UNIVERSITY OF BOHOL

College of Engineering, Technology, Architecture, and Fine Arts

DR. CECILIO PUTONG ST., TAGBILARAN CITY

Second Semester

SOLAR LIGHT TRACKER

In Partial Fulfillment of the Requirements for CPEP 322 course

Submitted to:

Engr. Liduvina L. Namocatcat

Instructor

Submitted by:

CAPILI, ROD VINCENT O.

BSCpE-3

May 2025

TABLE OF CONTENTS

FINALS

INTRODUCTION

Assembly language, often regarded as a bridge between high-level programming languages and machine code, plays a crucial role in the field of computer science and engineering. It is a low-level programming language that provides a symbolic representation of a computer's binary instructions, enabling programmers to write code that is closely aligned with the hardware architecture. Unlike high-level languages, which abstract away the details of the underlying machine, assembly language offers precise control over system resources, making it indispensable for tasks that require direct hardware manipulation, optimization, and performance tuning.

The relevance of assembly language extends beyond mere historical significance; it remains fundamental in areas such as embedded systems, operating system development, and performance-critical applications. By understanding assembly language, programmers gain deep insights into how computers execute instructions at the most granular level. This knowledge is essential for debugging complex software issues, developing firmware, and creating efficient code that maximizes the capabilities of the processor. Furthermore, assembly language serves as an educational tool that enriches one's comprehension of computer architecture and the interaction between software and hardware.

The purpose of studying assembly language is multifaceted. Primarily, it equips learners and professionals with the skills to write programs that can directly interface with hardware components, bypassing the layers of abstraction present in higher-level languages. This direct interaction is vital for optimizing system performance and ensuring reliability in critical applications. Additionally, proficiency in assembly language fosters a deeper appreciation for the intricacies of computer operations, encouraging more efficient and effective programming practices. Ultimately, the study of assembly language cultivates a foundational understanding that empowers individuals to innovate and troubleshoot at the core of computing technology.

SOLAR LIGHT TRACKER

PROBLEM REQUIREMENTS

- **Sunlight Detection Accuracy**
  The system must accurately detect the direction of the strongest sunlight using two or more Light Dependent Resistors (LDRs) positioned on the solar panel to measure light intensity differences
- **Servo Motor Control**
  The Arduino must control one or more servo motors to adjust the solar panel's orientation smoothly and precisely, ensuring it continuously faces the sun's position throughout the day
- **Power Efficiency**
  The solar tracker system should operate with minimal power consumption, ideally powered by the Arduino board itself or a low-voltage battery, without requiring an external power source for the servo motor
- **Real-Time Tracking**
  The system must continuously monitor light intensity and adjust the solar panel position in real time to maximize energy absorption from sunrise to sunset
- **Mechanical Range of Motion**
  The servo motor(s) must provide sufficient range of motion to cover the sun's path, typically from east to west (and optionally north to south), to ensure full tracking capability
- **Robustness and Stability**
  The tracker must maintain stable positioning without unnecessary oscillations or jitter when light intensity differences are minimal, using a threshold or error margin to avoid constant small adjustments

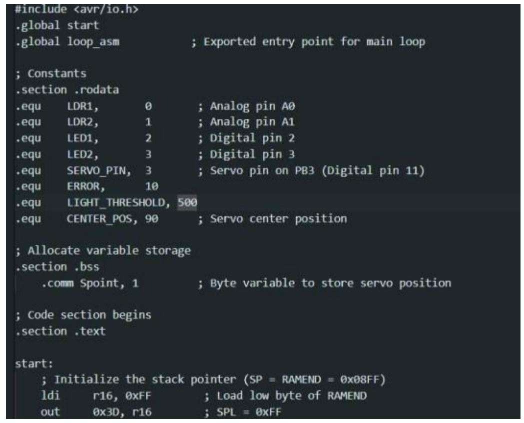
SCOPE AND LIMITATIONS

The scope of the solar light tracker using Arduino IDE encompasses the design and implementation of a single-axis tracking system that automatically adjusts the solar panel's position to maximize sunlight exposure throughout the day. The system utilizes Light Dependent Resistors (LDRs) to detect light intensity and servo motors controlled by an Arduino microcontroller to orient the panel accordingly. This project aims to improve the efficiency of solar energy capture compared to static panels, making it suitable for small to medium-scale solar setups. However, the limitations include its reliance on fair weather conditions, as heavy rain or dust can affect sensor accuracy and mechanical components. Additionally, the prototype typically supports only single-axis movement, restricting its ability to track the sun's elevation changes fully, and it may lack water resistance and robustness for harsh outdoor environments. Future improvements could include expanding to dual-axis tracking, enhancing sensor precision, and incorporating weatherproofing measures to increase durability and performance in diverse conditions.

ANALYSIS

The analysis of the solar light tracker system reveals that leveraging Arduino IDE for programming provides a flexible and accessible platform for controlling sensor inputs and servo motor outputs, making it ideal for prototyping and educational purposes. By using Light Dependent Resistors (LDRs) to measure sunlight intensity, the system can effectively determine the optimal direction for solar panel alignment, which is critical for maximizing energy absorption. However, the single-axis tracking approach, while simpler and cost-effective, limits the system's ability to follow the sun's elevation changes, potentially reducing overall efficiency compared to dual-axis trackers. Furthermore, the mechanical precision of servo motors and the responsiveness of the control algorithm directly impact the system's performance, requiring careful calibration to avoid oscillations or lag in tracking. Environmental factors such as weather conditions and sensor sensitivity also play a significant role in the system's reliability, highlighting the need for robust design considerations. Overall, the analysis underscores the balance between complexity, cost, and performance in developing an effective solar tracking solution using Arduino technology.

DESIGN AND IMPLEMENTATION



```
#include <avr/io.h>
.global start
.global loop_asm            ; Exported entry point for main loop

; Constants
.section .rodata
.equ    LDR1,       0       ; Analog pin A0
.equ    LDR2,       1       ; Analog pin A1
.equ    LED1,       2       ; Digital pin 2
.equ    LED2,       3       ; Digital pin 3
.equ    SERVO_PIN,  3       ; Servo pin on PB3 (Digital pin 11)
.equ    ERROR,      10
.equ    LIGHT_THRESHOLD, 500
.equ    CENTER_POS, 90      ; Servo center position

; Allocate variable storage
.section .bss
    .comm Spoint, 1         ; Byte variable to store servo position

; Code section begins
.section .text

start:
    ; Initialize the stack pointer (SP = RAMEND = 0x08FF)
    ldi     r16, 0xFF        ; Load low byte of RAMEND
    out     0x3D, r16        ; SPL = 0xFF
```

Figure 1: Assembly Code

TESTING AND DEBUGGING

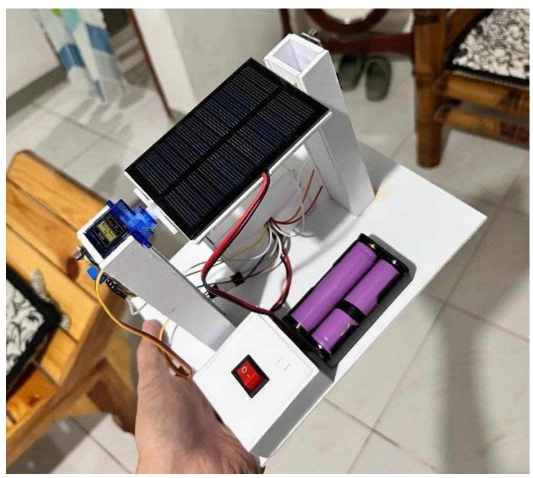


Figure 2: Actual Product

SOURCE CODE

```
extern "C" {
   void start(void);       // your ASM init routine
   void loop_asm(void);   // your ASM loop body (should return so Arduino can call it again)
}

void setup() {
   // call your ASM startup (I/O and timer init)
   start();
}
```

```
void loop() {
   // delegate main work to your ASM loop
   loop_asm();
}

#define __SFR_OFFSET 0          ; Map Special Function Registers starting at I/O address 0
#include <avr/io.h>             ; Include device-specific register definitions

; === Bit Masks ===
.set SERVO_PIN,   0b00000010    ; PB1 (D9/OC1A)  servo PWM output pin
.set LED1,        0b00000100    ; PD2 (D2)  indicator for right-side LDR (LDR1)
.set LED2,        0b00001000    ; PD3 (D3)  indicator for left-side LDR (LDR0)
.set LDR_SENSOR1, 0b00000010    ; PC1 (A1)  right LDR input mask
.set LDR_SENSOR0, 0b00000001    ; PC0 (A0)  left LDR input mask

.section .text
.global main

main:
   ; --- Stack Pointer Setup ---
   ldi   r16, 0xFF            ; Load low byte of stack start (0xFF)
   out   SPL, r16             ; Set SPL ← 0xFF (stack low byte)
   ldi   r16, 0x08            ; Load high byte of stack start (0x08)
   out   SPH, r16             ; Set SPH ← 0x08 (stack high byte)

   ; --- I/O Direction Configuration ---
   ldi   r16, SERVO_PIN       ; Load mask for servo pin
   out   DDRB, r16            ; DDRB ← SERVO_PIN (PB1 as output)
   ldi   r16, LED1 | LED2     ; Load mask for both LEDs
   out   DDRD, r16            ; DDRD ← LED1|LED2 (PD2,PD3 as outputs)
   ldi   r16, 0x00            ; Load zero for input pins
   out   DDRC, r16            ; DDRC ← 0x00 (PC0,PC1 as inputs)

   ; --- Timer1 Fast PWM Mode14, Prescaler=64 ---
   ldi   r16, 0b10000010      ; COM1A1=1 (non-inverting), WGM11=1, WGM10=1
   sts   TCCR1A, r16          ; Configure TCCR1A for fast PWM partial
   ldi   r16, 0b00011011      ; WGM13=1, WGM12=1 (mode14), CS12=1, CS11=1
(prescale64)
   sts   TCCR1B, r16          ; Configure TCCR1B for fast PWM + prescaler
   ldi   r16, 0x13            ; High byte of ICR1 (TOP = 5000)
   sts   ICR1H, r16           ; Set ICR1H ← 0x13
   ldi   r16, 0x88            ; Low byte of ICR1 (0x1388)
   sts   ICR1L, r16           ; Set ICR1L ← 0x88 (5000 ticks → 20ms period)

   ; --- Center Servo to 90° (1.5ms pulse → 375 ticks = 0x0177) ---
   ldi   r16, 0x01            ; High byte of 375
   sts   OCR1AH, r16          ; OCR1AH ← 0x01r
   ldi   r16, 0x77            ; Low byte of 375
   sts   OCR1AL, r16          ; OCR1AL ← 0x77
```

```
    ; --- Initialize LEDs ON (no light detected yet) ---
    ldi   r16, LED1 | LED2      ; Mask for both LEDs
    out   PORTD, r16            ; Turn ON LED1 and LED2

loop:
    ; --- Read both LDR inputs at once ---
    in    r18, PINC             ; r18 ← PINC (PC1..PC0 bits)
    andi  r18, 0b00000011       ; Mask out bits 0 and 1 (LDR_SENSOR0|1)

    ; --- Case 1: both sensors detect light? ---
    cpi   r18, 0b00000011       ; Compare r18 to 0b11
    breq  both_detect           ; If equal, branch to both_detect

    ; --- Case 2: neither sensor detects light? ---
    cpi   r18, 0x00             ; Compare r18 to 0
    breq  none_detect           ; If zero (no bits set), branch to none_detect

    ; --- Case 3: exactly one sensor active; test right (LDR1) first ---
    andi  r18, LDR_SENSOR1      ; Mask PC1 bit only
    brne  ldr1_detect           ; If non-zero, right sensor triggered
    rjmp  ldr0_detect           ; Else, left sensor must be triggered

; ──────────────────────────────────────────────────────────────
; LDR1 only (right) detected → move servo 90°→0°, LED1 OFF, LED2 ON
ldr1_detect:
    cbi   PORTD, 2              ; Clear PD2 → turn OFF LED1
    sbi   PORTD, 3              ; Set PD3 → turn ON LED2
    ; 0° pulse ~150 ticks (0x0096)
    ldi   r16, 0x00             ; High byte of 150 ticks
    sts   OCR1AH, r16           ; OCR1AH ← 0x00
    ldi   r16, 0x96             ; Low byte of 150
    sts   OCR1AL, r16           ; OCR1AL ← 0x96
    rjmp  loop                  ; Return to main loop

; LDR0 only (left) detected → move servo 90°→180°, LED2 OFF, LED1 ON
ldr0_detect:
    cbi   PORTD, 3              ; Clear PD3 → turn OFF LED2
    sbi   PORTD, 2              ; Set PD2 → turn ON LED1
    ; 180° pulse ~600 ticks (0x0258)
    ldi   r16, 0x02             ; High byte of 600
    sts   OCR1AH, r16           ; OCR1AH ← 0x02
    ldi   r16, 0x58             ; Low byte of 600
    sts   OCR1AL, r16           ; OCR1AL ← 0x58
    rjmp  loop                  ; Return to main loop

; Both sensors detect light → move servo to center (90°), both LEDs OFF
both_detect:
    cbi   PORTD, 2              ; Clear PD2 → LED1 OFF
    cbi   PORTD, 3              ; Clear PD3 → LED2 OFF
    ; 90° pulse =  375 ticks (0x0177)
```

```
    ldi   r16, 0x01          ; High byte of 375
    sts   OCR1AH, r16        ; OCR1AH ← 0x01
    ldi   r16, 0x77          ; Low byte of 375
    sts   OCR1AL, r16        ; OCR1AL ← 0x77
    rjmp  loop               ; Return to main loop

; Neither sensor detects light → LEDs ON, servo holds last position
none_detect:
    sbi   PORTD, 2           ; Set PD2 → LED1 ON
    sbi   PORTD, 3           ; Set PD3 → LED2 ON
    rjmp  loop               ; Return to main loop
```

FUTURE DEVELOPMENT

Future development of the solar light tracker can focus on adding dual-axis tracking to improve efficiency by following the sun's elevation and direction. Using more precise sensors and incorporating wireless communication for remote monitoring would enhance performance and usability. Weatherproofing and stronger mechanical parts are also important for durability.

STUDENT INFORMATION

CURRICULUM VITAE

ROD VINCENT O. CAPILI

PUROK 6 - BAYANIHAN, TAGUIHON,

BACLAYON, BOHOL, PHILIPPINES 6301

https://bit.ly/4kq6Efq

"If it is to be, it is up to me."

PROFILE

Date of Birth	:	May 15, 2002
Place of Birth	:	Tagbilaran City, Bohol
Nationality	:	Filipino
Status	:	Single
Gender	:	Male
Contact Number	:	09106724856
Email	:	rvocapili@universityofbohol.edu.ph

EDUCATIONAL BACKGROUND:

Primary	:	Immaculata High School Baclayon, Bohol 2017 - 2018
Tertiary	:	University of Bohol Tagbilaran City, Bohol 2019 - 2020
Tertiary	:	University of Bohol Tagbilaran City, Bohol Bachelor of Science in Computer Engineering 2024 – present

PROJECTS:

Final Project	**:**	Solar Light Tracker
Project Link	**:**	**http://bit.ly/3HqorVE**