src\main.cpp

```
1 #include <Arduino.h>
 2 #include <AccelStepper.h>
 3 #include <Bounce2.h>
 4 #include <SolarCalculator.h>
 5
  #include <TimeLib.h>
 6
 7
     // === Pin Definitions ===
 8
 9
     // Gnomon motor (Motor 2)
     #define GNOMON STEP PIN 27
                                 // DRV8825 STEP (Gnomon motor)
10
     #define GNOMON DIR PIN 32
                                  // DRV8825 DIR (Gnomon motor)
11
12
     #define GNOMON EN PIN 25
                                   // DRV8825 EN (Gnomon motor)
     AccelStepper gnomonMotor(AccelStepper::DRIVER, GNOMON_STEP_PIN, GNOMON_DIR_PIN);
13
14
     int gnomonZeroPosition = 0;
15
     // Base motor (Motor 1)
16
     #define BASE STEP PIN 14
                                  // DRV8825 STEP (Base motor)
17
     #define BASE_DIR_PIN 12
                                  // DRV8825 DIR (Base motor)
18
19
     #define BASE EN PIN 13
                                  // DRV8825 EN (Base motor)
20
     AccelStepper baseMotor(AccelStepper::DRIVER, BASE_STEP_PIN, BASE_DIR_PIN);
21
     // Light Dependent Resistors (LDR)
22
     #define LDR RIGHT PIN 34
23
                                 // Photocell (LDR) right side
     #define LDR_LEFT_PIN 26
24
                                 // Photocell (LDR) left side
25
26
     // Limit switch
     #define LIMIT_SWITCH_PIN 5
                                 // Limit switch (Normally Open)
27
28
     // LEDs
29
30
     #define LED_GREEN_PIN 18
                                  // Green LED (anode)
     #define LED_YELLOW_PIN 19
                                  // Yellow LED (anode)
31
32
     #define LED_RED_PIN 21
                                  // Red LED (anode)
33
34
     // Pushbuttons
35
     #define BUTTON GREEN PIN 4 // Green pushbutton
36
     #define BUTTON_RED_PIN 2
                                 // Red pushbutton
37
38
     Bounce greenButton = Bounce();
     Bounce redButton = Bounce();
39
40
41
     //Sun position values
42
     double latitude = 56.5047;
                                  // Your latitude
     double longitude = 21.0108; // Your longitude
43
                                  // Local time zone offset
44
     int utc_offset = 3;
45
46
     // Set manually:
47
     int setHour = 5;
48
     int setMinute = 50;
49
     int setSecond = 0;
50
     int setDay = 1;
```

```
51
      int setMonth = 6;
52
       int setYear = 2025;
53
54
      double sunAzimuth = 0.0;
      int azimuthTarget = 0;
55
56
57
      //Serial send delay
      unsigned long lastDataSendTime = 0;
58
59
       const unsigned long dataSendInterval = 1000; // send every 1000 ms = 1 second
60
      //Serial communication data
61
62
      bool timeReceived = false;
63
      bool locationReceived = false;
64
65
      //Serial Commands
      /*
66
67
68
    TIME:12,0,0,1,6,2025
    LOC:56.946,24.105
69
70
      */
71
72
      // System states
73 enum SystemState {
74
      SYSTEM_OFF,
75
      SYSTEM ON,
76
      SYSTEM_PAUSED,
77
      SYSTEM_RESET
78
      };
79
80 SystemState currentState = SYSTEM_OFF;
81
82 //Subsstates
83 enum SystemSubState {
84
      STEP ZERO GNOMON,
85
      STEP_ZERO_BASE,
86
      SUN_TRACKING,
87
      STEP_ROTATE_BASE,
88
      STEP_ROTATE_GNOMON,
89
      STEP_DONE
90
    };
91
92
    SystemSubState subState = STEP_ZERO_GNOMON;
93
94
      // AS5600 Analog Outputs
95
      #define AS5600_GNOMON_PIN 33 // Gnomon AS5600 analog out
      #define AS5600 BASE PIN 35 // Base AS5600 analog out
96
97
      const int BASE_SENSOR_OFFSET = -120; // Opposite of 2700 on a 12-bit scale
98
99
      // Solar positioning values
      int SunOffsetAmount = round(90 * (4096.0 / 360.0)); // Sun axiom Offset angle
100
       int gnomonOffsetAmount = (round(-(latitude-11.77)) * (4096.0 / 360.0)); // Gnomon
101
    Offset angle
      int gnomonZeroTarget = round(-220 * (4096.0 / 360.0)); // Gnomon Zero target angle
102
```

```
103
       const int baseZeroTarget = 2150; // or whatever value you consider zero
104
105
     int readBaseSensorOffset() {
106
       int raw = analogRead(AS5600_BASE_PIN);
       return (raw + BASE_SENSOR_OFFSET + 4096) % 4096;
107
108
     }
109
110 struct MotorRotationState {
       bool started = false;
111
112
       bool complete = false;
       int startingPosition = 0;
113
114
    };
115
116 MotorRotationState baseOffsetState;
117
     MotorRotationState gnomonOffsetState;
118
    MotorRotationState gnomonTargetState;
119
     MotorRotationState baseZeroState;
    MotorRotationState baseTrackingState;
120
121
122
    void resetMotorStates() {
123
       baseOffsetState.started = false;
124
       baseOffsetState.complete = false;
125
       gnomonOffsetState.started = false;
126
127
       gnomonOffsetState.complete = false;
128
129
       gnomonTargetState.started = false;
130
       gnomonTargetState.complete = false;
131
132
       baseZeroState.started = false;
       baseZeroState.complete = false;
133
134
135
       baseTrackingState.started = false;
136
       baseTrackingState.complete = false;
137
     }
138
     bool trackLightSource(AccelStepper &motor, int ldrLeftPin, int ldrRightPin,
139
     MotorRotationState &state) {
140
       const int tolerance = 5;
141
       const int minBrightness = 2000;
142
       const float brightnessThresholdRatio = 0.8;
143
       static int maxBrightnessSeen = 0;
144
       static unsigned long lastMaxUpdate = 0;
145
146
       if (state.complete) return true;
147
148
       if (!state.started) {
149
         motor.setMaxSpeed(200);
150
         state.started = true;
         Serial.println("Light tracking started...");
151
152
       }
153
154
       int leftValue = analogRead(ldrLeftPin);
```

```
155
       int rightValue = analogRead(ldrRightPin);
       int maxCurrent = (leftValue + rightValue)/2;
156
157
158
       // Update peak brightness
       if (millis() - lastMaxUpdate > 60000 || maxCurrent > maxBrightnessSeen) {
159
160
         maxBrightnessSeen = maxCurrent;
         lastMaxUpdate = millis();
161
162
       }
163
164
       int brightnessThreshold = brightnessThresholdRatio * maxBrightnessSeen;
165
166
       Serial.print("LDR Left: ");
167
       Serial.print(leftValue);
168
       Serial.print(" | LDR Right: ");
       Serial.print(rightValue);
169
       Serial.print(" | Max seen: ");
170
       Serial.print(maxBrightnessSeen);
171
172
       Serial.print(" | Required: ");
173
       Serial.println(brightnessThreshold);
174
175
       int difference = leftValue - rightValue;
176
177
       // Check if aligned and bright enough
178
       if (
179
         abs(difference) <= tolerance &&</pre>
180
         leftValue >= minBrightness &&
         rightValue >= minBrightness &&
181
182
         maxCurrent >= brightnessThreshold
183
       ) {
184
         motor.stop();
         state.complete = true;
185
186
         delay(2000);
187
         Serial.println("Light tracking complete.");
         return true;
188
       }
189
190
191
       // Keep moving toward brighter side
       motor.setSpeed(difference > 0 ? -50 : 50);
192
193
       motor.runSpeed();
194
       return false;
195
     }
196
     bool rotateMotorByOffset(AccelStepper &motor, int sensorPin,int offsetAngle,
197
     MotorRotationState &state, int maxSpeed = 200, int moveSpeed = 100, int tolerance = 2)
     {
198
       if (state.complete) return true;
199
200
       if (!state.started) {
201
         motor.setMaxSpeed(maxSpeed);
202
         state.startingPosition = analogRead(sensorPin);
203
         state.started = true;
204
205
         Serial.print("Starting position: ");
```

```
206
         Serial.println(state.startingPosition);
207
         Serial.println("Rotating motor by offset...");
208
       }
209
       int currentPosition = analogRead(sensorPin);
210
211
       int targetPosition = (state.startingPosition + offsetAngle + 4096) % 4096;
212
       int difference = (targetPosition - currentPosition + 4096) % 4096;
213
214
       if (difference > 2048) difference -= 4096;
215
       Serial.print("Current: ");
216
217
       Serial.print(currentPosition);
218
       Serial.print(" | Target: ");
219
       Serial.print(targetPosition);
       Serial.print(" | Diff: ");
220
       Serial.println(difference);
221
222
223
       if (abs(difference) <= tolerance) {</pre>
224
         motor.stop();
225
         state.complete = true;
226
         Serial.println("Offset rotation complete.");
227
         return true;
228
       }
229
230
       motor.setSpeed(difference > 0 ? -moveSpeed : moveSpeed);
231
       motor.runSpeed();
       return false;
232
233
     }
234
235
     bool rotateMotorToPosition(AccelStepper &motor, int sensorPin, int targetPosition,
     MotorRotationState &state) {
       const int tolerance = 2;
236
237
238
       if (state.complete) return true;
239
       if (!state.started) {
240
241
         motor.setMaxSpeed(200);
         state.started = true;
242
243
         Serial.print("Target Position: ");
244
         Serial.println(targetPosition);
245
       }
246
247
       int currentPosition = analogRead(sensorPin);
248
       int difference = (targetPosition - currentPosition + 4096) % 4096;
249
       if (difference > 2048) difference -= 4096;
250
251
       Serial.print("Current: ");
252
       Serial.print(currentPosition);
       Serial.print(" | Target: ");
253
254
       Serial.print(targetPosition);
       Serial.print(" | Diff: ");
255
       Serial.println(difference);
256
257
```

```
258
       if (abs(difference) <= tolerance) {</pre>
259
         motor.stop();
260
         state.complete = true;
261
         Serial.println("Motor reached target position.");
262
         return true;
263
       }
264
265
       motor.setSpeed(difference > 0 ? -100 : 100);
266
       motor.runSpeed();
267
       return false;
268
     }
269
270
    void checkButtons() {
271
       greenButton.update();
       redButton.update();
272
273
       static unsigned long redPressStart = 0;
274
275
       static bool redHoldChecked = false;
276
277
       bool greenPressed = greenButton.read() == HIGH;
278
       bool redPressed = redButton.read() == HIGH;
279
280
       // Handle both buttons for system reset
281
     if (greenPressed && redPressed) {
282
         currentState = SYSTEM RESET;
283
         Serial.println("System RESET");
284
         digitalWrite(LED_GREEN_PIN, HIGH);
285
         digitalWrite(LED_YELLOW_PIN, HIGH);
286
         digitalWrite(LED_RED_PIN, HIGH);
287
         return;
       }
288
289
290
       // Green button press
291
       if (greenButton.rose() && !redPressed) {
292
         currentState = SYSTEM_ON;
293
         Serial.println("System ON");
294
       }
295
296
       // Red button press
297
       if (redButton.rose()) {
298
         currentState = SYSTEM_PAUSED;
299
         redPressStart = millis();
300
         redHoldChecked = false;
301
         Serial.println("System PAUSED");
302
       }
303
304
       // Check if red has been held for 3 seconds
305
       if (redPressed && currentState == SYSTEM_PAUSED && !redHoldChecked) {
         if (millis() - redPressStart >= 3000) {
306
307
           currentState = SYSTEM OFF;
308
           redHoldChecked = true;
309
           Serial.println("System OFF (long hold)");
310
         }
```

```
311
       }
312
       // Reset the flag if red is released
313
314
       if (redButton.rose()) {
         redHoldChecked = false;
315
316
       }
317
     }
318
319
     void calculateSunAzimuth() {
320
       // Set system time
321
       setTime(setHour, setMinute, setSecond, setDay, setMonth, setYear);
322
       time_t utc = now();
323
324
       double elevation;
       calcHorizontalCoordinates(utc, (round(-(latitude-11.77)) * (4096.0 / 360.0)),
325
     longitude, sunAzimuth, elevation);
326
327
       Serial.print("Sun Azimuth: ");
328
       Serial.print(sunAzimuth);
       Serial.println("°");
329
330
331
       // Convert to sensor units (0-4096)
       azimuthTarget = round((sunAzimuth / 360.0) * 4096.0);
332
333
       Serial.print("Sensor units: ");
334
       Serial.println(azimuthTarget);
335
     }
336
337
     void parseSerialCommand(const String& input) {
       if (input.startsWith("TIME:")) {
338
         sscanf(input.c_str(), "TIME:%d,%d,%d,%d,%d,%d,%d",
339
340
                &setHour, &setMinute, &setSecond, &setDay, &setMonth, &setYear);
341
342
         timeReceived = true;
343
         Serial.println("Time variables updated.");
344
       }
345
       else if (input.startsWith("LOC:")) {
346
         sscanf(input.c_str(), "LOC:%lf,%lf", &latitude, &longitude);
347
         locationReceived = true;
348
         Serial.println("Location variables updated.");
349
       }
350
       else {
351
         Serial.println("Unknown command.");
352
       }
     }
353
354
355
     void receiveSerialData() {
       static String inputBuffer = "";
356
357
358
       while (Serial.available()) {
359
         char c = Serial.read();
360
361
         if (c == '\n') {
362
           // Process complete input line
```

```
363
           parseSerialCommand(inputBuffer);
           inputBuffer = "";
364
         } else {
365
366
           inputBuffer += c;
367
         }
368
       }
369
     }
370
371
     void sendSystemData() {
372
       // Get current time from TimeLib
373
       char nowTimestamp[20];
374
       snprintf(nowTimestamp, sizeof(nowTimestamp), "%04d-%02d-%02d %02d:%02d:%02d",
375
                year(), month(), day(), hour(), minute(), second());
376
377
       time_t utc = now();
378
       double elevation;
       calcHorizontalCoordinates(utc, round(-(latitude-11.77) * (4096.0 / 360.0)),
379
     longitude, sunAzimuth, elevation);
380
381
       // Begin with identifiable prefix
382
       Serial.print("DATA:");
383
384
       // Send CSV-formatted line over Serial
385
       Serial.print(nowTimestamp); Serial.print(",");
386
       Serial.print(latitude); Serial.print(",");
387
       Serial.print(longitude); Serial.print(",");
388
       Serial.print(sunAzimuth); Serial.print(",");
       Serial.print(analogRead(LDR LEFT PIN)); Serial.print(",");
389
390
       Serial.print(analogRead(LDR_RIGHT_PIN)); Serial.print(",");
       Serial.print(readBaseSensorOffset()); Serial.print(",");
391
392
393
       // Send system state as a string
394
       switch (currentState) {
                             Serial.println("SYSTEM OFF"); break;
395
         case SYSTEM OFF:
                              Serial.println("SYSTEM_ON"); break;
396
         case SYSTEM ON:
         case SYSTEM_PAUSED: Serial.println("SYSTEM_PAUSED"); break;
397
398
         case SYSTEM RESET: Serial.println("SYSTEM RESET"); break;
399
       }
400
     }
401
402
     // === Setup ===
403
     void setup() {
404
       Serial.begin(115200);
405
406
       // Initialize LED pins
407
       pinMode(LED_GREEN_PIN, OUTPUT);
408
       pinMode(LED YELLOW PIN, OUTPUT);
       pinMode(LED_RED_PIN, OUTPUT);
409
410
411
       // Example: Turn all LEDs off at startup
412
       digitalWrite(LED_GREEN_PIN, LOW);
413
       digitalWrite(LED_YELLOW_PIN, LOW);
414
       digitalWrite(LED_RED_PIN, LOW);
```

```
415
416
       pinMode(GNOMON_EN_PIN, OUTPUT);
417
       pinMode(BASE EN PIN, OUTPUT);
418
419
       //Attach buttons
420
       greenButton.attach(BUTTON_GREEN_PIN, INPUT);
421
       redButton.attach(BUTTON_RED_PIN, INPUT);
422
423
       // Set debounce interval
424
       greenButton.interval(10); // or whatever fits your physical setup
425
       redButton.interval(10);
426
       Serial.println("System OFF");
427
428
     }
429
430
    // === Loop ===
     void loop() {
431
432
       checkButtons();
433
       switch (currentState) {
434
         case SYSTEM_OFF:
435
           digitalWrite(GNOMON_EN_PIN, HIGH); // Enable gnomon driver
436
           digitalWrite(BASE_EN_PIN, HIGH);
                                                // Enable base driver
437
           digitalWrite(LED_GREEN_PIN, LOW);
438
           digitalWrite(LED_YELLOW_PIN, LOW);
           digitalWrite(LED_RED_PIN, HIGH);
439
440
           break;
441
442
         case SYSTEM ON:
443
           if (!timeReceived || !locationReceived) {
444
               receiveSerialData();
               Serial.println("Waiting for TIME and LOCATION data...");
445
               delay(1000);
446
447
               break;
           }
448
           digitalWrite(GNOMON_EN_PIN, LOW); // Enable gnomon driver
449
450
           digitalWrite(BASE_EN_PIN, LOW);
                                               // Enable base driver
451
           switch (subState) {
452
             case STEP_ZERO_GNOMON:
453
454
               digitalWrite(LED GREEN PIN, LOW);
455
               digitalWrite(LED_YELLOW_PIN, HIGH);
456
               digitalWrite(LED_RED_PIN, LOW);
               rotateMotorByOffset(gnomonMotor, AS5600_GNOMON_PIN, gnomonZeroTarget,
457
     gnomonTargetState);
               if ((digitalRead(LIMIT_SWITCH_PIN) == HIGH)) {
458
459
                 calculateSunAzimuth();
                 subState = STEP_ZERO_BASE;
460
461
               }
462
               break;
463
464
             case STEP_ZERO_BASE:
465
               digitalWrite(LED_GREEN_PIN, LOW);
               digitalWrite(LED_YELLOW_PIN, HIGH);
466
```

```
467
               digitalWrite(LED_RED_PIN, LOW);
               if (rotateMotorToPosition(baseMotor, AS5600_BASE_PIN, baseZeroTarget,
468
     baseZeroState)) {
                 delay(1000);
469
                 subState = SUN_TRACKING;
470
471
               }
472
               break;
473
474
             case SUN_TRACKING:
475
               digitalWrite(LED_GREEN_PIN, LOW);
476
               digitalWrite(LED_YELLOW_PIN, HIGH);
477
               digitalWrite(LED_RED_PIN, LOW);
               if (trackLightSource(baseMotor, LDR LEFT PIN, LDR RIGHT PIN,
478
     baseTrackingState)) {
479
                 subState = STEP_ROTATE_BASE; // Or whatever comes next
480
               }
481
             break;
482
483
             case STEP_ROTATE_BASE:
               digitalWrite(LED_GREEN_PIN, LOW);
484
485
               digitalWrite(LED_YELLOW_PIN, HIGH);
486
               digitalWrite(LED_RED_PIN, LOW);
               if (rotateMotorByOffset(baseMotor, AS5600 BASE PIN, azimuthTarget,
487
     baseOffsetState)) {
488
                 subState = STEP_ROTATE_GNOMON;
489
               }
490
               break;
491
492
             case STEP ROTATE GNOMON:
               digitalWrite(LED_GREEN_PIN, LOW);
493
494
               digitalWrite(LED_YELLOW_PIN, HIGH);
495
               digitalWrite(LED RED PIN, LOW);
               if (rotateMotorByOffset(gnomonMotor, AS5600_GNOMON_PIN, gnomonOffsetAmount,
496
     gnomonOffsetState)) {
497
                 subState = STEP_DONE;
498
               }
499
               break;
500
501
             case STEP_DONE:
502
               // System is now fully set up
503
               digitalWrite(LED_GREEN_PIN, HIGH);
               digitalWrite(LED_YELLOW_PIN, LOW);
504
               digitalWrite(LED_RED_PIN, LOW);
505
               Serial.println("All system tasks complete.");
506
               break;
507
508
           }
509
           break;
510
         case SYSTEM_PAUSED:
511
512
           digitalWrite(LED_GREEN_PIN, LOW);
           digitalWrite(LED_YELLOW_PIN, HIGH);
513
           digitalWrite(LED_RED_PIN, LOW);
514
515
           break;
```

```
516
517
        case SYSTEM_RESET:
518
           // Reset everything
519
           resetMotorStates();
520
          timeReceived = false;
521
           locationReceived = false;
522
           subState = STEP_ZERO_GNOMON;
523
           break;
524
       }
      if (millis() - lastDataSendTime >= dataSendInterval) {
525
526
           lastDataSendTime = millis();
           sendSystemData();
527
528
      }
529
    }
530
531
532
```