src\main.cpp

```cpp
#include <Arduino.h>
#include <AccelStepper.h>
#include <Bounce2.h>
#include <SolarCalculator.h>
#include <TimeLib.h>

  // === Pin Definitions ===

  // Gnomon motor (Motor 2)
  #define GNOMON_STEP_PIN 27    // DRV8825 STEP (Gnomon motor)
  #define GNOMON_DIR_PIN 32     // DRV8825 DIR (Gnomon motor)
  #define GNOMON_EN_PIN 25      // DRV8825 EN (Gnomon motor)
  AccelStepper gnomonMotor(AccelStepper::DRIVER, GNOMON_STEP_PIN, GNOMON_DIR_PIN);
  int gnomonZeroPosition = 0;

  // Base motor (Motor 1)
  #define BASE_STEP_PIN 14      // DRV8825 STEP (Base motor)
  #define BASE_DIR_PIN 12       // DRV8825 DIR (Base motor)
  #define BASE_EN_PIN 13        // DRV8825 EN (Base motor)
  AccelStepper baseMotor(AccelStepper::DRIVER, BASE_STEP_PIN, BASE_DIR_PIN);

  // Light Dependent Resistors (LDR)
  #define LDR_RIGHT_PIN 34      // Photocell (LDR) right side
  #define LDR_LEFT_PIN 26       // Photocell (LDR) left side

  // Limit switch
  #define LIMIT_SWITCH_PIN 5    // Limit switch (Normally Open)

  // LEDs
  #define LED_GREEN_PIN 18      // Green LED (anode)
  #define LED_YELLOW_PIN 19     // Yellow LED (anode)
  #define LED_RED_PIN 21        // Red LED (anode)

  // Pushbuttons
  #define BUTTON_GREEN_PIN 4    // Green pushbutton
  #define BUTTON_RED_PIN 2      // Red pushbutton

  Bounce greenButton = Bounce();
  Bounce redButton = Bounce();

  //Sun position values
  double latitude = 56.5047;     // Your latitude
  double longitude = 21.0108;    // Your longitude
  int utc_offset = 3;            // Local time zone offset

  // Set manually:
  int setHour = 5;
  int setMinute = 50;
  int setSecond = 0;
  int setDay = 1;
  int setMonth = 6;
  int setYear = 2025;

  double sunAzimuth = 0.0;
  int azimuthTarget = 0;

  //Serial send delay
  unsigned long lastDataSendTime = 0;
  const unsigned long dataSendInterval = 1000; // send every 1000 ms = 1 second

  //Serial communication data
  bool timeReceived = false;
  bool locationReceived = false;

  //Serial Commands
  /*

TIME:12,0,0,1,6,2025
LOC:56.946,24.105
  */

  // System states
enum SystemState {
  SYSTEM_OFF,
  SYSTEM_ON,
  SYSTEM_PAUSED,
  SYSTEM_RESET
  };

SystemState currentState = SYSTEM_OFF;

//Subsstates
enum SystemSubState {
  STEP_ZERO_GNOMON,
  STEP_ZERO_BASE,
  SUN_TRACKING,
  STEP_ROTATE_BASE,
  STEP_ROTATE_GNOMON,
  STEP_DONE
};

SystemSubState subState = STEP_ZERO_GNOMON;

  // AS5600 Analog Outputs
  #define AS5600_GNOMON_PIN 33  // Gnomon AS5600 analog out
  #define AS5600_BASE_PIN 35    // Base AS5600 analog out
  const int BASE_SENSOR_OFFSET = -120; // Opposite of 2700 on a 12-bit scale

  // Solar positioning values
  int  SunOffsetAmount = round(90 * (4096.0 / 360.0));  // Sun axiom Offset angle
  int  gnomonOffsetAmount = (round(-(latitude-11.77)) * (4096.0 / 360.0));  // Gnomon Offset angle
  int  gnomonZeroTarget = round(-220 * (4096.0 / 360.0));  // Gnomon Zero target angle
  const int baseZeroTarget = 2150;  // or whatever value you consider zero

int readBaseSensorOffset() {
  int raw = analogRead(AS5600_BASE_PIN);
  return (raw + BASE_SENSOR_OFFSET + 4096) % 4096;
}

struct MotorRotationState {
  bool started = false;
  bool complete = false;
  int startingPosition = 0;
};

MotorRotationState baseOffsetState;
MotorRotationState gnomonOffsetState;
MotorRotationState gnomonTargetState;
MotorRotationState baseZeroState;
MotorRotationState baseTrackingState;

void resetMotorStates() {
  baseOffsetState.started = false;
  baseOffsetState.complete = false;

  gnomonOffsetState.started = false;
  gnomonOffsetState.complete = false;

  gnomonTargetState.started = false;
  gnomonTargetState.complete = false;

  baseZeroState.started = false;
  baseZeroState.complete = false;

  baseTrackingState.started = false;
  baseTrackingState.complete = false;
}

bool trackLightSource(AccelStepper &motor, int ldrLeftPin, int ldrRightPin, MotorRotationState &state) {
  const int tolerance = 5;
  const int minBrightness = 2000;
  const float brightnessThresholdRatio = 0.8;
  static int maxBrightnessSeen = 0;
  static unsigned long lastMaxUpdate = 0;

  if (state.complete) return true;

  if (!state.started) {
    motor.setMaxSpeed(200);
    state.started = true;
    Serial.println("Light tracking started...");
  }

  int leftValue = analogRead(ldrLeftPin);
  int rightValue = analogRead(ldrRightPin);
  int maxCurrent = (leftValue + rightValue)/2;

  // Update peak brightness
  if (millis() - lastMaxUpdate > 60000 || maxCurrent > maxBrightnessSeen) {
    maxBrightnessSeen = maxCurrent;
    lastMaxUpdate = millis();
  }

  int brightnessThreshold = brightnessThresholdRatio * maxBrightnessSeen;

  Serial.print("LDR Left: ");
  Serial.print(leftValue);
  Serial.print(" | LDR Right: ");
  Serial.print(rightValue);
  Serial.print(" | Max seen: ");
  Serial.print(maxBrightnessSeen);
  Serial.print(" | Required: ");
  Serial.println(brightnessThreshold);

  int difference = leftValue - rightValue;

  // Check if aligned and bright enough
  if (
    abs(difference) <= tolerance &&
    leftValue >= minBrightness &&
    rightValue >= minBrightness &&
    maxCurrent >= brightnessThreshold
  ) {
    motor.stop();
    state.complete = true;
    delay(2000);
    Serial.println("Light tracking complete.");
    return true;
  }

  // Keep moving toward brighter side
  motor.setSpeed(difference > 0 ? -50 : 50);
  motor.runSpeed();
  return false;
}

bool rotateMotorByOffset(AccelStepper &motor, int sensorPin,int offsetAngle, MotorRotationState &state, int maxSpeed = 200, int moveSpeed = 100, int tolerance = 2) {
  if (state.complete) return true;

  if (!state.started) {
    motor.setMaxSpeed(maxSpeed);
    state.startingPosition = analogRead(sensorPin);
    state.started = true;

    Serial.print("Starting position: ");
    Serial.println(state.startingPosition);
    Serial.println("Rotating motor by offset...");
  }

  int currentPosition = analogRead(sensorPin);
  int targetPosition = (state.startingPosition + offsetAngle + 4096) % 4096;

  int difference = (targetPosition - currentPosition + 4096) % 4096;
  if (difference > 2048) difference -= 4096;

  Serial.print("Current: ");
  Serial.print(currentPosition);
  Serial.print(" | Target: ");
  Serial.print(targetPosition);
  Serial.print(" | Diff: ");
  Serial.println(difference);

  if (abs(difference) <= tolerance) {
    motor.stop();
    state.complete = true;
    Serial.println("Offset rotation complete.");
    return true;
  }

  motor.setSpeed(difference > 0 ? -moveSpeed : moveSpeed);
  motor.runSpeed();
  return false;
}

bool rotateMotorToPosition(AccelStepper &motor, int sensorPin, int targetPosition, MotorRotationState &state) {
  const int tolerance = 2;

  if (state.complete) return true;

  if (!state.started) {
    motor.setMaxSpeed(200);
    state.started = true;
    Serial.print("Target Position: ");
    Serial.println(targetPosition);
  }

  int currentPosition = analogRead(sensorPin);
  int difference = (targetPosition - currentPosition + 4096) % 4096;
  if (difference > 2048) difference -= 4096;

  Serial.print("Current: ");
  Serial.print(currentPosition);
  Serial.print(" | Target: ");
  Serial.print(targetPosition);
  Serial.print(" | Diff: ");
  Serial.println(difference);

  if (abs(difference) <= tolerance) {
    motor.stop();
    state.complete = true;
    Serial.println("Motor reached target position.");
    return true;
  }

  motor.setSpeed(difference > 0 ? -100 : 100);
  motor.runSpeed();
  return false;
}

void checkButtons() {
  greenButton.update();
  redButton.update();

  static unsigned long redPressStart = 0;
  static bool redHoldChecked = false;

  bool greenPressed = greenButton.read() == HIGH;
  bool redPressed = redButton.read() == HIGH;

  // Handle both buttons for system reset
if (greenPressed && redPressed) {
    currentState = SYSTEM_RESET;
    Serial.println("System RESET");
    digitalWrite(LED_GREEN_PIN, HIGH);
    digitalWrite(LED_YELLOW_PIN, HIGH);
    digitalWrite(LED_RED_PIN, HIGH);
    return;
  }

  // Green button press
  if (greenButton.rose() && !redPressed) {
    currentState = SYSTEM_ON;
    Serial.println("System ON");
  }

  // Red button press
  if (redButton.rose()) {
    currentState = SYSTEM_PAUSED;
    redPressStart = millis();
    redHoldChecked = false;
    Serial.println("System PAUSED");
  }

  // Check if red has been held for 3 seconds
  if (redPressed && currentState == SYSTEM_PAUSED && !redHoldChecked) {
    if (millis() - redPressStart >= 3000) {
      currentState = SYSTEM_OFF;
      redHoldChecked = true;
      Serial.println("System OFF (long hold)");
    }
  }

  // Reset the flag if red is released
  if (redButton.rose()) {
    redHoldChecked = false;
  }
}

void calculateSunAzimuth() {
  // Set system time
  setTime(setHour, setMinute, setSecond, setDay, setMonth, setYear);
  time_t utc = now();

  double elevation;
  calcHorizontalCoordinates(utc, (round(-(latitude-11.77)) * (4096.0 / 360.0)), longitude, sunAzimuth, elevation);

  Serial.print("Sun Azimuth: ");
  Serial.print(sunAzimuth);
  Serial.println("°");

  // Convert to sensor units (0-4096)
  azimuthTarget = round((sunAzimuth / 360.0) * 4096.0);
  Serial.print("Sensor units: ");
  Serial.println(azimuthTarget);
}

void parseSerialCommand(const String& input) {
  if (input.startsWith("TIME:")) {
    sscanf(input.c_str(), "TIME:%d,%d,%d,%d,%d,%d",
           &setHour, &setMinute, &setSecond, &setDay, &setMonth, &setYear);

    timeReceived = true;
    Serial.println("Time variables updated.");
  }
  else if (input.startsWith("LOC:")) {
    sscanf(input.c_str(), "LOC:%lf,%lf", &latitude, &longitude);
    locationReceived = true;
    Serial.println("Location variables updated.");
  }
  else {
    Serial.println("Unknown command.");
  }
}

void receiveSerialData() {
  static String inputBuffer = "";

  while (Serial.available()) {
    char c = Serial.read();

    if (c == '\n') {
      // Process complete input line
      parseSerialCommand(inputBuffer);
      inputBuffer = "";
    } else {
      inputBuffer += c;
    }
  }
}

void sendSystemData() {
  // Get current time from TimeLib
  char nowTimestamp[20];
  snprintf(nowTimestamp, sizeof(nowTimestamp), "%04d-%02d-%02d %02d:%02d:%02d",
           year(), month(), day(), hour(), minute(), second());

  time_t utc = now();
  double elevation;
  calcHorizontalCoordinates(utc, round(-(latitude-11.77) * (4096.0 / 360.0)), longitude, sunAzimuth, elevation);

  // Begin with identifiable prefix
  Serial.print("DATA:");

  // Send CSV-formatted line over Serial
  Serial.print(nowTimestamp); Serial.print(",");
  Serial.print(latitude); Serial.print(",");
  Serial.print(longitude); Serial.print(",");
  Serial.print(sunAzimuth); Serial.print(",");
  Serial.print(analogRead(LDR_LEFT_PIN)); Serial.print(",");
  Serial.print(analogRead(LDR_RIGHT_PIN)); Serial.print(",");
  Serial.print(readBaseSensorOffset()); Serial.print(",");

  // Send system state as a string
  switch (currentState) {
    case SYSTEM_OFF:    Serial.println("SYSTEM_OFF"); break;
    case SYSTEM_ON:     Serial.println("SYSTEM_ON"); break;
    case SYSTEM_PAUSED: Serial.println("SYSTEM_PAUSED"); break;
    case SYSTEM_RESET:  Serial.println("SYSTEM_RESET"); break;
  }
}

// === Setup ===
void setup() {
  Serial.begin(115200);

  // Initialize LED pins
  pinMode(LED_GREEN_PIN, OUTPUT);
  pinMode(LED_YELLOW_PIN, OUTPUT);
  pinMode(LED_RED_PIN, OUTPUT);

  // Example: Turn all LEDs off at startup
  digitalWrite(LED_GREEN_PIN, LOW);
  digitalWrite(LED_YELLOW_PIN, LOW);
  digitalWrite(LED_RED_PIN, LOW);

  pinMode(GNOMON_EN_PIN, OUTPUT);
  pinMode(BASE_EN_PIN, OUTPUT);

  //Attach buttons
  greenButton.attach(BUTTON_GREEN_PIN, INPUT);
  redButton.attach(BUTTON_RED_PIN, INPUT);

  // Set debounce interval
  greenButton.interval(10);  // or whatever fits your physical setup
  redButton.interval(10);
  Serial.println("System OFF");

}

// === Loop ===
void loop() {
  checkButtons();
  switch (currentState) {
    case SYSTEM_OFF:
      digitalWrite(GNOMON_EN_PIN, HIGH);  // Enable gnomon driver
      digitalWrite(BASE_EN_PIN, HIGH);    // Enable base driver
      digitalWrite(LED_GREEN_PIN, LOW);
      digitalWrite(LED_YELLOW_PIN, LOW);
      digitalWrite(LED_RED_PIN, HIGH);
      break;

    case SYSTEM_ON:
      if (!timeReceived || !locationReceived) {
          receiveSerialData();
          Serial.println("Waiting for TIME and LOCATION data...");
          delay(1000);
          break;
      }
      digitalWrite(GNOMON_EN_PIN, LOW);  // Enable gnomon driver
      digitalWrite(BASE_EN_PIN, LOW);    // Enable base driver
      switch (subState) {
        case STEP_ZERO_GNOMON:

          digitalWrite(LED_GREEN_PIN, LOW);
          digitalWrite(LED_YELLOW_PIN, HIGH);
          digitalWrite(LED_RED_PIN, LOW);
          rotateMotorByOffset(gnomonMotor, AS5600_GNOMON_PIN, gnomonZeroTarget, gnomonTargetState);
          if ((digitalRead(LIMIT_SWITCH_PIN) == HIGH)) {
            calculateSunAzimuth();
            subState = STEP_ZERO_BASE;
          }
          break;

        case STEP_ZERO_BASE:
          digitalWrite(LED_GREEN_PIN, LOW);
          digitalWrite(LED_YELLOW_PIN, HIGH);
          digitalWrite(LED_RED_PIN, LOW);
          if (rotateMotorToPosition(baseMotor, AS5600_BASE_PIN, baseZeroTarget, baseZeroState)) {
            delay(1000);
            subState = SUN_TRACKING;
          }
          break;

        case SUN_TRACKING:
          digitalWrite(LED_GREEN_PIN, LOW);
          digitalWrite(LED_YELLOW_PIN, HIGH);
          digitalWrite(LED_RED_PIN, LOW);
          if (trackLightSource(baseMotor, LDR_LEFT_PIN, LDR_RIGHT_PIN, baseTrackingState)) {
            subState = STEP_ROTATE_BASE;  // Or whatever comes next
          }
        break;

        case STEP_ROTATE_BASE:
          digitalWrite(LED_GREEN_PIN, LOW);
          digitalWrite(LED_YELLOW_PIN, HIGH);
          digitalWrite(LED_RED_PIN, LOW);
          if (rotateMotorByOffset(baseMotor, AS5600_BASE_PIN, azimuthTarget, baseOffsetState)) {
            subState = STEP_ROTATE_GNOMON;
          }
          break;

        case STEP_ROTATE_GNOMON:
          digitalWrite(LED_GREEN_PIN, LOW);
          digitalWrite(LED_YELLOW_PIN, HIGH);
          digitalWrite(LED_RED_PIN, LOW);
          if (rotateMotorByOffset(gnomonMotor, AS5600_GNOMON_PIN, gnomonOffsetAmount, gnomonOffsetState)) {
            subState = STEP_DONE;
          }
          break;

        case STEP_DONE:
          // System is now fully set up
          digitalWrite(LED_GREEN_PIN, HIGH);
          digitalWrite(LED_YELLOW_PIN, LOW);
          digitalWrite(LED_RED_PIN, LOW);
          Serial.println("All system tasks complete.");
          break;
      }
      break;

    case SYSTEM_PAUSED:
      digitalWrite(LED_GREEN_PIN, LOW);
      digitalWrite(LED_YELLOW_PIN, HIGH);
      digitalWrite(LED_RED_PIN, LOW);
      break;
    case SYSTEM_RESET:
      // Reset everything
      resetMotorStates();
      timeReceived = false;
      locationReceived = false;
      subState = STEP_ZERO_GNOMON;
      break;
  }
  if (millis() - lastDataSendTime >= dataSendInterval) {
      lastDataSendTime = millis();
      sendSystemData();
  }
}
```