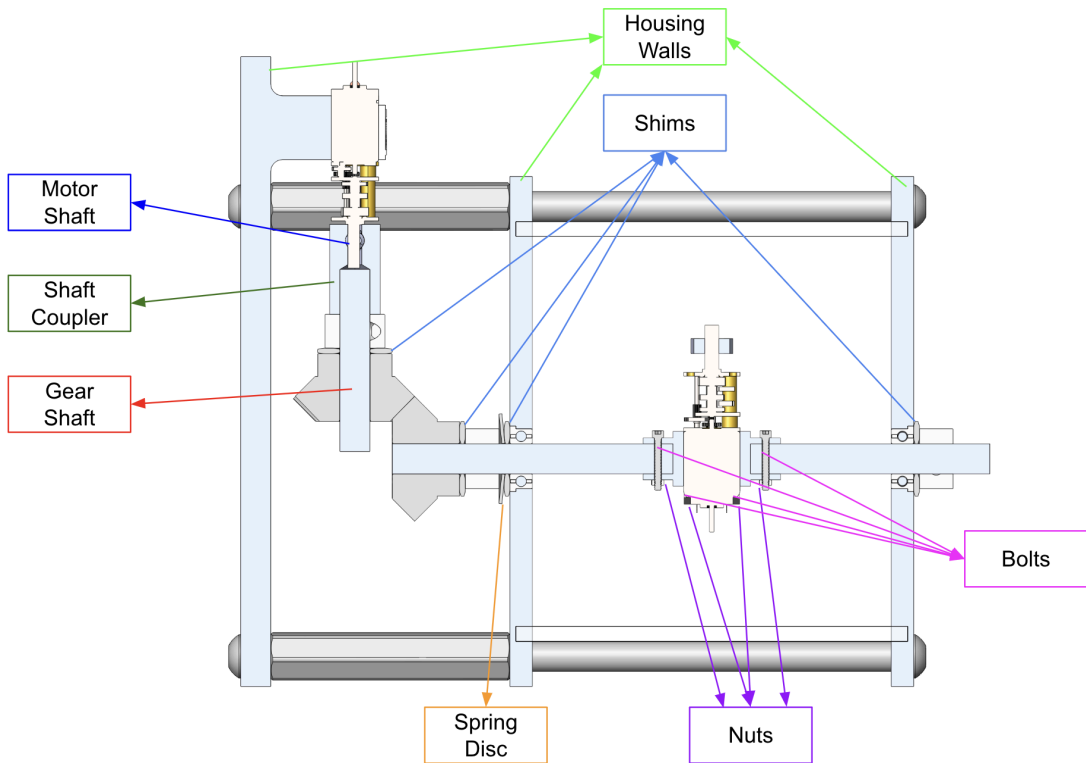
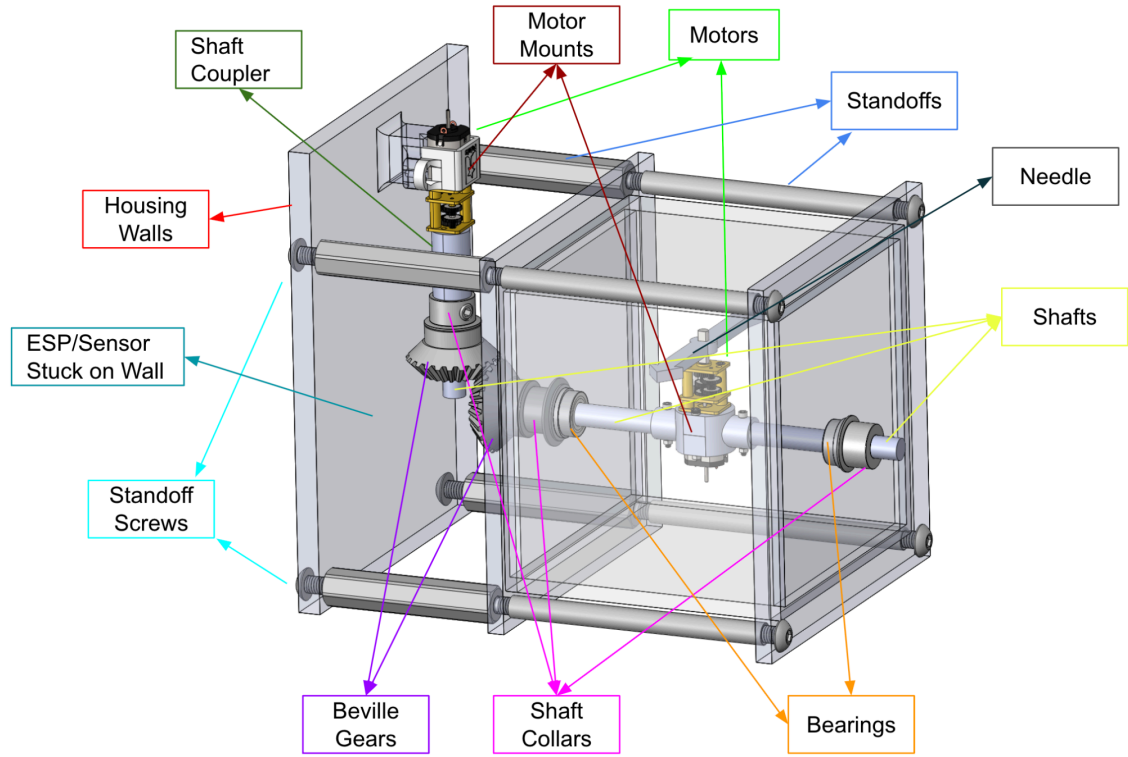
Working on this project was both fun and challenging. We had minor issues when it came to manufacturing such as parts not fitting as planned. Software-wise, most of it was straightforward but debugging took a long time since we integrated a lot of things such as the motors and the button. Also, the PID controller was very difficult to tune perfectly. Although our goal was to make the needle move accordingly with the potentiometers or completely track our motion, accuracy issues in our sensor prevented it from moving as intended. However, we were able to get consistent results since the needle responded to potentiometer changes and movement from the user.

Appendix:

Complete Bill of Materials

Name	Source	Number	Cost	Pkg Quantity	Quantity Needed	Total Price	Order Status	Links
Flanged Ball Bearings	McMaster	57155K323	6.05	1	2	12.1	Arrived	https://www.mcmaster.com/57155K323/
Shaft Collars	McMaster	9414T6	1.77	1	4	7.08	Arrived	https://www.mcmaster.com/9414T6/
Shims	McMaster	3088A145	6.86	5	2	6.86	Arrived	https://www.mcmaster.com/3088A145/
Belleville Disc Spring	McMaster	9712K116	2.44	3	1	2.44	Arrived	https://www.mcmaster.com/9712K116/
Bevel Gear	McMaster	7297K15	11.83	1	2	23.66	Arrived	https://www.mcmaster.com/7297K15/
Shaft Coupler	Servocity	https://www.servocity.com/0-250-to-3mm-set-scr	8.19	1	2	16.38	Arrived	https://www.servocity.com/0-250-to-3mm-set-screw-sha
DC Motor	Lab Kit	https://www.pololu.com/product/2215	0	1	2	0	Arrived	https://www.pololu.com/product/2215
Motor Driver	Lab Kit/Adafruit	DRV8833	0	1	1	0	Arrived	
Aluminum Shafts	McMaster	1257K118	10.73	1	1	10.73	Arrived	https://www.mcmaster.com/1257K118/
ESP 32	Lab Kit	HUZZAH32 - ESP32 Feather Board	0	1	2	0	Arrived	
Gyroscope Sensor	Lab Kit/ Sparkf	LSM 6DSO	0	1	1	0	Arrived	
12V Battery	Amazon	https://www.amazon.com/Amazon-Basics-23A-Al	4.74	1	1	4.74	Arrived	https://www.amazon.com/Amazon-Basics-23A-Alkaline
Breadboard	Amazon	https://www.amazon.com/dp/B09YN9PVY2/ref=s	8.99	9	3	8.99	Arrived	https://www.amazon.com/dp/B09YN9PVY2/ref=sspa_d
Potentiometer	Lab Kit		0	1	2	0	Arrived	
3.7V Battery (for ESP32)	Lab Kit		0	1	2	0	Arrived	
Aluminum Sheet (Hous	Machined	8975K421	10.97	1	1	10.97	Arrived	https://www.mcmaster.com/8975K921/
Acrylic Sheet (Housing)	Machined	1227T869	8.58	1	1	8.58	Arrived	https://www.mcmaster.com/1227T869-1227T862/
Standoffs	McMaster	93265A034	3.03	1	4	12.12	Arrived	https://www.mcmaster.com/93265A034/
Screws	McMaster	92949A267	11.61	100	1	11.61	Arrived	https://www.mcmaster.com/92949A267/
Motor Mount Screw	McMaster	91251A059	8.89	25	1	8.89	Arrived	https://www.mcmaster.com/91251A059/
Screw Nuts (all)	McMaster	92736A001	10.36	10	1	10.36	Arrived	https://www.mcmaster.com/92736A001/
Shaft to Motor Mount S	McMaster	91251A447	10.16	5	1	10.16	Arrived	https://www.mcmaster.com/91251A447/
Hex Keys	Home Depot	https://www.homedepot.com/p/TEKTON-3-64-3-8-in-Lo	12	1	1	12	Arrived	https://www.homedepot.com/p/TEKTON-3-64-3-8-in-Lo
Total Cost						177.67		

CAD



Code

```
1  #include <Arduino.h>
2  #include <Wire.h>
3  #include <Adafruit_Sensor.h>
4  #include <Adafruit_BNO055.h>
5  #include <utility/imuMaths.h>
6  #include <ESP32Encoder.h>
7  #include <WiFi.h>
8  #include <esp_now.h>
9
10 #define I2C_SDA 22
11 #define I2C_SCL 20
12
13 //needle motor pinout initialization
14 #define BIN_1 14
15 #define BIN_2 15
16 // transmission motor pinout initialization
17 #define AIN_1 7
18 #define AIN_2 8
19 #define LED_PIN 13
20
21 #define BUTTON 34// Set button to gpio pin
22
23 Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28);
24
25 ESP32Encoder encoder1;
26 ESP32Encoder encoder2;
27
28 // Initializing state
29 byte state = 0;
30
31 //Initializing Variables
32 int timerFreq= 10000;
33 int theta = 0;
34 int thetaDes1 = 0;
35 int thetaDes2 = 0;
36
37 volatile int xytheta = 0;
38 volatile int xythetaDes = 0;
39 volatile int yztheta = 0;
40 volatile int yzthetaDes = 0;
41
```

```

42 int thetaMotorNeedle = 360; // 75.8 * 6 counts per revolution
43 int thetaMotorTrans = 250; // 75.8 * 6 counts per revolution
44 int needleThetaDes = 0;
45 int D = 0;
46 int potReading1 = 0;
47 int potReading2 = 0;
48 int error = 0;
49 int error_sum = 0;
50 int prev_error = 0; // Previous error for derivative calculation
51 int derivative = 0;
52
53 int Kp = 3; // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
54 int Ki = 0.3;
55 int Kd = 0.5;
56 int IMax = 0;
57
58 //Setup interrupt variables -----
59 volatile int count = 0; // encoder count
60 volatile int count1 = 0; // encoder count for second motor
61 volatile bool interruptCounter = false; // check timer interrupt 1
62 volatile bool deltaT = false; // check timer interrupt 2
63 volatile bool deltaT1 = false; // check timer interrupt 3
64 int totalInterrupts = 0; // counts the number of triggering of the alarm
65 hw_timer_t* timer0 = NULL;
66 hw_timer_t* timer1 = NULL;
67 hw_timer_t * timer2 = NULL;
68 hw_timer_t * timer3 = NULL;
69 portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
70 portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
71 portMUX_TYPE timerMux2 = portMUX_INITIALIZER_UNLOCKED;
72 portMUX_TYPE timerMux3 = portMUX_INITIALIZER_UNLOCKED;
73 volatile bool buttonIsPressed = false; //
74 volatile bool DEBOUNCINGflag = false;
75 // volatile bool BUTTONflag = false;
76
77 // setting PWM properties -----
78 const int freq = 5000;
79 const int ledChannel_1 = 1;
80 const int ledChannel_2 = 2;
81 const int resolution = 8;
82 const int MAX_PWM_VOLTAGE = 255;
83 const int NOM_PWM_VOLTAGE = 150;
84
85 //Setup angle variables-----
86 float init_yaw = 0;
87 float init_roll = 0;
88 float init_pitch = 0;
89 float delta_yaw = 0;
90 float delta_roll = 0;
91 float delta_pitch = 0;
92

```

```

93 //Initialization -----
94 void IRAM_ATTR onTime0() { //IMU interupt
95     portENTER_CRITICAL_ISR(&timerMux0);
96     interruptCounter = true; // the function to be called when timer interrupt is triggered
97     count = encoder1.getCount();
98     encoder1.clearCount();
99     portEXIT_CRITICAL_ISR(&timerMux0);
100 }
101
102 void IRAM_ATTR onTime1() { //POT1 interupt
103     portENTER_CRITICAL_ISR(&timerMux1);
104     count = encoder1.getCount();
105     encoder1.clearCount();
106     deltaT = true; // the function to be called when timer interrupt is triggered
107     portEXIT_CRITICAL_ISR(&timerMux1);
108 }
109
110 void IRAM_ATTR isr() { // button interupt
111     buttonIsPressed = true; // the function to be called when interrupt is triggered
112 }
113
114 void IRAM_ATTR onTime2() { // Debouncing, Button interupt
115     portENTER_CRITICAL_ISR(&timerMux2);
116     DEBOUNCINGflag = false; // the function to be called when interrupt is triggered
117     portEXIT_CRITICAL_ISR(&timerMux2);
118     timerStop(timer2);
119 }
120
121 void IRAM_ATTR onTime3() { // POT2 interupt
122     portENTER_CRITICAL_ISR(&timerMux3);
123     count1 = encoder2.getCount();
124     encoder2.clearCount();
125     deltaT1 = true; // the function to be called when timer interrupt is triggered
126     portEXIT_CRITICAL_ISR(&timerMux3);
127 }
128
129 // initialization of recieved variables from other board
130 typedef struct struct_message {
131     int POT_1;
132     int POT_2;
133     bool BUTTON_1;
134 } struct_message;
135
136 // Create a struct_message called myData
137 struct_message myData;
138
139 // callback function that will be executed when data is received
140 void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
141     memcpy(&myData, incomingData, sizeof(myData));
142     // Serial.print("Bytes received: ");
143     // Serial.println(len);
144     // Serial.print("Pot 1: ");
145     // Serial.println(myData.POT_1);
146     // Serial.print("Pot 2: ");
147     // Serial.println(myData.POT_2);
148     // Serial.print("Button pressed boolean value: ");
149     // Serial.print(myData.BUTTON_1);
150 }
151

```



```

152 void setup() {
153     // put your setup code here, to run once:
154     Serial.begin(115200);
155     Wire.begin(I2C_SDA, I2C_SCL); // setup SDA and SCL channels for IMU
156     delay(500);
157     Wire.begin();
158     delay(10);
159
160     //Setup for I2C wifi communication
161     WiFi.mode(WIFI_STA);
162
163     //IMU initialization
164     Serial.println("BN0055 Euler Angles Test");
165     if(!bno.begin())
166     {
167         Serial.println("BN0055 not detected. Check wiring or I2C address!");
168         while(1);
169     }
170
171     delay(1000);
172     bno.setExtCrystalUse(true);
173
174     sensors_event_t event;
175     bno.getEvent(&event);
176     init_roll = event.orientation.x;
177     init_pitch = event.orientation.y;
178     init_yaw = event.orientation.z;
179
180     // Init ESP-NOW
181     if (esp_now_init() != ESP_OK) {
182         Serial.println("Error initializing ESP-NOW");
183         return;
184     }
185
186     // Once ESPNow is successfully Init, we will register for recv CB to
187     // get recv packer info
188     esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));
189
190     pinMode(LED_PIN, OUTPUT);
191     digitalWrite(LED_PIN, HIGH); // sets the initial state of LED as turned-off
192
193     ESP32Encoder::useInternalWeakPullResistors = puType::up; // Enable the weak pull up resistors
194     encoder1.attachHalfQuad(27, 33); // C 27, 33 // Attache pins for use as encoder pins
195     encoder1.setCount(0); // set starting count value after attaching
196
197     ESP32Encoder::useInternalWeakPullResistors = puType::up; // Enable the weak pull up resistors
198     encoder2.attachHalfQuad(4, 32); // C 4, 32 // Attache pins for use as encoder pins
199     encoder2.setCount(0); // set starting count value after attaching
200
201     // Method 2
202     // configure PWM functionalitites with attaching the channel to the GPIO to be controller
203     ledcAttach(BIN_1, freq, resolution);
204     ledcAttach(BIN_2, freq, resolution);
205
206     ledcAttach(AIN_1, freq, resolution);
207     ledcAttach(AIN_2, freq, resolution);
208

```

```
209 // initialize timers
210 timer0 = timerBegin(1000000); // Set timer frequency to 1Mhz
211 timerAttachInterrupt(timer0, &onTime0); // Attach onTimer0 function to our timer.
212 timerAlarm(timer0, 10000, true, 0); // 5000000 * 1 us = 5 s, autoreload true
213
214 timer1 = timerBegin(1000000/3); // Set timer frequency to 1Mhz
215 timerAttachInterrupt(timer1, &onTime1); // Attach onTimer1 function to our timer.
216 timerAlarm(timer1, 10000, true, 0); // 10000 * 1 us = 10 ms, autoreload true
217
218 timer2 = timerBegin(1000000); // Set timer frequency to 1Mhz
219 timerAttachInterrupt(timer2, &onTime2); // Attach onTimer1 function to our timer.
220 timerAlarm(timer2, 10000, true, 0); // 10000 * 1 us = 10 ms, autoreload true
221
222 timer3 = timerBegin(1000000/2); // Set timer frequency to 1Mhz
223 timerAttachInterrupt(timer3, &onTime3); // Attach onTimer1 function to our timer.
224 timerAlarm(timer3, 10000, true, 0); // 10000 * 1 us = 10 ms, autoreload true
225
226 Serial.begin(115200);
227 pinMode(BUTTON, INPUT); // configures the specified pin to behave either as an input or an output
228 attachInterrupt(BUTTON, isr, RISING); // set the "BTN" pin as the interrupt pin; call function named "isr" when
229 // the interrupt is triggered; "Rising" means triggering interrupt when the
230 // pin goes from LOW to HIGH
231
232 Serial.println("end of setup");
233 }
234
```

```

234
235 void loop() {
236
237 switch (state) {
238
239 case 0: // Motors being controlled by user
240     Serial.println("State 1: Using the potentiometers to direct the users movements");
241     // event checker
242     if (CheckForButtonPress()== true) {
243         // service response/fucntion
244         digitalWrite(LED_PIN, LOW);
245         state = 1;
246         // service response/fucntion
247         ButtonResponse();
248         break;
249
250     // event checker
251     } else if (deltaT) { // checks and contorls motor when POT1 is turned
252         portENTER_CRITICAL(&timerMux1);
253         deltaT = false;
254         portEXIT_CRITICAL(&timerMux1);
255
256         xytheta += count;
257         xythetaDes = potpositionMapping(myData.POT_1, thetaMotorNeedle);
258         // service response/fucntion
259         motorControl(xythetaDes, xytheta, BIN_1, BIN_2);
260
261     // event checker
262     } else if (deltaT1) { // checks and contorls motor when POT2 is turned
263         portENTER_CRITICAL(&timerMux3);
264         deltaT1 = false;
265         portEXIT_CRITICAL(&timerMux3);
266
267         yztheta += count1;
268         yzthetaDes = potpositionMapping(myData.POT_2, thetaMotorTrans);
269         // service response/fucntion
270         motorControl(yzthetaDes, yztheta, AIN_2, AIN_1);
271
272     }else {
273         break;
274     }
275     break;
276
277

```

```
278 case 1: // Absolut position
279     Serial.println("State 2: Absolute Position control for the needle ");
280     // event checker
281     if (CheckForButtonPress()== false) {
282         // service response/fucntion
283         digitalWrite(LED_PIN, HIGH);
284         state = 0;
285         // service response/fucntion
286         ButtonResponse();
287
288         break;
289
290     // event checker
291     }
292     if (interruptCounter) {
293         portENTER_CRITICAL(&timerMux0);
294         interruptCounter = false;
295         portEXIT_CRITICAL(&timerMux0);
296
297         // service response/fucntion
298         trackingMotorContorl(init_roll, init_pitch, init_yaw);
299
300     }else {
301         break;
302     }
303     break;
304
305
306
307 }
308 }
309
310
```

```

311 //Service functions
312
313 |
314 // Button Event Checker
315 bool CheckForButtonPress(){
316     if(myData.BUTTON_1 == true){
317         return true;
318     } else {
319         return false;
320     }
321 }
322
323 void ButtonResponse(){
324     buttonIsPressed = false;
325     Serial.println("Pressed!");
326 }
327
328 float potpositionMapping(int reading,int range) {
329     float thetaDes = map(reading, 0, 4095,0,range);
330     return thetaDes;
331 }
332
333 void motorControl(float thetaDes, float theta, int input1, int input2){
334     error = thetaDes - theta;
335     error_sum = error_sum + error/10;
336     derivative = (error - prev_error) / 0.01; // Assuming deltaT is 10 ms = 0.01 seconds
337     D = (Kp * error) + (Ki * error_sum) + (Kd * derivative);
338     prev_error = error;
339
340     //Ensure that you don't go past the maximum possible command
341     if (D > MAX_PWM_VOLTAGE) {
342         D = MAX_PWM_VOLTAGE;
343         error_sum -= error;
344     } else if (D < -MAX_PWM_VOLTAGE) {
345         D = -MAX_PWM_VOLTAGE;
346         error_sum -= error;
347     }
348
349     if (D > 0) {
350         ledcWrite(input1, LOW);
351         ledcWrite(input2, D);
352     } else if (D < 0) {
353         ledcWrite(input2, LOW);
354         ledcWrite(input1, -D);
355     } else {
356         ledcWrite(input2, LOW);
357         ledcWrite(input1, LOW);
358     }
359
360 }
361

```

```

362 void trackingMotorControl(float init_roll, float init_pitch, float init_yaw){
363
364     sensors_event_t event;
365     bno.getEvent(&event);
366
367     delta_roll = calculateAngleDifference(event.orientation.x, init_roll);
368     delta_pitch = calculateAngleDifference(event.orientation.y, init_pitch);
369     delta_yaw = calculateAngleDifference(event.orientation.z, init_yaw);
370
371     init_roll = event.orientation.x;
372     init_pitch = event.orientation.y;
373     init_yaw = event.orientation.z;
374
375     // theta += count;
376     thetaDes1 = map(-delta_pitch, -360, 360, -thetaMotorNeedle*1.5, thetaMotorNeedle*1.5);
377     thetaDes2 = map(-delta_roll, -360, 360, -thetaMotorNeedle, thetaMotorNeedle);
378
379     // service response/function
380     motorControl(thetaDes2, 0, BIN_1, BIN_2); // theta is always zero here needle
381     motorControl(thetaDes1, 0, AIN_2, AIN_1); // theta is always zero here transmission
382
383 }
384
385 float calculateAngleDifference(float current, float previous) {
386     float diff = current - previous;
387     if (diff > 180) {
388         diff -= 360;
389     } else if (diff < -180) {
390         diff += 360;
391     }
392     return diff;
393 }

```