جامعة عفت

EFFAT UNIVERSITY

CS 4111 -1 - Programming Embedded Systems

Solar tracking Project Report - Fall 2023

Student Name: **Jana Abu Hantash - S21107114**

Instructor: **Dr. Zain Balfagih**

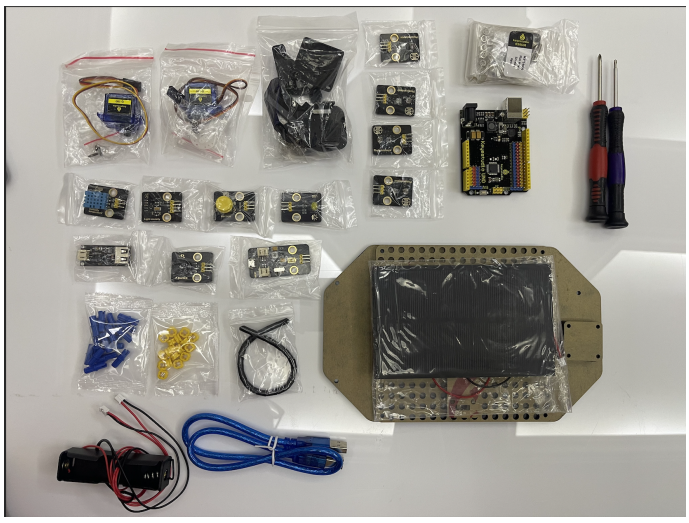
Date Last Edited: December 16, 2023

Contents

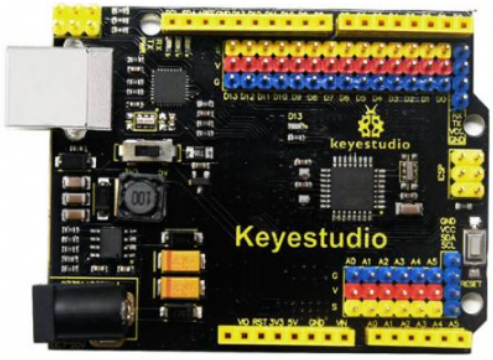
1 Introduction

The Keyestudio Sun Follower project is an innovative approach to optimizing solar energy collection. Utilizing a combination of sensors and actuators, the system dynamically adjusts the orientation of a solar panel to maximize exposure to sunlight, enhancing energy efficiency. This project embodies a practical application of embedded systems in renewable energy technology.

2 Hardware Components

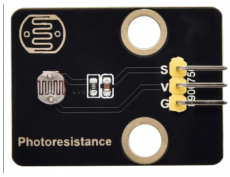


1. **Keyestudio UNO Development Board:** This board is the project's brain, handling sensor data processing and actuator control. Its ATMEGA328P microcontroller offers ample computational power for real-time sensor data analysis and servo motor control.

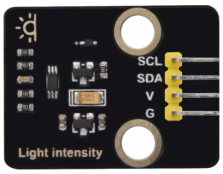


2. **Sensors**

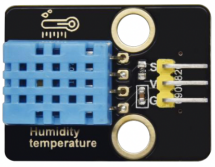
   - **BH1750 Light Sensor:** Measures the intensity of ambient light. This data is crucial for determining the optimal angle for the solar panel to face the sun.



- **Digital Light Intensity Module:** Enhances the accuracy of light measurement, contributing to the precision in solar panel positioning.



- **DHT11 Temperature and Humidity Sensor:** Provides ambient temperature and humidity readings, which are essential for monitoring environmental conditions that could affect the solar panel's performance.



- **Raindrop Sensor:** Detects the presence of rain or moisture, playing a key role in protecting the solar panel from adverse weather conditions.

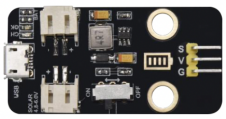
3. **Actuators**

- **Servo Motors:** Two precision motors control the solar panel's horizontal and vertical orientation. Their movements are based on data from the light sensors, ensuring that the panel maintains optimal exposure to sunlight throughout the day.

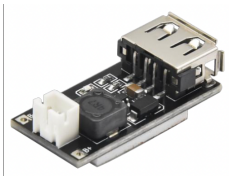
4. **Power Management:**

- **Solar Panel:** The primary source of energy, this panel captures solar energy, which is then converted and stored in batteries.

- **Battery Holder and Lithium Power Module:** The system is powered by solar energy, stored in rechargeable batteries. Additionally, it can be powered via USB when solar energy is insufficient. This dual-mode power supply ensures consistent operation.

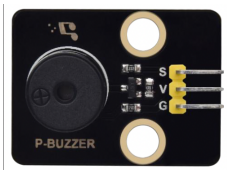


- **Smart Phone Charging Module:** A practical feature that allows the stored solar energy to be used for charging smartphones, demonstrating the system's utility in everyday life.



5. **Additional Components**

- **Piezo Buzzer:** Serves as an auditory alert system, particularly useful for signaling rain detection, prompting immediate protective measures.



- **LCD Display:** A critical interface for real-time data visualization, displaying light intensity, temperature, humidity, and system status messages.

3 Software

The software component of the Keyestudio Sun Follower project is structured into several key functions within an Arduino sketch, designed to control and monitor the various hardware components efficiently.

- **Library Inclusions and Global Definitions:**
  - Libraries for I2C communication (*Wire.h*), LCD interface (*LiquidCrystal_I2C.h*), light measurement (*BH1750.h*), DHT11 sensor (*dht11.h*), and servo control (*Servo.h*) are included.
  - Global definitions and initializations of sensors, servos, pins, and variables are made.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);

#include <BH1750.h>
BH1750 lightMeter;

#include <dht11.h>
dht11 DHT;
#define DHT11_PIN 7

#include <Servo.h>
```

```
Servo lr_servo;//define the name of the servo rotating right and left
Servo ud_servo;//efine the name of the servo rotating upwards and downwards

const byte interruptPin = 2;  //the pin of button;the corruption is disrupted

int RAIN_SENSOR_PIN = A2;
int thresholdValue = 500;

int lr_angle = 90;//set the initial angle to 90 degree
int ud_angle = 10;//set the initial angle to 10 degree;keep the solar panels upright to
     detect the strongest light
int l_state = A0;//define the analog voltage input of the photoresistors
int r_state = A1;
int u_state = A2;
int d_state = A3;
const byte buzzer = 6;  //set the pin of the buzzer to digital pin 6
const byte lr_servopin = 9;//define the control signal pin of the servo rotating right
    and lef
const byte ud_servopin = 10;//define the control signal pin of the servo rotating
    clockwise and anticlockwise

unsigned int light; //save the variable of light intensity
byte error = 15;//Define the error range to prevent vibration
byte m_speed = 10;//set delay time to adjust the speed of servo;the longer the time,
    the smaller the speed
byte resolution = 1;   //set the rotation accuracy of the servo, the minimum rotation
    angle
int temperature;  //save the variable of temperature
int humidity; //save the variable of humidity
```

- ***setup* Function:**
  - Initializes serial communication, sets up I2C for the light sensor, attaches servos to their control pins, and configures sensor and actuator pins.
  - Sets initial positions for the servo motors and initializes the LCD display.

```
void setup() {
  Serial.begin(9600);
  Wire.begin();
  lightMeter.begin();

  lr_servo.attach(lr_servopin);  // set the control pin of servo
  ud_servo.attach(ud_servopin);

  pinMode(l_state, INPUT); //set the mode of pin
  pinMode(r_state, INPUT);
  pinMode(u_state, INPUT);
  pinMode(d_state, INPUT);
  pinMode(RAIN_SENSOR_PIN, INPUT);
  pinMode(buzzer, OUTPUT);

  pinMode(interruptPin, INPUT_PULLUP);  //the button pin is set to input pull-up mode
  attachInterrupt(digitalPinToInterrupt(interruptPin), adjust_resolution, FALLING);
  function ISR

  lcd.init();          // initialize the LCD
  lcd.backlight();     //set LCD backlight

  lr_servo.write(lr_angle);//return to initial angle
  delay(1000);
  ud_servo.write(ud_angle);
  delay(1000);
}
```

- ***loop* Function:**

- The main loop of the program, which perpetually calls other functions to control the system.
- Includes *ServoAction, read_light, read_dht11, read_rain,* and *LcdShowValue* functions.

```
void loop() {
  ServoAction();  //servo performs the action
  read_light();   //read the light intensity of bh1750
  read_dht11();   //read the value of temperature and humidity
  read_rain();    //read the value of rain sensor
  LcdShowValue(); //Lcd shows the values of light intensity, temperature and humidity
}
```

- ***ServoAction* Function:**
  - Reads analog values from the light sensors and adjusts the servo angles to align the solar panel for optimal light exposure.
  - Uses logic to determine if the difference in light intensity between sensors exceeds a defined error threshold, then adjusts the servo angles accordingly.

```
void ServoAction(){
  int L = analogRead(l_state);//read the analog voltage value of the sensor, 0-1023
  int R = analogRead(r_state);
  int U = digitalRead(u_state);
  int D = analogRead(d_state);
  // system adjusting left and right
  //  abs() is the absolute value function
  if (abs(L - R) > error && L > R) { //Determine whether the error is within the
    acceptable range, otherwise adjust the steering gear
    lr_angle -= resolution;//reduce the angle
    if (lr_angle < 0) { //limit the rotation angle of the servo
      lr_angle = 0;
    }
    lr_servo.write(lr_angle);  //output the angle of the servooutput the angle of servo
    delay(m_speed);

  }
  else if (abs(L - R) > error && L < R) { //Determine whether the error is within the
    acceptable range, otherwise adjust the steering gear
    lr_angle += resolution;//increase the angle
    if (lr_angle > 180) { //limit the rotation angle of servo
      lr_angle = 180;
    }
    lr_servo.write(lr_angle);  //output the angle of servo
    delay(m_speed);

  }
  else if (abs(L - R) <= error) { //Determine whether the error is within the
    acceptable range, otherwise adjust the steering gear
    lr_servo.write(lr_angle); //output the angle of servo
  }
  // system adjusting up and down
  if (abs(U - D) > error && U >= D) { //Determine whether the error is within the
    acceptable range, otherwise adjust the steering gear
    ud_angle -= resolution;//reduce the angle
    if (ud_angle < 10) { //limit the rotation angle of servo
      ud_angle = 10;
    }
    ud_servo.write(ud_angle);  //output the angle of servo
    delay(m_speed);

  }
  else if (abs(U - D) > error && U < D) { //Determine whether the error is within the
    acceptable range, otherwise adjust the steering gear
    ud_angle += resolution;//increase the angle
    if (ud_angle > 90) { //limit the rotation angle of servo
      ud_angle = 90;
```

```
    }
    ud_servo.write(ud_angle);  //output the angle of servo
    delay(m_speed);
  }
  else if(abs(U - D) <= error) { //Determine whether the error is within the acceptable
     range. If it is, keep it stable and make no change in angle
    ud_servo.write(ud_angle);  //output the angle of servo
  }
}
```

- ***read_light* Function:**
  - Retrieves light intensity data from the BH1750 sensor and stores it in the *light* variable.

```
void read_light(){
  light = lightMeter.readLightLevel();  //read the light intensity detected by BH1750
}
```

- ***read_dht11* Function:**
  - Reads temperature and humidity data from the DHT11 sensor and handles possible errors in reading.

```
void read_dht11(){
  int chk;
  chk = DHT.read(DHT11_PIN);      // read data
  switch (chk) {
    case DHTLIB_OK:
      break;
    case DHTLIB_ERROR_CHECKSUM:   //check and return error
      break;
    case DHTLIB_ERROR_TIMEOUT:    //Timeout and return error
      break;
    default:
      break;
  }
  temperature = DHT.temperature;
  humidity = DHT.humidity;
}
```

- ***read_rain* Function:**
  - Reads the analog value from the rain sensor, compares it with a threshold, and activates the buzzer if rain is detected.

```
void read_rain(){
  int sensorValue = analogRead(RAIN_SENSOR_PIN);
  if(sensorValue < thresholdValue) {
    soundBuzzer(buzzer); // Sound the buzzer when it's wet
  } else {
    noTone(buzzer); // Ensure the buzzer is off when it's dry
  }
  delay(500); // Delay to limit how often the sensor is read
}
```

- ***soundBuzzer* Function:**
  - Sounds the buzzer in a pattern when activated, typically in response to rain detection.

```
void soundBuzzer(int buzzerPin) {
  // Make the buzzer sound three times
  for (int i = 0; i < 3; i++) {
    tone(buzzerPin, 1000); // Start the buzzer at 1000 Hz
    delay(500); // Sound for 500 milliseconds
```

```
    noTone(buzzerPin); // Stop the buzzer
    if (i < 2) { // If it's not the last iteration, pause before sounding again
      delay(500);
    }
  }
}
```

- ***LcdShowValue* Function:**
  - Updates the LCD display with real-time data, showing light intensity (lux), temperature (°C), humidity (%), and the current resolution setting for the servo adjustment.

```
void LcdShowValue() {
  char str1[5];
  char str2[2];
  char str3[2];
  dtostrf(light, -5, 0, str1); //Format the light value data as a string, left-aligned
  dtostrf(temperature, -2, 0, str2);
  dtostrf(humidity, -2, 0, str3);
  //LCD1602 display
  //display the value of the light intensity
  lcd.setCursor(0, 0);
  lcd.print("Light:");
  lcd.setCursor(6, 0);
  lcd.print(str1);
  lcd.setCursor(11, 0);
  lcd.print("lux");

  //display the value of temperature and humidity
  lcd.setCursor(0, 1);
  lcd.print(temperature);
  lcd.setCursor(2, 1);
  lcd.print("C");
  lcd.setCursor(5, 1);
  lcd.print(humidity);
  lcd.setCursor(7, 1);
  lcd.print("%");

  //show the accuracy of rotation
  lcd.setCursor(11, 1);
  lcd.print("res:");
  lcd.setCursor(15, 1);
  lcd.print(resolution);
}
```

- ***adjust_resolution* Function (Interrupt Service Routine):**
  - Tied to an external button interrupt, this function increases the resolution of the servo movement each time the button is pressed, cycling back to 1 after reaching a maximum value.

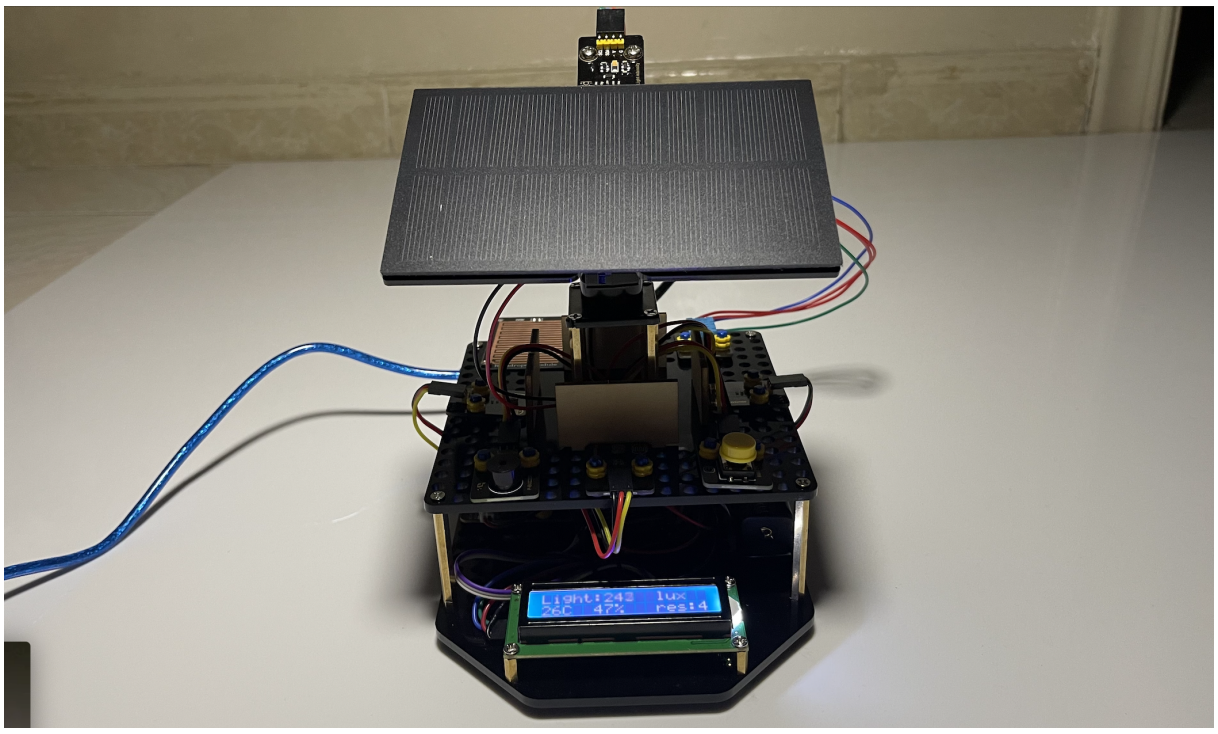
```
void adjust_resolution() {
  tone(buzzer, 800, 100);
  delay(10);  //delay to eliminate vibration
  if (!digitalRead(interruptPin)){
    if(resolution < 5){
      resolution++;
    }else{
      resolution = 1;
    }
  }
}
```

Each function is crafted to handle specific tasks, ensuring that the solar panel is optimally positioned, environmental conditions are monitored, and the user is informed through the LCD display. This modular approach facilitates easy troubleshooting and potential future enhancements.

4 Operation and Process

The development process involved several key stages:

- **Model Construction:** Building the physical structure of the solar tracking system.
- **Wiring and Sensor Integration:** Careful wiring was undertaken, ensuring that each sensor and actuator was correctly connected and functional.
- **Servo Motor Calibration:** The servo motors were calibrated to align with the light sensors, ensuring precise panel positioning for optimal sunlight exposure.
- **Rain Sensor Configuration:** Initially set up as a digital sensor, the rain sensor was reconfigured to analog mode for more reliable moisture detection.
- **Battery Testing:** Rechargeable batteries were rigorously tested to confirm their capacity to charge external devices.
- **Software Development:** Code was iteratively developed and tested, with adjustments made to integrate sensor readings and control the servo motors effectively.



5 Challenges and Solutions

The project faced several challenges:

1. **Limited Analog Pins:** To incorporate the rain sensor, one of the light sensors was sacrificed due to the limited number of available analog pins.
2. **Sensor Configuration:** The rain sensor's initial digital setup was ineffective in wet conditions, necessitating a switch to analog.
3. **Energy Efficiency:** Ensuring that the rechargeable batteries were capable of charging external devices.

6 Related Work

Solar tracking systems are not a new concept and have been explored and implemented in various forms by hobbyists and professionals alike. These systems typically focus on optimizing solar panel orientation to maximize energy capture.

My project aligns with these initiatives in its fundamental aim. However, it distinguishes itself from other similar projects with the integration of a raindrop sensor. This addition significantly enhances the system's functionality by making it responsive to weather changes, particularly rain, which is a unique feature not commonly found in standard solar tracking systems.

7 Conclusion

The Keyestudio Sun Follower project is a testament to the potential of embedded systems in renewable energy applications. The successful integration of sensors, actuators, and renewable energy sources showcases the feasibility of smart, efficient solar tracking systems in real-world scenarios. Future enhancements could focus on increasing the system's scalability and connectivity, further extending its applicability and efficiency.