

**SUPSI**

# Sistema di controllo per la misura AOI (Angle of Incidence)

---

Studente/i

**Tonani Federico**

Relatore

**Chianese Domenico**

---

Correlatore

**Ceretti Mattia**

---

Committente

**Chianese Domenico**

---

Corso di laurea

**Ingegneria elettronica TP**

Modulo

**Progetto di diploma**

---

Anno

**2023/2024**

---

Data

**30 agosto 2024**

STUDENTSUPSI



# Indice

<b>1 Abstract</b>	<b>1</b>
<b>2 Design di sistema</b>	<b>3</b>
2.1 Consegnna . . . . .	3
2.2 Compiti assegnati . . . . .	3
2.3 Obiettivi . . . . .	4
<b>3 Analisi Teorica</b>	<b>5</b>
3.1 Situazione Iniziale . . . . .	5
3.1.1 Analisi Struttura Meccanica . . . . .	5
3.1.2 Analisi Sistema di Motorizzazione della Struttura . . . . .	6
3.2 Controllo del motore . . . . .	7
3.3 Sistema di finecorsa . . . . .	9
3.4 Sistema di Autocentraggio . . . . .	9
3.5 Interfaccia Grafica . . . . .	9
<b>4 Componenti</b>	<b>11</b>
4.1 Encoder . . . . .	11
4.1.1 Tipologie principali di encoder . . . . .	11
4.1.2 Scelta dell'encoder . . . . .	12
4.1.3 Analisi metodo di utilizzo . . . . .	14
4.2 Modulo Laser . . . . .	16
4.3 Sensore di prossimità induttivo . . . . .	17
4.3.1 Princípio di funzionamento . . . . .	17
4.3.2 Scelta del sensore di prossimità induttivo . . . . .	18
4.3.3 Analisi metodo di utilizzo . . . . .	19
4.4 Motore stepper . . . . .	20
4.5 Microcontrollori . . . . .	21
4.5.1 Arduino Uno . . . . .	21
4.5.2 Esp32 . . . . .	22

<b>5 Implementazione</b>	<b>23</b>
5.1 Sistema di finecorsa . . . . .	24
5.1.1 Codice di funzionamento . . . . .	25
5.2 Sistema di auto-centraggio . . . . .	27
5.2.1 Codice di funzionamento . . . . .	27
5.3 Controllo con feedback del motore . . . . .	29
5.3.1 Posizionamento encoder . . . . .	30
5.3.2 Codice di funzionamento . . . . .	31
5.3.3 Codice di funzionamento esp32 . . . . .	31
5.3.4 Codice di funzionamento arduino uno . . . . .	33
5.4 Indirizzo IP statico . . . . .	39
5.4.1 Codice di funzionamento . . . . .	39
<b>6 Interfaccia Processing</b>	<b>43</b>
6.1 Introduzione allo sviluppo di interafacce . . . . .	43
6.2 Sviluppo Interfaccia . . . . .	43
6.2.1 Parte di background . . . . .	44
6.2.2 Parte di comando . . . . .	45
6.2.2.1 Valori angolari standard . . . . .	45
6.2.2.2 Inserimento valore angolare da tastiera . . . . .	48
6.2.3 Parte di visualizzazione dei dati . . . . .	50
6.2.3.1 Live Angle . . . . .	50
6.2.3.2 Self - Centering System Plot . . . . .	52
<b>7 Circuito Stampato</b>	<b>55</b>
7.1 Componenti PCB . . . . .	55
7.1.0.1 Componenti Microcontrollori . . . . .	55
7.1.0.2 Componenti Sensori . . . . .	56
7.1.0.3 Componenti Circuito di Regolazione di Tensione . . . . .	57
7.2 Schematico . . . . .	58
7.3 PCB . . . . .	59
7.3.1 Top Layer . . . . .	59
7.3.2 Bottom Layer . . . . .	59
7.3.3 Layer Alimentazione . . . . .	60
7.3.4 Layer GND . . . . .	62
<b>8 Test</b>	<b>63</b>
8.1 Misura angolo d'incidenza . . . . .	63
8.1.1 Dettagli Test . . . . .	64
8.1.2 Risultati Ottenuuti e Analisi . . . . .	67

**9 Conclusioni**

**69**



# Elenco delle figure

3.1 Struttura Meccanica . . . . .	6
3.2 Controllo con Feedback . . . . .	8
3.3 Controllo a ciclo chiuso . . . . .	8
3.4 Graphich User Interface . . . . .	9
4.1 Incremental Encoder vs Absolute Encoder . . . . .	12
4.2 Encoder SICK DFS60B-TDPM10000 . . . . .	13
4.3 PIN assignment . . . . .	14
4.4 Modulo Laser . . . . .	16
4.5 Sensore di prossimità induttivo, principio di funzionamento . . . . .	18
4.6 Sensore di prossimità induttivo . . . . .	19
4.7 Pin sensore di prossimità induttivo . . . . .	19
4.8 Motore Stepper Nema 34 x 48 . . . . .	20
4.9 Motore Stepper Nema 34 x 48 Specifiche Tecniche . . . . .	21
4.10 Arduino Uno . . . . .	21
4.11 Esp32 wroom . . . . .	22
5.1 Posizionamento sensore di finecorsa . . . . .	24
5.2 Posizionamento magneti . . . . .	25
5.3 Libreria <TimerOne.h> . . . . .	25
5.4 Flag finecorsa . . . . .	25
5.5 Timer1 interrupt . . . . .	26
5.6 Void checkInduttivo () . . . . .	26
5.7 Void finecorsa () . . . . .	26
5.8 Void finecorsa () . . . . .	26
5.9 Inizializzazione Interrupt . . . . .	27
5.10 Void checkFotoresistenza () . . . . .	27
5.11 Void self_centering () . . . . .	28
5.12 Posizionamento tendicinghia . . . . .	29
5.13 Posizionamento encoder . . . . .	30
5.14 Ricezione comando . . . . .	31

5.15 Ricezione dati via seriale . . . . .	32
5.16 Invio dati fotoresistenza . . . . .	32
5.17 Invio dati encoder . . . . .	33
5.18 Inizializzazione interrupt encoder . . . . .	33
5.19 Void updateEncoder() . . . . .	33
5.20 Lettura seriale . . . . .	34
5.21 Void control_with_feedback() . . . . .	35
5.22 Void sendEncoderValueToESP32() . . . . .	36
5.23 Void sendValueToESP32() . . . . .	36
5.24 Comando input . . . . .	37
5.25 Indirizzo IP dinamico . . . . .	39
5.26 Include Librerie . . . . .	39
5.27 Dichiarazione SSID e Password . . . . .	39
5.28 Configurazione IP statico . . . . .	40
5.29 Configurazione Web Server . . . . .	40
5.30 Void setup () . . . . .	41
 6.1 Interfaccia grafica . . . . .	44
6.2 Void mousePressed () . . . . .	45
6.3 Void sendCommand () . . . . .	46
6.4 Inserimento da tastiera . . . . .	48
6.5 Void sendInputValue() . . . . .	49
6.6 Void getESP32EncoderValue () . . . . .	50
6.7 Thread . . . . .	51
6.8 Void grafico() . . . . .	52
 7.1 Pin LM2596T-5.0/NOPB . . . . .	57
7.2 Schematico . . . . .	58
7.3 Top Layer . . . . .	59
7.4 Bottom Layer . . . . .	60
7.5 Layer Alimentazione . . . . .	60
7.6 Alimentazione 3.3V . . . . .	61
7.7 Alimentazione 5V . . . . .	61
7.8 Alimentazione 24V . . . . .	62
7.9 Layer GND . . . . .	62
 8.1 Centraggi struttura . . . . .	64
8.2 Montaggio Modulo Fotovoltaico . . . . .	65
8.3 Coperta Struttura . . . . .	66
8.4 Grafico Risultati . . . . .	67

# **Elenco delle tabelle**



# Capitolo 1

## Abstract

Il progetto è volto alla realizzazione di una struttura meccanica che permetta di ruotare un modulo fotovoltaico in modo estremamente preciso per poter svolgere il test dell'angolo d'incidenza (AOI). In laboratorio era già presente una struttura non automatizzata che veniva utilizzata per svolgere il test del AOI, il mio compito era di analizzare questa struttura e realizzarne una nuova migliore, automatizzandola in tutte le sue funzioni. Dopo aver definito e concordato il prototipo della nuova struttura meccanica, si è tenuto un incontro con un'azienda ticinese specializzata. Le tempistiche e i costi riportati dall'ingegnere dell'azienda non hanno permesso di continuare il progetto in quella direzione, è stata quindi presa la decisione di riutilizzare la vecchia struttura, migliorandola dove possibile. Su questa struttura sono stati integrati un sistema di finecorsa volto alla sicurezza e uno di auto-centraggio della struttura affinchè, una volta posizionata, la struttura si centrassse con la normale del simulatore solare. Per migliorare la precisione è stato utilizzato un encoder per applicare un controllo a ciclo chiuso con feedback sul motore stepper. La struttura risulta interamente pilotabile tramite interfaccia grafica wireless dedicata, realizzata per facilitare l'interazione dell'operatore con essa. È stato realizzato un circuito stampato contenente tutta l'elettronica necessaria per comunicare con i sensori utilizzati. La struttura è stata testata svolgendo il test della misura dell'angolo d'incidenza. I risultati indicano che la struttura è in grado di svolgere il test per il quale è stata costruita inoltre automatizzandola si è riscontrato una notevole riduzione di tempo dello svolgimento del test.

**Abstract english version**

The project is aimed at creating a mechanical structure that allows a photovoltaic module to be rotated extremely precisely in order to carry out the angle of incidence (AOI) test. In the laboratory there was already a non-automated structure that was used to carry out the AOI test, my task was to analyze this structure and create a new, better one, automating all its functions. After defining and agreeing on the prototype of the new mechanical structure, a meeting was held with a specialized Ticino company. The timing and costs recommended by the company engineer did not allow the project to continue in that direction, so the decision was made to reuse the old structure, improving it where possible. A limit switch system aimed at safety and a self-centering system of the structure have been integrated into this structure so that, once positioned, the structure centers itself with the normal of the solar simulator. To improve accuracy, an encoder was used to apply closed-loop control with feedback on the stepper motor. The structure can be entirely controlled via a dedicated wireless graphic interface, created to facilitate the operator's interaction with it. A printed circuit has been created containing all the electronics necessary to communicate with the sensors used. The structure was tested by carrying out the angle of attack measurement test. The results show that the structure is capable of carrying out the test for which it was built. Furthermore, by automating it, a significant reduction in the time required to carry out the test was found.

## Capitolo 2

# Design di sistema

In questo primo capitolo verrà introdotto il concetto di **AOI 'Angle of Incidence'**, inoltre verrà descritto il progetto in termini di obiettivi e compiti assegnati.

### 2.1. Consegnna

Il settore fotovoltaico è in costante evoluzione tecnologica, accompagnata da aggiornamenti normativi per la certificazione e caratterizzazione dei moduli fotovoltaici. L' "Angle of Incidence" (AOI) rappresenta la caratteristica di un modulo fotovoltaico al variare dell'angolo di incidenza della luce generata da un simulatore solare. La rifrazione e diffusione della luce all'interno del vetro e dell'incapsulante dipendono dalla forma, dimensione e caratteristiche meccaniche del pannello fotovoltaico.

Per effettuare misure precise e ripetibili conformi ai nuovi standard presso il SUPSI PVLab, è necessario modificare una struttura meccanica rotante, in grado di controllare l'angolo di rotazione del modulo con una precisione di 0,01 %/step.

Per ottenere questa precisione la struttura dovrà essere controllata tramite un motore con feedback per impostare con precisione l'angolo di incidenza del modulo fotovoltaico. La struttura dovrà centrarsi automaticamente con il punto centrale del simulatore solare.

Il sistema rotante dovrà essere privo di attrito e sopportare un peso in rotazione di 100kg e fino a 300kg sulla struttura (peso del modulo più peso di 2 persone).

### 2.2. Compiti assegnati

- Realizzazione di una struttura meccanica rotante

- Sviluppo del sistema di controllo motore con feedback sull'angolo e sistema di autocentraggio
- Sviluppo software di controllo
- Assemblare e validare il funzionamento al SUPSI-PVLab
- Test del sistema e misura AOI
- Analisi dei risultati ottenuti

### **2.3. Obiettivi**

- Realizzare la piattaforma rotante con sistema di autocentraggio e feedback sull'angolo
- Realizzare il controllo per il motore con precisione di 0.01 %/step, con encoder e interfaccia software
- Movimento a bassissimo attrito

# Capitolo 3

## Analisi Teorica

In questo capitolo viene descritta la situazione iniziale. In seguito viene fatta un'analisi sullo studio effettuato per adempiere agli obiettivi del progetto.

### 3.1. Situazione Iniziale

La richiesta iniziale del progetto consisteva nel realizzare da zero una struttura meccanica, prendendo spunto da quella preesistente, con lo scopo di automatizzarla il più possibile e rendere il suo funzionamento facile e sicuro. Mettendo in conto di dover realizzare anche la struttura meccanica mi è stato comunicato di aver a disposizione un budget di 5000 franchi per realizzare l'intero progetto.

#### 3.1.1 Analisi Struttura Meccanica

La prima parte del progetto è stata dedicata al pensare come sviluppare la struttura meccanica affinchè fosse in grado di sostenere una massa di 200kg e permettesse alla stessa massa di ruotare con il minimo attrito possibile.

Dopo aver trovato una possibile soluzione concordata con il direttore del PVlab e con il relatore è stato preso un appuntamento con la ditta **Poretti e Gaggini** specializzata in lavorazioni meccaniche. L'appuntamento è stato preso per capire le tempestiche e il costo di realizzazione delle parti meccaniche.

Purtroppo l'esito non è stato quello sperato, di fatti l'ingegnere con il quale si è tenuto il colloquio ci ha comunicato tempi di realizzazione superiori ai due mesi e un costo superiore ai 5000 franchi.

Appresa la notizia, è stato deciso di riutilizzare la struttura meccanica già esistente.

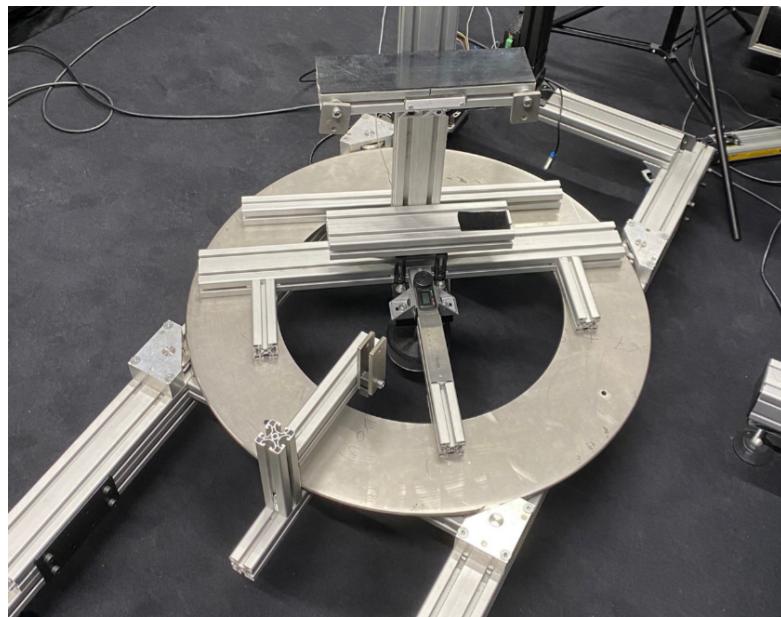


Figura 3.1: Struttura Meccanica

La piastra veniva fatta ruotare tramite un motore stepper posizionato a lato della circonferenza comandato da un arduino che riceveva comandi via seriale. L'angolo veniva misurato tramite un goniometro digitale e la posizione andava corretta manualmente azionando il motore per compiere un certo numero di passi in avanti o indietro.

Dopo aver appreso di dover riutilizzare questa struttura il progetto è stato ridimensionato poichè la struttura presentava delle criticità.

La criticità principale era dovuta dal fatto che la piastra rotante presentava in alcune zone dei "micro-lag", se posizionata in determinati punti la piastra non stava ferma bensì se lasciata libera di muoversi si spostava un po' più avanti o un po' più indietro. Gli spostamenti erano minimi ma dovendo lavorare per ottenere precisioni elevate il problema condizionava il risultato finale.

### 3.1.2 Analisi Sistema di Motorizzazione della Struttura

Il motore stepper presentava una notevole coppia , era in grado di compiere 200 steps al giro, all'albero motore era collegata una puleggia con un rapporto di 1:3, questo rapporto faceva sì che la circonferenza si potesse muovere di 600 steps per rivoluzione.

#### Calcolo

$$\theta_{\text{step}} = \frac{360^\circ}{N_{\text{steps}}} = \frac{360^\circ}{600} = 0.6^\circ$$

Prendendo in analisi il dato di  $\theta_{\text{step}}$  notiamo che il motore in questi condizioni avrebbe potuto effettuare passi non più piccoli di  $0.6^\circ$ . Dovendo cercare di rimanere dentro i  $0.05^\circ$  è stato fondamentale utilizzare la tecnica del microstepping.

### **Microstepping**

Con il termine "micropasso(microstep)" si identifica una tecnica diffusa di pilotaggio dei motori passo passo che consente di posizionare il rotore in una posizione intermedia rispetto al passo intero.

Utilizzando il microstepping il driver connesso al motore permetteva di arrivare a 20000 steps per rivoluzione che moltiplicati per il fattore 3 della puleggia permetteva di arrivare a 60000 steps/giro.

### **Calcolo con microstepping**

$$\theta_{\text{step}} = \frac{360^\circ}{N_{\text{steps}}} = \frac{360^\circ}{60000} = 0.006^\circ$$

Di conseguenza utilizzando il microstepping il nuovo valore di  $\theta_{\text{step}}$  permetteva di rientrare all'interno degli  $0.05^\circ$  per step.

## **3.2. Controllo del motore**

Il motore utilizzato per mettere in moto la struttura è un motore stepper. Su questo motore stepper è stato necessario applicare un controllo con feedback per ridurre l'errore dei movimenti.

### **Controllo con Feedback**

Il controllo con **Feedback** si riferisce all'uso di un segnale di ritorno, o feedback, per confrontare l'output del sistema con il valore desiderato (setpoint) e fare correzioni di conseguenza.

Il **feedback** fornisce informazioni su come si sta comportando il sistema e permette di correggere e limitare l'errore complessivo del sistema.



Figura 3.2: Controllo con Feedback

In particolare in questo progetto verrà applicato un **Controllo a ciclo chiuso** sviluppato su questi punti:

- **1. Feedback Posizionale:** L'encoder misura continuamente la posizione effettiva del motore e fornisce feedback al sistema di controllo. Questo feedback viene confrontato con il setpoint desiderato (la posizione target).
- **2. Correzione degli Errori:** Se c'è una discrepanza tra la posizione attuale del motore (rilevata dall'encoder) e la posizione desiderata, il sistema di controllo legge la differenza e prende "decisioni" per correggere tale errore. Queste decisioni si traducono in azioni correttive, come muovere il motore di un certo numero di passi in avanti o indietro.
- **3. Ciclo continuo:** Questo processo di misurazione, confronto e correzione avviene in un ciclo continuo fino a quando l'errore è ridotto a un livello accettabile (precisione desiderata). Il sistema continua a monitorare e correggere la posizione del motore, adattandosi in tempo reale a eventuali cambiamenti o disturbi.

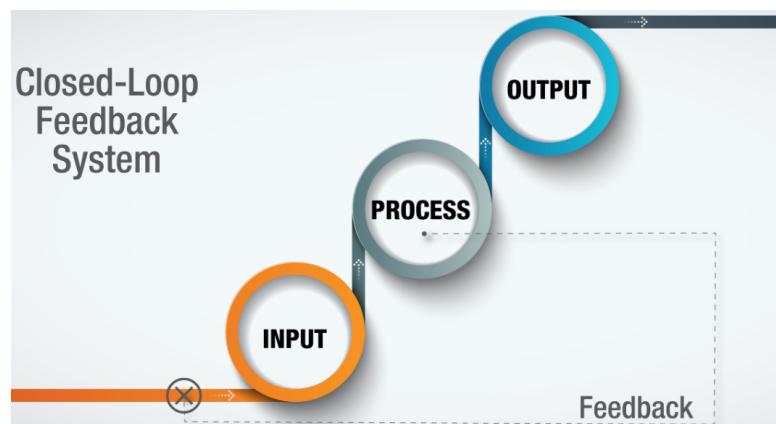


Figura 3.3: Controllo a ciclo chiuso

### 3.3. Sistema di finecorsa

Un'altra richiesta dal progetto era quello di realizzare un sistema di finecorsa. Il sistema di finecorsa doveva essere in grado di fermare istantaneamente il motore se superato un certo angolo.

La struttura meccanica doveva potersi muovere di  $\pm 90^\circ$ , se superati i  $90^\circ$  o i  $-90^\circ$  il motore si sarebbe dovuto arrestare per questioni di sicurezza. Per realizzare questo sistema si è optato per l'utilizzo di un sensore di prossimità che rilevasse il passaggio di un oggetto posizionato in un determinato punto.

### 3.4. Sistema di Autocentraggio

Per centrare la struttura meccanica occorreva un sistema di centraggio. Il sistema di centraggio doveva essere in grado di portare la struttura al angolo  $0^\circ$  rispetto al simulatore solare in modo automatico.

Per realizzarlo è stato pensato di utilizzare un modulo laser con un raggio di luce focalizzato e una LDR(Light Dependent Resistor) di dimensioni ridotte con un diametro di 2.5mm. Il motore ruota la struttura che si ferma nel punto in cui la LDR rileva il massimo valore.

### 3.5. Interfaccia Grafica

Un'interfaccia grafica (GUI, Graphic User Interface) serve a fornire un modo visivo e intuitivo per interagire con un sistema informatico o un'applicazione.

Invece di usare solo comandi testuali o script per comunicare con un software, gli utenti possono interagire attraverso elementi visivi. Ecco le principali funzioni e vantaggi di implementare un'interfaccia grafica:



Figura 3.4: Graphich User Interface

- **Intuitività:** L'interfaccia grafica rende l'applicazione più accessibile e facile da usare. Pulsanti e icone rendono più semplice l'interazione con il sistema.
- **Esperienza Utente Migliore:** L'interfaccia fornisce un feedback immediato all'utente, questo aiuta l'utente a capire ad esempio se un'azione è stata completata con successo o se c'è stato un errore.
- **Efficienza Operativa:** Le interfacce grafiche permettono un accesso rapido e riducono il tempo necessario per eseguire operazioni comuni per un determinato sistema.

## Capitolo 4

# Componenti

In questo capitolo vengono analizzati i componenti principali utilizzati nel progetto analizzando le loro funzionalità e specifiche tecniche.

### 4.1. Encoder

Un encoder è un dispositivo che converte il movimento meccanico in un segnale elettrico digitale o analogico. Viene utilizzato in vari ambiti, tra cui l'automazione industriale, la robotica, e i sistemi di controllo.

#### 4.1.1 Tipologie principali di encoder

##### 1. Encoder Rotativi:

Gli encoder rotativi misurano la posizione angolare, la velocità e la direzione di rotazione di un albero. Possono essere classificati in incrementali e assoluti.

##### 2. Encoder Lineari:

Gli encoder lineari misurano la posizione, la velocità e la direzione di un movimento lineare. Possono anch'essi essere classificati come incrementali o assoluti.

##### Differenze tra encoder assoluto e incrementale

Un'altra distinzione doverosa da fare è quella tra encoder di tipo incrementale e di tipo assoluto. Gli **encoder assoluti** misurano la posizione angolare vera o assoluta. Gli **encoder incrementali** misurano la variazione della posizione angolare.

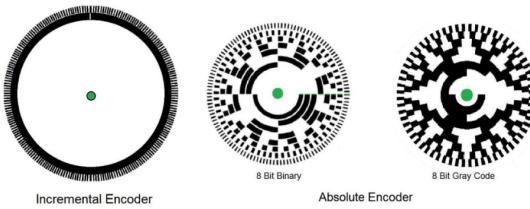


Figura 4.1: Incremental Encoder vs Absolute Encoder

Come possiamo notare dall'immagine sopra, il primo disco rappresenta quello di un **Encoder Incrementale**, esso presenta delle fessure per generare segnali ON / OFF.

Il secondo disco possiede maggiori zone di lettura che forniscono informazioni univoche per ogni posizione. Gli **Encoder Assoluti** sono più complessi ma allo stesso tempo permettono di semplificare le parti di calcolo per le misurazioni delle distanze, delle velocità e delle posizioni. Anche il terzo disco rappresenta un **Encoder Assoluto** che però presenta una codifica binaria differente rispetto al secondo.

#### 4.1.2 Scelta dell'encoder

Per scegliere il tipo di encoder da utilizzare sono stati presi in considerazione diversi aspetti, quali:

- **Costi**
- **Funzioni richieste**
- **Specifiche necessarie**

Analizzando le differenze tra le diverse tipologie di encoder descritte in precedenza, è stato scelto di utilizzare un Encoder Rotativo di tipo Incrementale per le seguenti motivazioni:

- **Costi** → Gli encoder assoluti hanno un costo solitamente più elevato rispetto agli encoder incrementali
- **Funzioni richieste** → L'encoder deve misurare lo spostamento angolare con una certa precisione, essendo presente un sistema di Self-Centering per l'autocentrazione del macchinario non era necessario l'utilizzo di encoder assoluto non dovendo nemmeno misurare la velocità di movimento.
- **Specifiche necessarie** → L'intero sistema doveva presentare un grado di errore contenuto dovendo cercare di stare dentro i  $\pm 0.05^\circ$  rispetto alla posizione reale. Inoltre era presente anche una specifica meccanica da rispettare ovvero la dimensione dell'albero che doveva essere di 10 mm.

La scelta finale è ricaduta sull'encoder incrementale di SICK DFS60B-TDPM10000, ecco di seguito un'immagine dell'encoder e un riassunto delle sue caratteristiche principali:



Figura 4.2: Encoder SICK DFS60B-TDPM10000

### Specifiche Tecniche

- **Risoluzione elevata:** Potendo generare fino a 10.000 impulsi per ogni rotazione completa dell'albero questo encoder offre una risoluzione molto alta.
- **Precisione di misura:** L'encoder è in grado di misurare con un passo di 90° elettrici per ogni impulso, garantendo un controllo accurato della posizione. La deviazione angolare è estremamente ridotta, con un errore massimo di  $\pm 0,05^\circ$  sulla posizione reale.
- **Interfacce di comunicazione:** Compatibile con interfacce come TTL (Transistor-Transistor Logic) e HTL (High Threshold Logic), l'encoder può essere integrato in diversi sistemi di controllo.

Si sarebbe potuto scegliere un encoder ancora più preciso e con un numero maggiore di PPR (Pulses Per Revolution) ma i costi sarebbero stati maggiori e tenendo in considerazione anche il fattore costi si è optato per questa soluzione.

### 4.1.3 Analisi metodo di utilizzo

L'encoder utilizzato presenta 6 canali differenti come possiamo vedere dall'immagine che segue :

PIN Male connec- tor M12, 8-pin	PIN Male connec- tor M23, 12-pin	Wire colors (ca- ble connection)	TTL/HTL signal	Sin/Cos 1.0 V <sub>pp</sub>	Explanation
1	6	Brown	$\bar{A}$	COS-	Signal wire
2	5	White	A	COS+	Signal wire
3	1	Black	$\bar{B}$	SIN-	Signal wire
4	8	Pink	B	SIN+	Signal wire
5	4	Yellow	$\bar{Z}$	$\bar{Z}$	Signal wire
6	3	Purple	Z	Z	Signal wire
7	10	Blue	GND	GND	Ground connection
8	12	Red	+U <sub>S</sub>	+U <sub>S</sub>	Supply voltage
-	9	-	N.c.	N.c.	Not assigned
-	2	-	N.c.	N.c.	Not assigned
-	11	-	N.c.	N.c.	Not assigned
-	7 <sup>1)</sup>	-	O-SET <sup>1)</sup>	N.c.	Set zero pulse <sup>1)</sup>
Screen	Screen	Screen	Screen	Screen	Screen connected to housing on encoder side. Connected to ground on control side.

<sup>1)</sup> For electrical interfaces only: M, U, V, W with O-SET function on PIN 7 on M23 plug. The O-SET input is used to set the zero pulse to the current shaft position. If the O-SET input is applied to US for longer than 250 ms after it has previously been open or applied to GND for at least 1,000 ms, the current shaft position is assigned zero pulse signal "Z".

Figura 4.3: PIN assignment

### Analisi Canali Encoder

- **Canale A:** Canale principale che fornisce impulsi per la misurazione della posizione. I segnali del canale A sono utilizzati per contare gli impulsi e determinare la posizione angolare.
- **Canale B:** Fornisce un segnale in quadratura rispetto al canale A. Questo permette di determinare la direzione di rotazione. La fase del segnale B rispetto a A aiuta a rilevare se l'encoder sta ruotando in senso orario o antiorario.
- **Canale Z:** Fornisce un impulso di riferimento per ogni giro completo dell'encoder (indice o "zero pulse"). Questo segnale indica un punto di riferimento per la posizione angolare, utile per la sincronizzazione e per la determinazione della posizione assoluta relativa.
- **Canali  $\bar{A}, \bar{B}, \bar{Z}$ :** Possono essere utilizzati per funzionalità aggiuntive come la sincronizzazione o per segnali di incremento specializzati, ma non sono essenziali per la misurazione della posizione angolare di base.

Nel nostro caso specifico nel quale il sistema dovrà potersi muovere di massimo  $90^\circ$  in senso orario e in senso anti-orario i canali da utilizzare per rilevare la posizione angolare sono il **Canale A** e il **Canale B**, essi andranno letti tramite Pin digitali del microcontrollore.

## 4.2. Modulo Laser

Per il sistema di auto-centraggio è stato scelto di utilizzare un modulo laser. Il modulo emette una luce altamente focalizzata ad intensità elevata.



Figura 4.4: Modulo Laser

### Specifiche Tecniche

- **Potenza ottica di uscita** → 2.5mW
- **Lunghezza d'onda** → 650nm
- **Classe** → 3R
  - La classe 3R indica che il laser ha una potenza d'uscita limitata a non più di 5 mW.
- **Tensione di alimentazione** → 3Vdc ± 0.2V
- **Assorbimento di corrente** → 7-18mA
- **Vita media** → 3000 ore
- **Dot a 5 metri** → Ø3mm
  - Indica che a 5 metri il punto luminoso prodotto dal laser è di 3mm.
- **Divergenza del raggio** → <1.5mrad
  - É l'angolo con cui il fascio di luce si allarga mentre si propaga nello spazio.

- **Temperatura di funzionamento** → da -10°C a +40°C
- **Dimensioni** → 7mm x 28mm

### 4.3. Sensore di prossimità induttivo

In molte realtà di automazione industriale (AI) è richiesta la capacità di percepire la presenza di un oggetto e/o di una persona senza contatto fisico per motivi di sicurezza e/o logistici. I **sensori di prossimità** sono ideali per svolgere questo tipo di funzioni, sono presenti sul mercato in vari tipi.

#### Tipi di sensori di prossimità

- **Magnetici**
- **Capacitivi**
- **Induttivi**
- **Ottici**

La composizione materiale dell'oggetto è alla base della scelta del tipo di sensore di prossimità da utilizzare. Nel mio caso per rilevare un oggetto metallico è stato scelto di utilizzare un **sensore di prossimità induttivo**.

#### 4.3.1 Principio di funzionamento

I sensori di prossimità induttivi sono in grado di rilevare la presenza di oggetti conduttori. Alla base del funzionamento di questi sensori c'è un campo magnetico ad alta frequenza generato da un circuito di oscillazione avvolto da una spira come mostra l'immagine che segue.

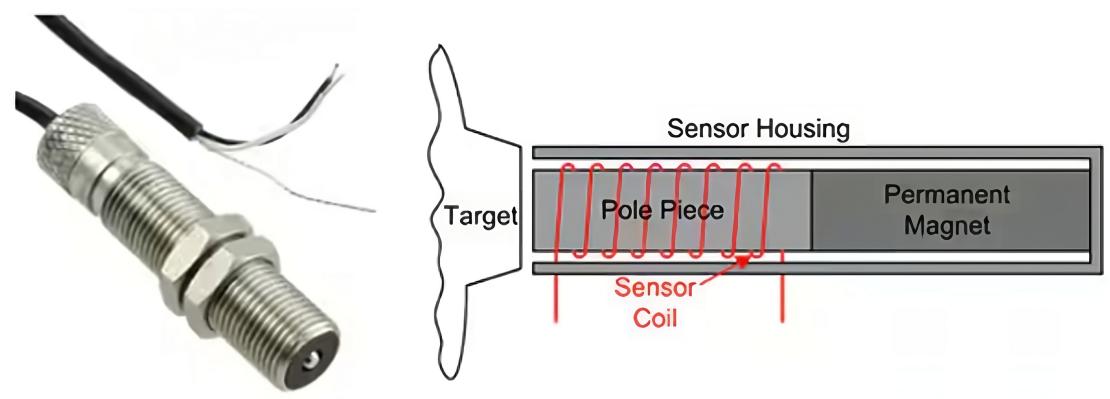


Figura 4.5: Sensore di prossimità induttivo, principio di funzionamento

Un'oggetto metallico, e quindi conduttivo, che si avvicina al campo magnetico ha un'induzione o una corrente parassita indotta, questo riduce notevolmente l'induttanza del sensore induttivo.

In pratica quando l'oggetto si avvicina al sensore, il flusso di corrente induttiva aumenta e questo fa sì che l'oscillazione del circuito oscillante venga attenuata o si ferma completamente.

Questo cambiamento viene rilevato dal sensore che a sua volta emette un segnale di rilevamento.

#### 4.3.2 Scelta del sensore di prossimità induttivo

La scelta è ricaduta sul sensore **Sensore di Prossimità Induttivo LJ12A3- 4-Z BX AZ-Delivery**. Di seguito l'immagine del sensore anteposta all'illustrazione delle sue specifiche principali.



Figura 4.6: Sensore di prossimità induttivo

### Specifiche Tecniche

- **Tensione di esercizio** → 6V - 36V DC
- **Output** → NPN, NO (normalmente aperto)
- **Corrente di uscita** → 300 mA
- **Distanza di rilevamento** → 4mm
- **Materiale rilevato** → Metallo

#### 4.3.3 Analisi metodo di utilizzo

Il sensore preso in analisi come possiamo notare dall'immagine sottostante presenta 3 pin. Due sono i pin di alimentazione mentre il terzo è il pin di output leggibile attraverso l'ingresso analogico del microcontrollore.



Figura 4.7: Pin sensore di prossimità induttivo

Essendo lo stato del sensore normalmente aperto, avremmo sempre un certo livello di tensione misurabile sul pin analogico a seconda dell'alimentazione del sensore, quando l'oggetto metallico sarà a 4mm di distanza la tensione misurata sarà pari a zero a seguito del rilevamento dell'oggetto.

#### 4.4. Motore stepper

I motori stepper, o motori a passo, sono dispositivi elettromeccanici che convertono impulsi elettrici in movimenti meccanici incrementali. A differenza dei motori convenzionali, che ruotano in modo continuo, i motori stepper si muovono in passi discreti. Questa caratteristica li rende estremamente precisi e adatti per applicazioni in cui è necessario un controllo accurato della posizione e della velocità.

Il motore stepper utilizzato era già presente sulla precedente versione della struttura, è stato deciso di riutilizzarlo poiché ancora in ottime condizioni, inoltre il costo di un motore stepper nuovo con le stesse caratteristiche e che quindi permettesse di muovere una struttura di questo peso non era contenuto, quindi la scelta è stata fatta anche per il budget.



Figura 4.8: Motore Stepper Nema 34 x 48

#### Specifiche tecniche Nema st 34x48

Data		ST 34x37			ST 34x48			ST 34x55			ST 34x62
Rated phase current	A	3.00	5.50	8.00	3.00	5.50	8.00	3.00	5.50	8.00	8.00
Phase resistance	Ohm	1.240	0.420	0.200	1.500	0.465	0.215	1.700	0.550	0.290	0.330
Phase inductance	mH	12.00	3.60	1.65	12.96	4.00	1.85	20.00	5.60	2.60	2.88
Holding torque Bipolar	Ncm	520.00	550.00	550.00	700.00	700.00	700.00	1000.00	1000.00	1000.00	1200.00
Detent torque	Ncm	20.00	20.00	20.00	20.00	20.00	20.00	30.00	30.00	30.00	35.00
Rotor inertia	gcm <sup>2</sup>	2860.00	2860.00	2860.00	4732.00	4732.00	4732.00	6018.00	6018.00	6018.00	7030.00
Max. voltage	VDC	160	160	160	160	160	160	160	160	160	160
Weight	Kg	3.000	3.000	3.000	4.000	4.000	4.000	4.600	4.600	4.600	5.400

All data measured with standard cables 500 mm at 25°C

Figura 4.9: Motore Stepper Nema 34 x 48 Specifiche Tecniche

## 4.5. Microcontrollori

I microcontrollori sono componenti essenziali che contengono un microprocessore, memorie e periferiche di input/output. Essi vengono spesso utilizzati per automatizzare, controllare e gestire molte apparecchiature elettroniche. In questo progetto sono stati utilizzati per automatizzare una struttura meccanica rotante usata per ruotare di un dato angolo un modulo fotovoltaico per misurare il variare della sua caratteristica al variare dell'angolo d'incidenza della luce su di esso.

I microcontrollori scelti erano già entrambi disponibili in laboratorio, dopo aver verificato che potessero gestire i sensori da utilizzare è stato deciso di utilizzarli entrambi per il progetto.

### 4.5.1 Arduino Uno

Arduino Uno è una delle schede di sviluppo più popolari al mondo. Monta il microprocessore ATmega328P, dispone di 14 pin input/output digitali, 6 ingressi analogici, un clock da 16MHz e una connessione USB.

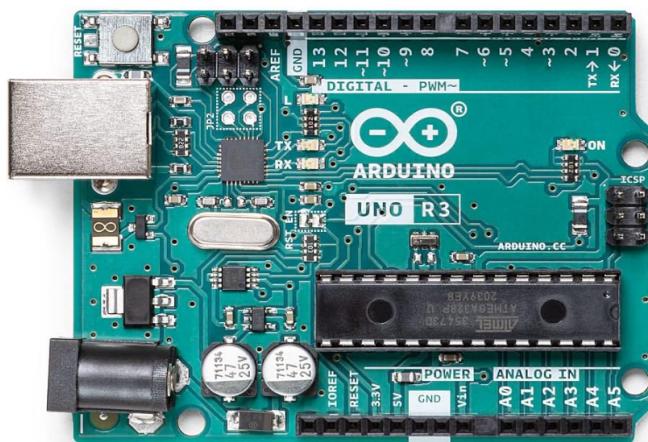


Figura 4.10: Arduino Uno

#### 4.5.2 Esp32

La scheda di sviluppo è dotata di un chip CP2102, sulla scheda è integrato un modulo esp32 wroom che permette di avere sia funzionalità Wi-Fi che Bluetooth. La scheda è molto versatile, supporta tre modalità, AP (Access Point), STA (Station) e la modalità di coesistenza AP + STA. Avendo dovuto integrare un'interfaccia Wi-Fi la sua versatilità di comunicazione si è rilevata molto utile.

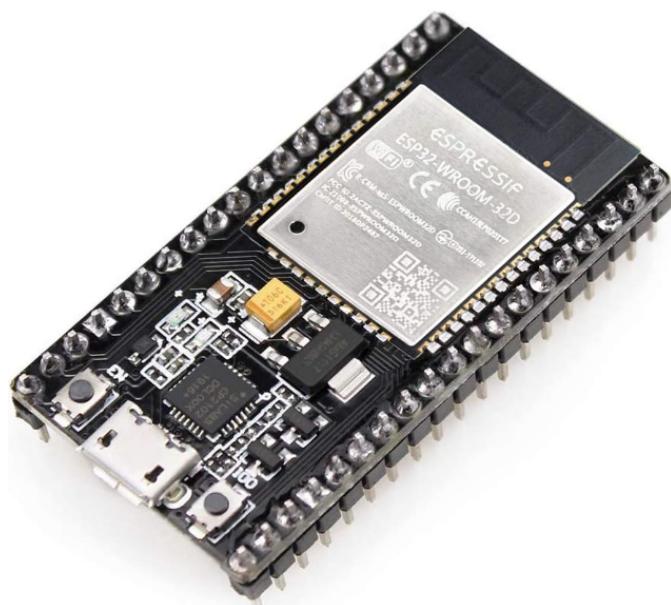


Figura 4.11: Esp32 wroom

## Capitolo 5

# Implementazione

In questo capitolo verranno descritti i processi effettuati per la realizzazione dell'intera struttura.

Prima verrà trattato il sistema di finecorsa, in seguito la realizzazione del sistema di autocentraggio, seguito dall'implementazione della parte di controllo del motore.

Come ultimo punto verrà descritta l'assegnazione di un indirizzo IP statico al microcontrollore.

## 5.1. Sistema di finecorsa

Per realizzare il sistema di finecorsa è stato utilizzato il sensore di prossimità induttivo. Il sensore è stato posizionato al di sotto della piastra rotante.



Figura 5.1: Posizionamento sensore di finecorsa

La piastra deve poter ruotare di  $\pm 90^\circ$ , quindi  $90^\circ$  in senso orario e  $90^\circ$  in senso anti orario. Il sensore rileva la presenza di due magneti che sono stati posti in modo tale che il sensore li rilevi al raggiungimento dei  $95^\circ$  sia in un senso che nell'altro. Da codice, non è permesso alla piastra ruotare oltre i  $90^\circ$  ma come ulteriore sicurezza si è voluto inserire questo sistema di finecorsa.



Figura 5.2: Posizionamento magneti

Una volta rilevata la presenza di un magnete la piastra smetterà immediatamente di ruotare (la luce rossa che si intravede nella foto superiore indica che il sensore di finecorsa ha rilevato il magnete).

### 5.1.1 Codice di funzionamento

1. Come primo step è stata inclusa la libreria <TimerOne.h> per poter utilizzare interrupt software.

```
#include <TimerOne.h>
```

Figura 5.3: Libreria <TimerOne.h>

2. Successivamente è stata dichiarata la variabile **emergencyStopFlag** utilizzata come flag di controllo.

```
//Variabili finecorsa  
bool emergencyStopFlag = false;
```

Figura 5.4: Flag finecorsa

3. Nel void setup è stato inizializzato il **Timer1** inserendo il tempo di campionamento e la funzione da richiamare.

```
// Timer1 per generare un interrupt ogni 100 ms
Timer1.initialize(100000); // 100 ms in microsecondi
Timer1.attachInterrupt(checkInduttivo);
```

Figura 5.5: Timer1 interrupt

4. La funzione **checkInduttivo ()** legge il valore del sensore e lo confronta con una soglia limite che indica il rilevamento del magnete. Una volta rilevato il magnete, il flag **emergencyStopFlag** cambia stato.

```
// Funzione di interrupt software rilevamento finecorsa
void checkInduttivo() {
    if (analogRead(A1) < 350) { // Soglia per finecorsa
        emergencyStopFlag = true;
    }
}
```

Figura 5.6: Void checkInduttivo ()

5. La funzione **finecorsa ()** legge il cambiamento di stato del flag **emergencyStopFlag** e ferma il motore.

```
void finecorsa () {
    if (emergencyStopFlag) {
        motor.moveSteps(0,0);
    }
}
```

Figura 5.7: Void finecorsa ()

6. La funzione **finecorsa ()** viene richiamata in ogni loop di movimento del motore così che se il flag dovesse cambiare di stato verrebbe interrotto il movimento del motore stepper.

```
while (motor.isRunning()) {
    sendValueToESP32(analogRead(A2));
    motor.update();
    finecorsa();
}
```

Figura 5.8: Void finecorsa ()

## 5.2. Sistema di auto-centraggio

Il sistema di auto-centraggio deve centrare la struttura portando il pannello a 0° rispetto al simulatore solare. La distanza tra la struttura e il simulatore solare è di 8 metri, la difficoltà è stata quella di centrare la fotoresistenza in modo tale che il laser posto a 8 metri di distanza la centrasse in posizione 0°. Di seguito le analisi di come è stato implementato il software.

### 5.2.1 Codice di funzionamento

```
// Timer1 per generare un interrupt ogni 100 ms
Timer1.initialize(10000); // 100 ms in microsecondi
Timer1.attachInterrupt(checkFotoresistenza);
self_centering();
```

Figura 5.9: Inizializzazione Interrupt

1. Il primo passo è stato quello di inizializzare l'interrupt timer per leggere il valore analogico della fotoresistenza in modo ciclico ogni 100 ms richiamando la funzione **checkFotoresistenza ()**, dopo aver inizializzato l'interrupt viene richiamata la funzione **self\_centering ()** posta nel **void setup**.

```
void checkFotoresistenza(){
    Value_fotoresistenza = analogRead(A2);
}
```

Figura 5.10: Void checkFotoresistenza ()

2. La funzione **checkFotoresistenza ()** si occupa di aggiornare il valore di **Value\_fotoresistenza** leggendo il pin analogico associato.

```

void self_centering () {
    save_position = 10000;
    motor.setDirection(true);
    motor.moveSteps(10000, 80);
    while (motor.isRunning()) {
        sendValueToESP32(analogRead(A2));
        motor.update();
        finecorsa();
        if(Value_fotoresistenza > Max_value){
            Posizione_centro = lastEncoded;
        }
    }
    control_with_feedback();
    if(Max_value > 600){
        FotoresistenzaFlag = true;
    }

    if(FotoresistenzaFlag){
        steps = 10000 - Posizione_centro;
        motor.setDirection(false);
        motor.moveSteps(steps, 80);
        while (motor.isRunning()) {
            motor.update();
            finecorsa();
        }
        save_position = Posizione_centro;
        control_with_feedback();
    }else{
        motor.setDirection(false);
        motor.moveSteps(10000, 80);
        while (motor.isRunning()) {
            sendValueToESP32(analogRead(A2));
            motor.update();
            finecorsa();
            if(Value_fotoresistenza > Max_value){
                Posizione_centro = lastEncoded;
            }
        }
        save_position = 10000;
        control_with_feedback();
        steps = 10000 - Posizione_centro;
        motor.setDirection(true);
        motor.moveSteps(steps, 80);
        while (motor.isRunning()) {
            sendValueToESP32(analogRead(A2));
            motor.update();
            finecorsa();
        }
        save_position = Posizione_centro;
        control_with_feedback();
    }
}

```

Figura 5.11: Void self\_centering ()

3. La funzione **self\_centering ()** si occupa effettivamente di centrare la struttura. La piastra rotante viene mossa di 10000 steps in un senso, mentre la struttura si muove nel ciclo **while** viene controllato continuamente il valore massimo della fotoresistenza, ogni volta che viene trovato un nuovo valore massimo viene salvata la posizione **Posizione\_centro = lastEncoded**, il processo viene ripetuto in modo analogo dal lato opposto, terminata l'analisi la struttura si porterà al valore 0°.

### 5.3. Controllo con feedback del motore

Il moto del motore viene trasferito alla piastra rotante attraverso una cinghia posta tra due pulegge con lo scopo di incrementare di un fattore 3 il numero di passi per rivoluzione del motore stepper. Per migliorare il trasferimento del moto tra motore e struttura è stato applicato un **tendicinghia**.

#### Tendicinghia

Un tendicinghia è un componente meccanico utilizzato nei sistemi di trasmissione a cinghia per mantenere una tensione adeguata e costante sulla cinghia stessa.

Utilizzare un **tendicinghia** comporta molti vantaggi, in questo progetto è stato utilizzato per ridurre le vibrazioni del sistema mantenendo una tensione della cinghia sempre adeguata e costante.



Figura 5.12: Posizionamento tendicinghia

### 5.3.1 Posizionamento encoder

Il corretto posizionamento dell'encoder è stato essenziale per utilizzarlo al meglio. L'encoder presenta un foro centrale di 10mm nel quale è stato fissato un albero di egual misura, il quale a sua volta è in presa diretta con la piastra circolare che si intende monitorare.

Il posizionamento è risultato molto complicato poichè per leggere il valore corretto angolare della circonferenza è stato necessario riuscire a posizionarlo perfettamente in asse con il centro della circonferenza.

Come possiamo notare dalle immagini che seguono lo spazio di lavoro utile ovvero quello in cui l'encoder è stato fissato era molto limitato.

Per fissarlo in modo rigoroso affinchè non fosse presente alcun tipo di "gioco" è stata fatta una piccola struttura in lega di alluminio che lo tenesse perfettamente in posizione.



Figura 5.13: Posizionamento encoder

### 5.3.2 Codice di funzionamento

L'angolo di movimento della piastra rotante viene selezionato attraverso l'interfaccia sviluppata (analizzata in seguito). Il comando selezionato dall'interfaccia viene letto dall'esp32 tramite web server. Una volta che il comando è stato letto dall'esp32 viene comunicato via seriale all'arduino, l'arduino si occupa in seguito di attivare il motore e muoverlo fino a raggiungere la posizione angolare richiesta. La comunicazione seriale tra esp32 e arduino uno avviene in entrambe le direzioni, l'arduino oltre che a ricevere il comando dall'esp32 invia informazioni sui valori misurati dalla fotoresistenza e dal encoder che poi l'esp32 invierà tramite web server per essere visualizzati sull'interfaccia.

### 5.3.3 Codice di funzionamento esp32

```
server.on("/command1", []() {
    Serial.println("command1");
    server.send(200, "text/plain", "Command 1 received");
});
```

Figura 5.14: Ricezione comando

1. Ogni volta che il server web riceve una richiesta HTTP per il percorso **/command1**, stampa il comando su seriale e invia una risposta HTTP al client che ha fatto la richiesta **server.send(200, "text/plain", "Command 1 received")**.

```

// Controlla se ci sono dati disponibili sulla seriale
if (Serial.available() > 0) {
    // Leggi la stringa ricevuta
    String receivedString = Serial.readStringUntil('\n');
    // Controlla se il messaggio inizia con "VALUE:"
    if (receivedString.startsWith("VALUE:")) { // Valore fotoresistenza
        // Estrai il valore numerico dalla stringa ricevuta
        int value = receivedString.substring(6).toInt(); // 6 è la lunghezza di "VALUE:"
        // Salva il valore ricevuto nella variabile globale
        esp32Value = value;
        Serial.print("Received value from Arduino Uno: ");
        Serial.println(value);
    }
    /*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*STUDENTSUPSI
```

Figura 5.15: Ricezione dati via seriale

- 2.** Questa parte di codice si occupa di monitorare i messaggi inviati da arduino via seriale. Per distinguere se i dati inviati appartengono alla fotoresistenza o all'encoder il dato è preceduto dalla scritta **VALUE:** per la fotoresistenza e da **VALUE EN:** per l'encoder. Il dato viene estratto dalla stringa eliminando la parte utilizzata per distinguerli **int value\_en = receivedString.substring(9).toInt();**

```

server.on("/getValue", []() { // Valore della fotoresistenza
    String response = "VALUE: " + String(esp32Value);
    server.send(200, "text/plain", response);
});
```

Figura 5.16: Invio dati fotoresistenza

- 3.** Questo blocco di codice si occupa di gestire una richiesta HTTP al percorso **/getValue**. Una volta accertata la richiesta viene generata una stringa che contiene il valore della fotoresistenza **String response = "VALUE: " + String(esp32Value)**. Una volta generata la stringa viene inviata come risposta HTTP al client **server.send(200, "text/plain", response)**.

```
server.on("/getValueEncoder", []() { // Valore dell'encoder
    String response = "VALUE EN: " + String(esp32EncoderValue);
    server.send(200, "text/plain", response);
});
```

Figura 5.17: Invio dati encoder

4. L'invio dei dati dell'encoder viene gestito in maniera analoga a quello della fotoresistenza.

#### 5.3.4 Codice di funzionamento arduino uno

```
// Inizializzazione dei pin come input (fasi A e B dell'encoder)
pinMode(pinA, INPUT);
pinMode(pinB, INPUT);

// Abilitazione degli interrupt sui pin A e B
attachInterrupt(digitalPinToInterrupt(pinA), updateEncoder, CHANGE);
attachInterrupt(digitalPinToInterrupt(pinB), updateEncoder, CHANGE);
```

Figura 5.18: Inizializzazione interrupt encoder

1. In questa parte di codice vengono inizializzati i due pin utilizzati come input per leggere la fase A e la fase B dell'encoder.

Successivamente vengono abilitati gli interrupt sui pin delle due fasi richiamando ogni volta la funzione **updateEncoder**.

```
void updateEncoder() {
    // Lettura dei segnali A e B
    int MSB = digitalRead(pinA); // Most significant bit
    int LSB = digitalRead(pinB); // Least significant bit

    // Combinazione dei segnali A e B in una singola variabile
    int encoded = (MSB << 1) | LSB;
    int sum = (lastEncoded << 2) | encoded;

    // Determinazione della direzione di rotazione e aggiornamento del conteggio
    if (sum == 0b0010 || sum == 0b0100 || sum == 0b1101 || sum == 0b1011) encoderCount++;
    if (sum == 0b0001 || sum == 0b0111 || sum == 0b1110 || sum == 0b1000) encoderCount--;

    // Memorizzazione dell'ultimo valore codificato
    lastEncoded = encoded;
}
```

Figura 5.19: Void updateEncoder()

2. La funzione **updateEncoder** si occupa di leggere i segnali A e B, il segnale A viene considerato come il **MSB (most significant bit)** mentre il segnale B come **LSB (last significant bit)**.

In seguito i segnali vengono combinati in una singola variabile per poi determinare la direzione di rotazione e l'incremento del conteggio.

```

if (Serial.available()) {
    String command = Serial.readStringUntil('\n');
    command.trim(); // Rimuove spazi bianchi iniziali e finali, incluso '\n'
    Serial.println("Command received: " + command); // Stampa il comando ricevuto

    if (command.equals("command1")) { // Comando per 10°
        if(save_position < 1667){
            steps = 1667 - save_position;
            motor.setDirection(true);
            motor.moveSteps(steps, 100);
            while (motor.isRunning()) {
                sendEncoderValueToESP32(encoderCount);
                motor.update();
                finecorsa();
            }
            save_position = 1667;
            control_with_feedback();
        }
        if(save_position > 1667){
            steps = save_position - 1667;
            motor.setDirection(false);
            motor.moveSteps(steps, 100);
            while (motor.isRunning()) {
                sendEncoderValueToESP32(encoderCount);
                motor.update();
                finecorsa();
            }
            save_position = 1667;
            control_with_feedback();
        }
    } else if (command.equals("command2")) { // Comando per 20°
}
}

```

Figura 5.20: Lettura seriale

**3.** Questo blocco di codice si occupa di monitorare i messaggi sulla seriale, se è presente una nuova stringa essa viene letta. Se si tratta di un comando, il comando viene stampato e confrontato con quelli presenti nel ciclo if/else if. Ci sono 2 tipi di comandi, al primo tipo di comando è associato un angolo fisso, in questo caso a **command1** è associata la posizione 10°. Il secondo tipo verrà analizzato in seguito e permette di inserire in input un angolo qualsiasi.

Una volta entrato nel ciclo come prima cosa viene verificato se la struttura per raggiungere quel determinato angolo dovrà muoversi in avanti o indietro, questo viene fatto confrontando la posizione attuale, che corrisponde alla variabile **save\_position**, con quella in cui si vuole muoversi. Una volta verificato il tipo di movimento necessario viene settata la direzione del motore (**motor.setDirection(true)**) e il numero di steps da fare (**motor.moveSteps(steps, 100)**). Il ciclo **while** serve per far muovere il motore aggiornando i passi di volta in volta, inoltre viene mandata ciclicamente la posizione letta dall'encoder **sendEncoderValueToESP32(encoderCount)** per visualizzare l'angolo in continuo aggiornamento sull'interfaccia, l'ultima funzione presente nel ciclo while **finecorsa()** è già stata analizzata.

Una volta terminato il movimento la posizione attuale viene aggiornata e salvata. La posi-

zione viene corretta ad ogni movimento attraverso la funzione **control\_with\_feedback()**.

```
void control_with_feedback(){
    do{
        // noInterrupts();
        currentPosition = encoderCount;
        // interrupts();
        save_position_encoder = round(currentPosition * 1.5);
        Serial.print("sve : ");
        Serial.println(save_position_encoder);
        if(save_position_encoder > save_position){
            steps = save_position_encoder - save_position;
            motor.setDirection(false);
            motor.moveSteps(steps, 5);
            while (motor.isRunning()) {
                sendEncoderValueToESP32(encoderCount);
                motor.update();
            }
        }
        if(save_position_encoder < save_position){
            steps = save_position - save_position_encoder;
            motor.setDirection(true);
            motor.moveSteps(steps, 5);
            while (motor.isRunning()) {
                sendEncoderValueToESP32(encoderCount);
                motor.update();
            }
        }
        //noInterrupts();
        currentPosition = encoderCount;
        // interrupts();
        save_position_encoder = round(currentPosition * 1.5);
    }while(abs(save_position_encoder - save_position) > 1 );
}
```

Figura 5.21: Void control\_with\_feedback()

4. La funzione **control\_with\_feedback()** permette di correggere la posizione della struttura. Per prima cosa viene letta la posizione dell'encoder, in seguito la posizione letta viene confrontata con quella nella quale ci si vuole spostare, se ci sono delle discrepanze vengono rilevate e verranno corrette in modo ciclico finché non soddisfano la condizione del ciclo **do while** ovvero **while(abs(save\_position\_encoder - save\_position) > 1 )**.

```
void sendEncoderValueToESP32(int value_en) {
    unsigned long currentMillis = millis(); // Leggi il tempo attuale
    // Controlla se è trascorso l'intervallo desiderato
    if (currentMillis - encoderpreviousMillis >= interval) {
        encoderpreviousMillis = currentMillis; // Salva il tempo dell'ultimo aggiornamento
        Serial.print("VALUE EN:");
        Serial.println(value_en);
    }
}
```

Figura 5.22: Void sendEncoderValueToESP32()

5. La funzione **sendEncoderValueToESP32()** si occupa di mandare via seriale il valore letto dall'encoder all'esp32, il quale tramite web server lo inoltrerà all'interfaccia per visualizzare l'angolo aggiornato. La funzione verifica che sia passato un certo intervallo di tempo **interval** per poi stampare il valore sulla seriale.

```
void sendValueToESP32(int value) {
    unsigned long currentMillis = millis(); // Leggi il tempo attuale
    // Controlla se è trascorso l'intervallo desiderato
    if (currentMillis - previousMillis >= interval) {
        previousMillis = currentMillis; // Salva il tempo dell'ultimo aggiornamento
        Serial.print("VALUE:");
        Serial.println(value);
    }
}
```

Figura 5.23: Void sendValueToESP32()

6. La funzione **sendValueToESP32()** funziona in modo analogo a quella mostrata in precedenza inviando però il valore della fotoresistenza.

```

} else if (command.startsWith("Received degrees: ")) { // Comando per ricevere valore angolare qualsiasi
    String degreeValue = command.substring(18); // Ottiene il valore dei gradi dopo "Received degrees: "
    degreeValue.trim(); // Rimuove eventuali spazi bianchi
    degreeValue.replace(',', '.'); // Sostituisce la virgola con il punto decimale
    Serial.print("Extracted degree value: ");
    Serial.println(degreeValue);

    floatDegrees = degreeValue.toFloat();
    Serial.print("Degrees received and stored: ");
    Serial.println(floatDegrees);

    if(floatDegrees >= 0){

        // Calcola i passi necessari per raggiungere il grado ricevuto
        new_steps = (60000 * floatDegrees) / 360;
        roundedDegrees = round(new_steps);
        Serial.print("Calculated steps (rounded degrees): ");
        Serial.println(roundedDegrees);

        if((save_position < roundedDegrees) && (roundedDegrees <= 15000) && (senseFlag == false)){
            Serial.println("Entering the 'Received degrees' block"); // Debug
            steps = roundedDegrees - save_position;
            motor.setDirection(true);
            motor.moveSteps(steps, 80);
            while (motor.isRunning()) {
                sendEncoderValueToESP32(encoderCount);
                motor.update();
                finecorsa();
            }
            save_position = roundedDegrees;
            control_with_feedback();
        } else if((save_position > roundedDegrees) && (roundedDegrees <= 15000) && (senseFlag == false)){
            steps = save_position - roundedDegrees;
            motor.setDirection(false);
            motor.moveSteps(steps, 80);
            while (motor.isRunning()) {
                sendEncoderValueToESP32(encoderCount);
                motor.update();
                finecorsa();
            }
            save_position = roundedDegrees;
            control_with_feedback();
        } else if((senseFlag) && (roundedDegrees == 0)){
            steps = save_position - roundedDegrees;
            motor.setDirection(true);
            motor.moveSteps(steps, 80);
            while (motor.isRunning()) {
                sendEncoderValueToESP32(encoderCount);
                motor.update();
                finecorsa();
            }
            save_position = roundedDegrees;
            control_with_feedback_sx();
        } else {
            Serial.println("Invalid steps range or already at position");
        }
        senseFlag = false;
    }
}

```

Figura 5.24: Comando input

7. Questo è il secondo tipo di comando introdotto in precedenza. Tramite l'interfaccia viene inviato un angolo qualsiasi in input digitato da tastiera. L'angolo viene letto dalla stringa eliminando i caratteri superflui.

Una volta isolato l'angolo viene verificato se l'angolo inserito è positivo (**floatDegrees >= 0**) o negativo (**floatDegrees < 0**). Le logiche di funzionamento sono molto simili a quelle analizzate in precedenza in **command1**.



## 5.4. Indirizzo IP statico

Durante lo sviluppo dell'interfaccia per pilotare la struttura è stato deciso di assegnare al microcontrollore esp32 un indirizzo IP statico. È stato fatto per semplificare l'uso degli utenti futuri poiché senza utilizzare un IP statico ad ogni nuovo accesso alla rete l'esp32 avrebbe utilizzato un indirizzo IP dinamico assegnato automaticamente dal server DHCP (Dynamic Host Configuration Protocol) del router. L'indirizzo IP dinamico sarebbe dovuto essere stato letto e sostituito ogni volta all'interno del codice dell'interfaccia per comunicare correttamente.

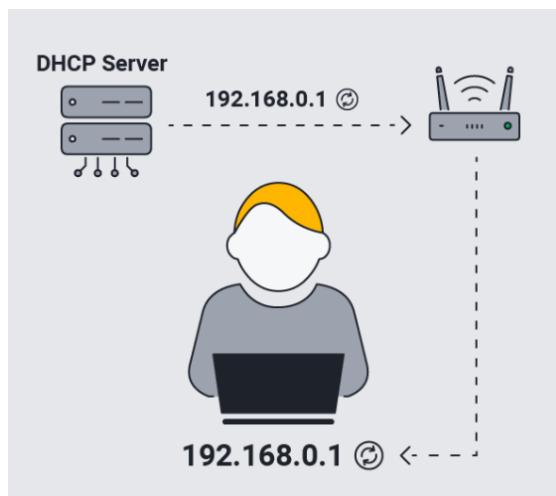


Figura 5.25: Indirizzo IP dinamico

### 5.4.1 Codice di funzionamento

- Le librerie `<WiFi.h>` e `<WebServer.h>` vengono incluse nel progetto per gestire le funzionalità di Wi-Fi e di web server, potendo così gestire richieste HTTP.

```
#include <WiFi.h>
#include <WebServer.h>
```

Figura 5.26: Include Librerie

- Vengono dichiarate le costanti **ssid** e **password** per salvare il nome e la password della rete Wi-Fi alla quale collegarsi.

```
const char* ssid = "Hotspot-AOI";
const char* password = "12345678";
```

Figura 5.27: Dichiarazione SSID e Password

## 3.

**IPAddress local\_IP(192, 168, 137, 219)** specifica l'indirizzo IP statico che l'esp32 utilizzerà nella rete.

**IPAddress gateway(10, 14, 96, 1)** Indica l'indirizzo IP del gateway (router) utilizzato per accedere ad altre reti.

**IPAddress subnet(255, 255, 240, 0)** Definisce la subnet mask, utilizzata per determinare la porzione di rete dell'indirizzo IP.

```
IPAddress local_IP(192, 168, 137, 219);
IPAddress gateway(10, 14, 96, 1);
IPAddress subnet(255, 255, 240, 0);
```

Figura 5.28: Configurazione IP statico

4. Viene creata un'istanza **WebServer server(80)** che "ascolta" sulla porta 80, la porta standard per il traffico HTTP.

```
WebServer server(80);
```

Figura 5.29: Configurazione Web Server

## 5.

**if (!WiFi.config(local\_IP, gateway, subnet))** Configura l'esp32 per utilizzare un indirizzo IP statico specificando l'IP locale, il gateway e la subnet mask. Se la configurazione fallisce viene stampato un messaggio d'errore.

**WiFi.begin(ssid, password)** Avvia la connessione alla rete Wi-Fi utilizzando l'SSID e la password forniti.

**while (WiFi.status() != WL\_CONNECTED) ...** Questo ciclo while continua a ripetere il messaggio "Connecting to WiFi..." finchè l'esp32 non si connette alla rete Wi-Fi.

**Serial.println("Connected to WiFi")** Una volta connesso viene stampato un messaggio di conferma sul serial monitor.

**Serial.println(WiFi.localIP())** Stampa l'indirizzo IP locale assegnato all'esp32. L'indirizzo IP risulterà sempre lo stesso a meno che fallisca la configurazione.

```
if (!WiFi.config(local_IP, gateway, subnet)) {  
    Serial.println("STA Failed to configure");  
}  
  
WiFi.begin(ssid, password);  
  
while (WiFi.status() != WL_CONNECTED) {  
    delay(1000);  
    Serial.println("Connecting to WiFi...");  
}  
Serial.println("Connected to WiFi");  
Serial.println(WiFi.localIP());
```

Figura 5.30: Void setup ()



## Capitolo 6

# Interfaccia Processing

In questo capitolo sarà spiegato il processo di sviluppo dell'interfaccia per l'utente (UI) realizzata tramite Processing e verranno analizzate le parti di codice fondamentali.

### 6.1. Introduzione allo sviluppo di interafacce

L'interfaccia utente (UI) è un elemento fondamentale nel design e nello sviluppo del software di un prodotto poichè rappresenta il punto di interazione tra l'utente e il sistema.

#### Processing

L'interfaccia è stata realizzata interamente con **Processing**. La decisione di sviluppare un'interfaccia in **Processing** deriva dalla necessità di gestire in modo efficace il flusso di dati attraverso strumenti visivi che permettono di semplificare alcuni tipi di operazioni che altrimenti risulterebbero per l'utente più complesse.

L'interfaccia permette di semplificare i passaggi che l'utente deve svolgere per visualizzare un determinato tipo di dato o per "lanciare" un comando del sistema.

In particolare l'interfaccia realizzata permette interazioni con un esp32 tramite web-server che a sua volta comunicherà all'arduino il comando da eseguire e viceversa, ovvero l'arduino comunicherà dati all'esp32 che poi l'utente potrà visualizzare sull'interfaccia.

### 6.2. Sviluppo Interfaccia

Di seguito un'immagine dell'interfaccia sviluppata della quale, in questa sezione, verrà spiegato il processo di realizzazione.

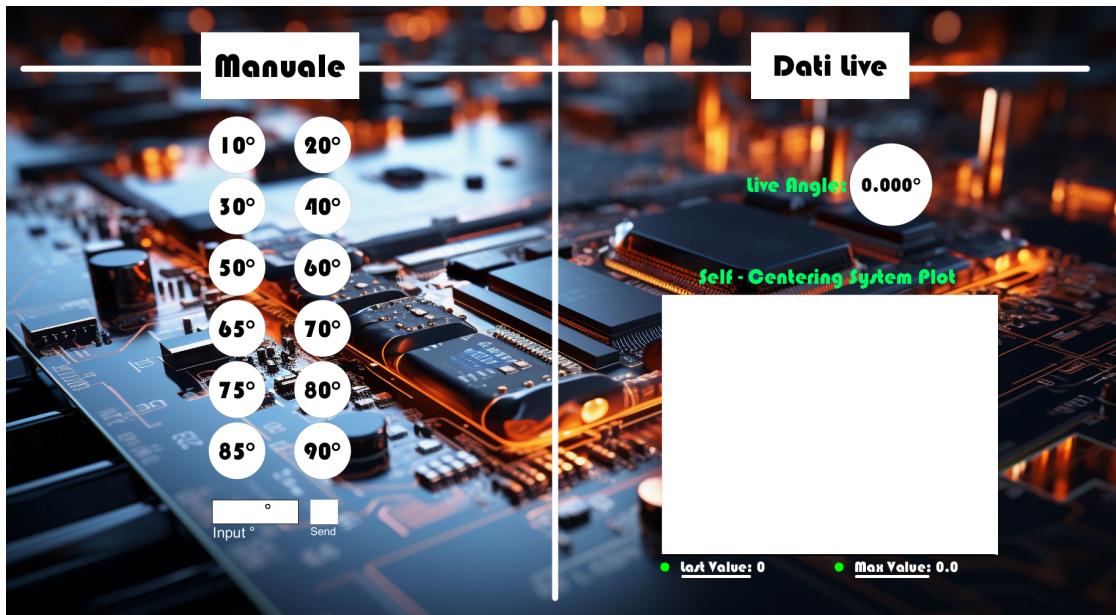


Figura 6.1: Interfaccia grafica

L'immagine dell'interfaccia può essere scomposta in 3 macro parti: **Parte di background**, **Parte di comando**, **Parte di visualizzazione dei dati**.

### 6.2.1 Parte di background

Nella parte di background è stata costruita la struttura dell'interfaccia pensando alle funzioni disponibili per l'utente, a un'area di visualizzazione dati e al contesto del progetto da sviluppare.

Analizzare il codice di questa parte risulta complesso, poiché sparso nelle 420 righe di codice scritto, di conseguenza verrà solamente descritto quanto fatto e la struttura base.

- **1. Immagine di sfondo** Trattandosi di un progetto di diploma di ingegneria elettronica come immagine di sfondo è stata scelta un'immagine che rappresenta un circuito elettronico, un po' accattivante.
- **2. Divisione degli spazi** L'interfaccia è stata suddivisa in due macro-aree chiamate '**Manuale**' e '**Dati Live**'. All'interno di '**Manuale**' sono presenti tutti i comandi attuabili dall'utente mentre all'interno di '**Dati Live**', come dice il nome, sono visualizzabili i dati live del sistema.
- **3. Parte grafica** All'interno dell'interfaccia sono presenti molti tratti di linee che vanno a contornare parole, dividere fisicamente ambienti, delimitare forme. Una parte quindi del lavoro è stata quella di suddividere e delimitare il tutto cercando di renderlo piacevole alla vista e allo stesso tempo il più funzionale possibile.

## 6.2.2 Parte di comando

Nella parte di comando è possibile selezionare dei valori angolari standard che l'utente utilizza con maggior frequenza, è presente anche l'opzione d'inserimento tramite tastiera di un valore angolare qualsiasi anche non intero.

### 6.2.2.1 Valori angolari standard

I valori angolari standard vanno da 10° fino a 60° in multipli di 10° e da 60° a 90° in multipli di 5°. I cerchi raffiguranti gli angoli predefiniti se selezionati dal mouse inviano una stringa univoca di un comando al web-server.

Questo viene fatto tramite la funzione **mousePressed()** e la funzione **sendCommand()** di seguito rappresentate:

```
void mousePressed() {
    float raggio = 85 / 2.0;
    String inputText = cp5.getText(Textfield.class, "Input").getText();

    if (dist(mouseX, mouseY, 380, 210) < raggio) {
        sendCommand("command1");
    } else if (dist(mouseX, mouseY, 520, 210) < raggio) {
        sendCommand("command2");
    } else if (dist(mouseX, mouseY, 380, 310) < raggio) {
        sendCommand("command3");
    } else if (dist(mouseX, mouseY, 520, 310) < raggio) {
        sendCommand("command4");
    } else if (dist(mouseX, mouseY, 380, 410) < raggio) {
        sendCommand("command5");
    } else if (dist(mouseX, mouseY, 520, 410) < raggio) {
        sendCommand("command6");
    } else if (dist(mouseX, mouseY, 380, 510) < raggio) {
        sendCommand("command7");
    } else if (dist(mouseX, mouseY, 520, 510) < raggio) {
        sendCommand("command8");
    } else if (dist(mouseX, mouseY, 380, 610) < raggio) {
        sendCommand("command9");
    } else if (dist(mouseX, mouseY, 520, 610) < raggio) {
        sendCommand("command10");
    } else if (dist(mouseX, mouseY, 380, 710) < raggio) {
        sendCommand("command11");
    } else if (dist(mouseX, mouseY, 520, 710) < raggio) {
        sendCommand("command12");
    }
}
```

Figura 6.2: Void mousePressed ()

```
void sendCommand(String command) {
    sendCommand(command, "");
}

void sendCommand(String command, String degrees) {
    String server = "192.168.137.219";
    String url = "http://" + server + "/" + command;
    if (!degrees.equals("")) {
        url += "?degrees=" + degrees;
    }
    try {
        URL serverUrl = new URL(url);
        HttpURLConnection connection = (HttpURLConnection) serverUrl.openConnection();
        connection.setRequestMethod("GET");
        connection.connect();

        BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            println(inputLine);
        }
        in.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figura 6.3: Void sendCommand ()

### Analisi funzioni `mousePressed()` e `sendCommand()`

- **float raggio = 85 / 2.0;** Con questo comando viene calcolato il raggio della circonferenza
- **String inputText = cp5.get(Textfield.class, "Input").getText();** Recupera il testo inserito da tastiera nella apposita casella utilizzando la libreria ControlP5, il testo verrà utilizzato in seguito e non in questa funzione.
- **dist(mouseX, mouseY, x, y)** La funzione dist permette di calcolare la distanza tra due punti in particolare in questo caso verifica che il puntatore del mouse venga premuto all'interno dell'area di uno dei cerchi presenti.
- **sendCommand("commandX")** Se verificato che il mouse è stato premuto all'interno di uno dei cerchi, allora viene chiamata la funzione sendCommand (string command) per inviare un comando specifico "commandX". Questa funzione ne invoca un'altra omonima ma con parametri differenti passandole il comando e una stringa vuota.
- **sendCommand(String command, String degrees)** La funzione ha il compito di "assemblare" un URL basato sul comando ricevuto e sul parametro degrees se non è associato ad una stringa vuota. Il parametro degrees risulta una stringa vuota sempre, tranne se il valore angolare viene inserito da tastiera nell'apposita casella. L'URL necessita anche dell'indirizzo IP al quale inviare i dati, viene creato un oggetto URL e viene aperta una connessione HTTP al server specificato, in seguito viene letta la risposta del server e stampata. Al verificarsi di un errore, la funzione sendCommand () prevede anche una parte di debug che segnala l'errore e lo stampa.

### Analisi Overloading (`sendCommand`)

In Java ( Processing è basato su Java ), è possibile avere due o più funzioni con lo stesso nome. Questo concetto è noto come **overloading** che consente di definire più versioni della stessa funzione purchè differiscano per:

- **Numero di parametri**
- **Tipo di parametri**
- **Ordine dei parametri**

In questo caso il numero dei parametri delle due funzioni è diverso e quindi l'overloading è possibile.

L'overloading è stato utilizzato per fornire quanta più flessibilità possibile alla funzione `sendCommand`, di fatti, nella prima versione `void sendCommand(String command)` la funzione può essere chiamata quando non è necessario specificare il parametro '`degrees`', essa invocherà la versione completa della funzione con una stringa vuota come secondo parametro. Se invece il parametro '`degrees`' non risulta nullo, verrà usata la seconda versione della funzione `void sendCommand(String command, String degrees)`.

### 6.2.2.2 Inserimento valore angolare da tastiera

Nell'interfaccia è presente un apposita casella che permette l'inserimento e l'invio al web server di un valore angolare qualsiasi, di tipo '`float`'. Di seguito l'analisi del codice necessario al fine di implementare l'inserimento da tastiera, la funzione '`sendCommand()`' non verrà trattata nuovamente poichè già spiegata in precedenza.

```
cp5.addTextfield("Input °")
    .setPosition(340, 810)
    .setSize(140, 40)
    .setFont(createFont("Arial", 28))
    .setFocus(true)
    .setColor(color(0, 0, 0))
    .setColorBackground(color(255, 255, 255))
    .setColorForeground(color(255, 20, 100))
    .setColorActive(color(0))
    .setColorValueLabel(color(0, 0, 0))
    .getCaptionLabel()
    .setFont(createFont("Arial", 24))
    .toUpperCase(false);

cp5.addLabel("infoLabel")
    .setText("°")
    .setPosition(420, 816)
    .setColor(color(0))
    .setFont(createFont("Arial", 32));

cp5.addBang("send")
    .setPosition(500, 810)
    .setSize(45, 40)
    .setLabel("Send")
    .setColorBackground(color(255, 0, 255))
    .setColorForeground(color(255, 255, 255))
    .setColorActive(color(0, 250, 100))
    .getCaptionLabel()
    .setFont(createFont("Arial", 18))
    .toUpperCase(false);
```

Figura 6.4: Inserimento da tastiera

- `cp5.addTextfield("Input °")`: Questa funzione genera un campo di testo in cui l'utente può inserire un valore, successivamente sono state settate le impostazioni grafiche

per definire colori, dimensioni, posizionamento e carattere.

- **cp5.addLabel("infoLabel"):** Viene creata un'etichetta che mostra semplicemente il simbolo dell'angolo ' $\circ$ ' per aiutare visivamente l'utente, al fine di fargli intuire che in quella casella andrà inserito un valore angolare, come nel caso precedente successivamente sono state definite le impostazioni grafiche.
- **cp5.addButton("send"):** Viene generato un pulsante chiamato '**Bang**', definito come '**send**' sull'interfaccia, sempre per aiutare l'utente. Quando l'utente clicca su questo pulsante il valore inserito viene convertito in una stringa tramite la funzione '**send inputValue()**'.

```
void sendInputValue() {  
    String degrees = cp5.get(Textfield.class, "Input °").getText();  
    println("Degrees to send: " + degrees);  
    sendCommand("sendDegrees", degrees);  
}
```

Figura 6.5: Void sendInputValue()

La funzione '**send inputValue()**' ha il compito di leggere il valore inserito nel campo di testo, stamparlo e inviarlo alla funzione '**sendCommand()**'.

### 6.2.3 Parte di visualizzazione dei dati

Nella parte di visualizzazione dei dati l'utente può visualizzare la posizione angolare del sistema misurata direttamente dall'encoder, inoltre può visualizzare un grafico aggiornato ciclicamente che mostra l'andamento dei valori letti da una foto-resistenza. Il grafico dovrebbe mostrare una parabola rivolta verso l'alto durante le fasi di auto-centraggio del sistema.

#### 6.2.3.1 Live Angle

Di seguito l'analisi delle principali funzioni per visualizzare l'angolo rilevato dall'encoder.

##### Analisi getESP32EncoderValue ()

```
void getESP32EncoderValue() {
    String server = "192.168.137.219";
    String url = "http://" + server + "/getValueEncoder";

    try {
        URL serverUrl = new URL(url);
        HttpURLConnection connection = (HttpURLConnection) serverUrl.openConnection();
        connection.setRequestMethod("GET");
        connection.connect();

        int responseCode = connection.getResponseCode();
        println("Response Code: " + responseCode);

        if (responseCode == 200) {
            BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                println("Received from ESP32: " + inputLine);
                if (inputLine.startsWith("VALUE EN:")) {
                    esp32Value = inputLine.substring(9).trim(); // Estrae il valore dopo "VALUE:"
                    esp32ValueEncoder = Integer.parseInt(esp32Value); // Converte la stringa in un intero
                    println("ESP32 Value: " + esp32ValueEncoder);
                }
            }
            in.close();
        } else {
            println("HTTP error code: " + responseCode);
        }
        connection.disconnect();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figura 6.6: Void getESP32EncoderValue ()

- **Definizione e Configurazione dell'URL:** Nella funzione getESP32EncoderValue come prima cosa viene definito e configurato l'URL da utilizzare.
- **Invio della Richiesta HTTP GET:** Viene creato un oggetto '**URL**' e si apre una connessione HTTP, in seguito la richiesta viene configurata come un metodo GET e successivamente inviata con '**connection.connect()**'.
- **Gestione della risposta:** Il codice d risposta HTTP viene recuperato e stampato.

- **Lettura della risposta:** Se il codice di risposta è '**200**', la funzione legge l'input tramite '**BufferedReader**'. Ogni riga viene letta per vedere se inizia con "**VALUE EN:**", se trovata la stringa allora viene salvato il valore numerico successivo a "**VALUE EN:**". Il valore numerico viene convertito e salvato nella variabile '**esp32ValueEncoder**'
- **Gestione degli errori:** Eventuali errori durante la connessione o la lettura saranno stampati per effettuare il Debug.

In sintesi la funzione **getESP32EncoderValue()** invia una richiesta HTTP GET a un server ESP32 per ottenere il valore attuale di un encoder collegato al sistema. Il valore ricevuto viene convertito in un numero intero per poi essere stampato a schermo sotto forma di angolo (°).

### Analisi Thread

```
// Thread per chiamare getESP32Value periodicamente
Thread thread = new Thread(new Runnable() {
    public void run() {
        while (running) {
            getESP32Value();
            getESP32EncoderValue();
            try {
                Thread.sleep(350); // Pausa Thread
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
});
thread.start();
```

Figura 6.7: Thread

- **Creazione del Thread:** Viene creato un thread che esegue un oggetto '**Runnable**'. All'interno di '**Runnable**' c'è un ciclo '**while**' che continua a funzionare finchè la variabile '**running**' è vera.
- **Esecuzione periodica delle funzioni:** Dentro il ciclo '**while**' vengono chiamate due funzioni: **getESP32Value()** e **getESP32EncoderValue()**. La funzione **getESP32Value()** verrà spiegata nel Self - Centering System Plot.  
Dopo aver eseguito entrambe le funzioni, il thread viene sospeso per 350 millisecondi con '**Thread.sleep(350)**', con questo comando è quindi possibile regolare la frequenza con la quale queste funzioni vengono eseguite.

Questo thread ha la funzione di chiamare in modo periodico le funzioni **getESP32Value()** e **getESP32EncoderValue()** rispettivamente per leggere il valore della fotoresistenza e dell'encoder.

### 6.2.3.2 Self - Centering System Plot

Lo sviluppo del codice per la visualizzazione di un grafico live sull'interfaccia utente (UI) è avvenuto utilizzando principalmente la funzione **grafico()** e la funzione **getESP32Value()**. In questa parte sarà analizzata solamente la funzione **grafico()** poichè la funzione **getESP32Value()** risulta quasi identica alla funzione **getESP32EncoderValue()**, già descritta in precedenza.

```

void grafico() {
    // Intervallo di aggiornamento
    if (millis() - lastUpdateTime > interval) {
        lastUpdateTime = millis();

        values.add(float(esp32ValueInt));

        // Aggiornamento del valore massimo
        if (max_value < esp32ValueInt) {
            max_value = esp32ValueInt;
        }

        // Rimozione del valore più vecchio se la lista supera maxValues
        if (values.size() > maxValues) {
            values.remove(0);
        }
    }

    // Dimensioni del grafico
    int enlargedGraphWidth = 550; // Larghezza del grafico
    int enlargedGraphHeight = 425; // Altezza del grafico
    int xOffset = 125; // Spostamento asse x
    int yOffset = 200; // Spostamento asse y

    // Disegno del rettangolo del grafico
    fill(255);
    rect(graphX - xOffset, graphY - enlargedGraphHeight + yOffset, enlargedGraphWidth, enlargedGraphHeight);

    // Impostazioni del tratto
    stroke(0);
    strokeWeight(3); // Imposta lo spessore della linea
    noFill();

    // Inizio
    beginShape();
    for (int i = 0; i < values.size(); i++) {
        // Mapping delle coordinate x e y con le dimensioni del grafico ingrandito e lo spostamento
        float x = map(i, 0, maxValues, graphX - xOffset, graphX - xOffset + enlargedGraphWidth);
        float y = map(values.get(i), 1000, 000, graphY - enlargedGraphHeight + yOffset, graphY + yOffset);
        vertex(x, y);
    }
    // Fine
    endShape();

    // Impostazioni del testo
    fill(255);
    textSize(25);
    // Testo dell'ultimo valore
    text("Last Value: " + esp32ValueInt, graphX - xOffset + 100, graphY + yOffset + 20);
    // Testo del valore massimo
    text("Max Value: " + max_value, graphX - xOffset + 400, graphY + yOffset + 20);
}

```

Figura 6.8: Void grafico()

- **Aggiornamento dei Valori:** La funzione verifica se è passato abbastanza tempo ('interval') dall'ultimo aggiornamento. Il valore corrente della foto-resistenza ('esp32ValueInt') viene convertito in 'float' e aggiunto alla lista 'values' contenente tutti i valori recenti. Se la lista valori supera il numero massimo di valori consentiti, viene rimosso il valore più vecchio per mantenere costante la dimensione della lista FIFO (First In First Out).
- **Impostazione delle Dimensioni del Grafico:** In seguito vengono definite le dimensioni del grafico e gli spostamenti rispetto agli assi 'x' e 'y' della finestra di visualizzazione.

- **Disegno del Rettangolo del Grafico:** Viene disegnato un rettangolo bianco ('**fill(255)**') che rappresenta lo sfondo del grafico. Il rettangolo viene posizionato in base agli offset e alle dimensioni definite.
- **Tracciamento del Grafico:** La linea del grafico viene disegnata con un certo spessore ('**strokeWeight(3)**'), viene utilizzata la funzione '**beginShape()**' per iniziare a tracciare il grafico. Ogni valore nella lista '**values**' viene mappato a una posizione '**x**' e '**y**' nel grafico. La funzione '**map()**' converte l'indice del valore e il valore stesso in coordinate appropriate all'interno del grafico. La funzione '**vertex(x,y)**' aggiunge un vertice alla forma del grafico in base alle coordinate calcolate. Infine '**endShape()**' completa il disegno della linea che rappresenta l'andamento dei valori.
- **Visualizzazione del testo:** Viene visualizzato in basso, a sinistra del grafico, mostrando l'ultimo valore letto ('**Last Value**'), a destra invece viene visualizzato il valore massimo letto ('**Max Value**').

La funzione aggiorna periodicamente i dati del grafico, in questo caso è stato richiesto che mostrasse i valori della foto-resistenza per il sistema di auto-centraggio, ma con estrema facilità è possibile convertire la funzione affinchè mostri l'andamento di qualsiasi sensore del sistema.



## Capitolo 7

# Circuito Stampato

### 7.1. Componenti PCB

La progettazione di un **PCB (Printed Circuit Board)** è un processo fondamentale per la costruzione di tutti i dispositivi elettronici.

Prima di iniziare la progettazione di un **PCB**, è essenziale avere una chiara comprensione dei componenti che si andranno a utilizzare, inclusi valori, dimensioni e specifiche elettriche.

In questo progetto, volto a sviluppare un sistema di controllo per la misura AOI (Angle of Incidence), la decisione di realizzare un circuito stampato è stata presa per permettere di avere una certa pulizia in termini di cavi e di componenti utilizzati.

Sulla scheda andranno posizionati con un certo criterio i due microcontrollori utilizzati, resistenze per regolare le tensioni d'ingresso nei pin dei microcontrollori di segnali provenienti dai sensori utilizzati e infine un circuito per regolare la tensione di alimentazione.

#### 7.1.0.1 Componenti Microcontrollori

I due microcontrollori utilizzati sono un **Arduino Uno** e un **ESP32** entrambi già descritti in precedenza. Per far comunicare i due microcontrollori è stata utilizzata la comunicazione seriale(UART) utilizzando i pin **TX (trasmissione)** e **RX (ricezione)**. Il pin RX di esp32 lavora a 3.3V mentre il pin TX di arduino uno a 5V, è stato quindi necessario utilizzare un partitore di tensione per evitare di danneggiare l'esp32.

## Calcolo

**Formula generale partitore di tensione** →  $V_{\text{out}} = V_{\text{in}} \frac{R_2}{R_1 + R_2}$

- $R_2 = 2k\Omega$
- $R_1 = 1.1k\Omega$
- $V_{\text{in}} = 5V$

$$V_{\text{out}} = 5V \frac{2k\Omega}{2k\Omega + 1.1k\Omega} = 3.22V$$

### 7.1.0.2 Componenti Sensori

#### Sensore induttivo di finecorsa

Per il sensore induttivo, utilizzato come sistema di finecorsa, è stato necessario utilizzare un partitore per ridurre la tensione sul pin analogico del microcontrollore.

Il sensore è alimentato a +24V, il microcontrollore sui pin analogici permette di leggere segnali con un'ampiezza massima di 5V. Per questo motivo è stato fatto un partitore di tensione per diminuire la tensione da 24V a circa 3.3V.

## Calcolo

**Formula generale partitore di tensione** →  $V_{\text{out}} = V_{\text{in}} \frac{R_2}{R_1 + R_2}$

- $R_2 = 2k\Omega$
- $R_1 = 13k\Omega$
- $V_{\text{in}} = 24V$

$$V_{\text{out}} = 24V \frac{2k\Omega}{2k\Omega + 13k\Omega} = 3.2V$$

## Fotoresistenza

La fotoresistenza per essere letta correttamente è stata messa all'interno di un partitore di tensione con un'altra resistenza di valore pari al suo. La fotoresistenza è stata misurata e di conseguenza è stata presa un'altra resistenza di valore pari a  $10k\Omega$ , il tutto alimentato a 3.3V.

### 7.1.0.3 Componenti Circuito di Regolazione di Tensione

Come sarà spiegato meglio in seguito il PCB è suddiviso in 3 diverse zone di alimentazione, la funzione di questo circuito di regolazione di tensione è quella di abbassare la tensione d'ingresso da 24V a 5V al fine di alimentare i due microcontrollori e l'encoder.

Come regolatore a commutazione di tensione STEP-DOWN è stato utilizzato LM2596T-5.0/NOPB attorno al quale sono stati scelti gli altri componenti del circuito di regolazione.

#### Analisi Pin LM2596T-5.0/NOPB

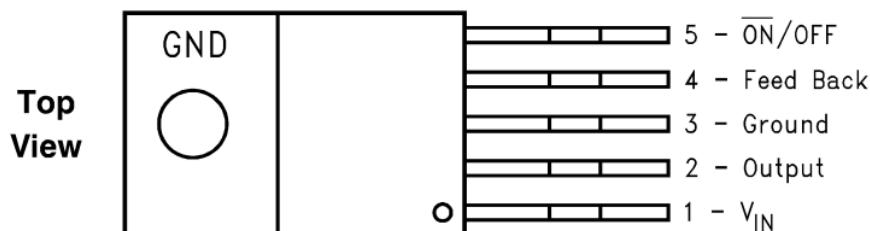


Figura 7.1: Pin LM2596T-5.0/NOPB

- **Vin** Questo è il pin positivo di alimentazione. È necessario collegare un condensatore di bypass adeguato a questo pin.
- **Output** Switch interno. La tensione su questo pin varia tra  $(+V_{IN} - V_{sat})$ , con un duty cycle di  $V_{out}/V_{in}$ .
- **Ground** Ground del circuito
- **Feedback** Misura la regolazione della tensione d'uscita per il feedback loop.
- **ON/OFF** Consente di spegnere il circuito del regolatore di commutazione tramite segnali logici, riducendo così la corrente di alimentazione in ingresso totale a circa 80

$\mu\text{A}$ . Se la tensione sul pin è al di sotto di una tensione di soglia di circa 1,3 V, si accende il regolatore, mentre alzando la tensione di questo pin al di sopra di 1,3 V (fino a un massimo di 25 V), si spegne il regolatore. Se questa funzione di spegnimento non è richiesta, il pin ON/OFF può essere cablato al pin GND oppure può essere lasciato aperto. In entrambi i casi, il regolatore sarà in condizione ON.

Di seguito sono elencati i componenti utilizzati per ultimare il circuito:

- Condensatore d'ingresso  $680\mu\text{F}$
- Condensatore da  $220\mu\text{F}$
- Diodo Schottky 30V, 3A
- Induttanza sull'uscita da  $330\mu\text{H}$

## 7.2. Schematico

Disegnare lo schematico del PCB è stato il primo passo per progettare il circuito elettronico. Possiamo dire che lo schematico rappresenta una 'mappa visiva del circuito'.

Durante lo svolgimento del progetto spesso ho trovato molto utile consultare lo schematico per evitare di commettere errori.

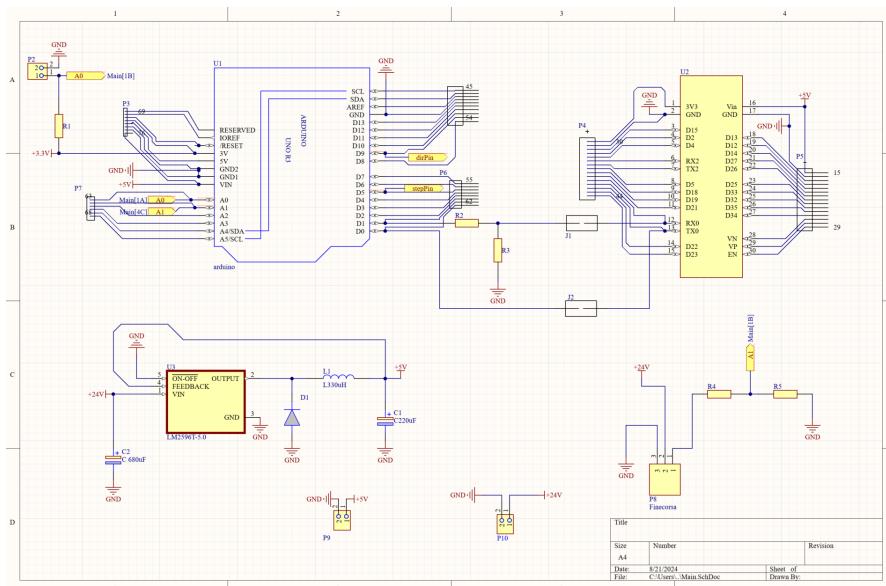


Figura 7.2: Schematico

### 7.3. PCB

Il PCB è formato da 4 layer, **Top Layer**, **Bottom Layer**, **Layer di Alimentazione**, **Layer per il GND**. I componenti del PCB sono stati suddivisi con criterio sui due lati disponibili, questo per cercare di contenere al massimo le dimensioni.

### 7.3.1 Top Layer

Sul top layer sono presenti l'arduino uno e i morsetti attraverso i quali i sensori possono essere alimentati e inviare dati.

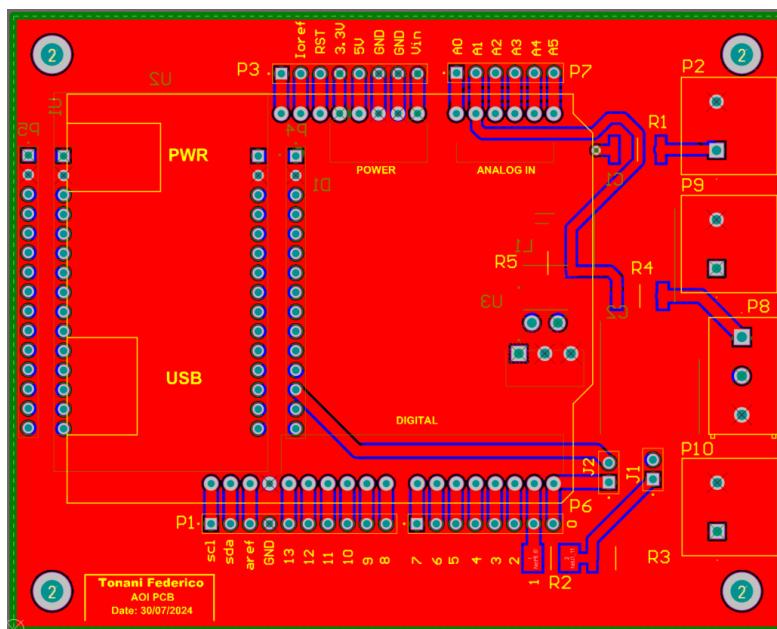


Figura 7.3: Top Layer

### 7.3.2 Bottom Layer

Sul bottom layer sono stati inseriti l'esp32 e il circuito regolatore di tensione.

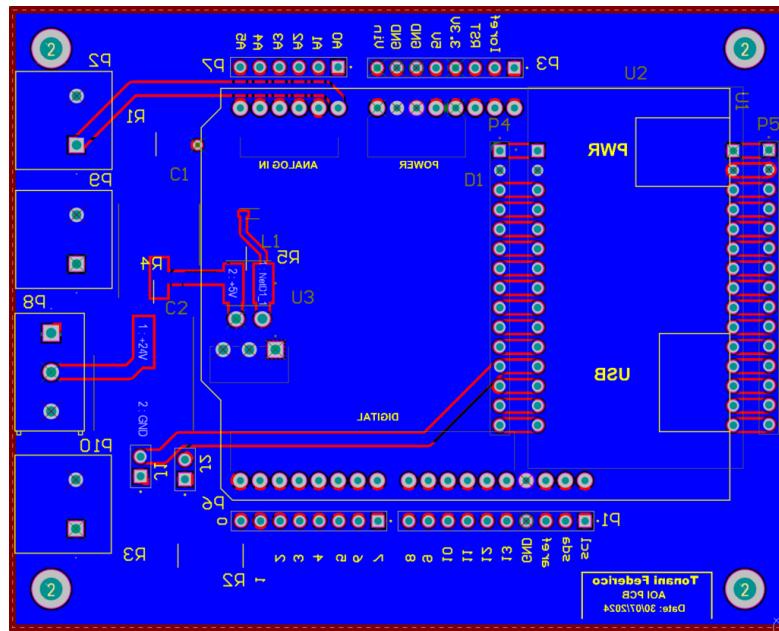


Figura 7.4: Bottom Layer

### 7.3.3 Layer Alimentazione

Il layer di alimentazione è progettato per garantire una distribuzione efficace delle tensioni di alimentazione ai vari componenti elettronici. Esso permette inoltre di ridurre le perdite di tensione, migliorare il raffreddamento e ridurre le interferenze elettromagnetiche ( utile poichè utilizziamo sensori ).

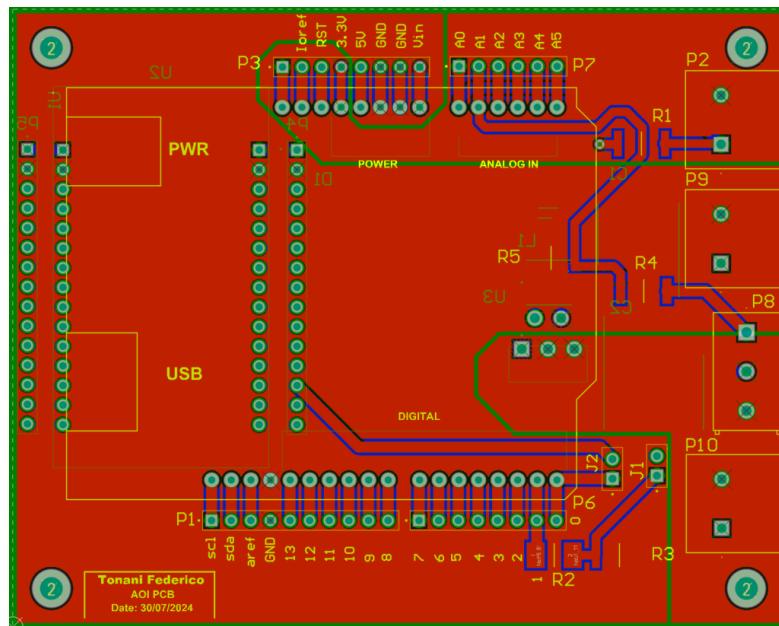


Figura 7.5: Layer Alimentazione

L'alimentazione del circuito è suddivisa in 3 zone:

- **Alimentazione 3.3V**

- **Alimentazione 5V**

- **Alimentazione 24V**

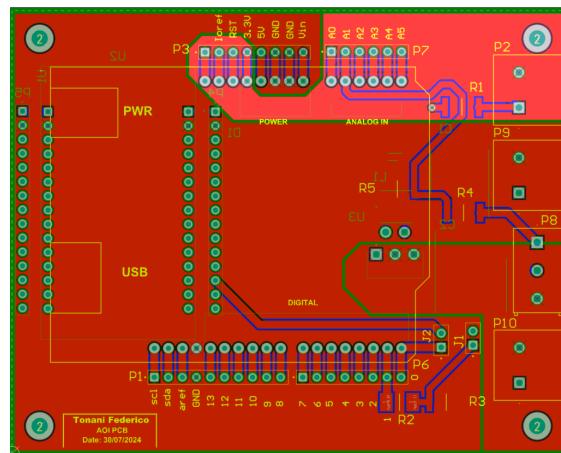


Figura 7.6: Alimentazione 3.3V

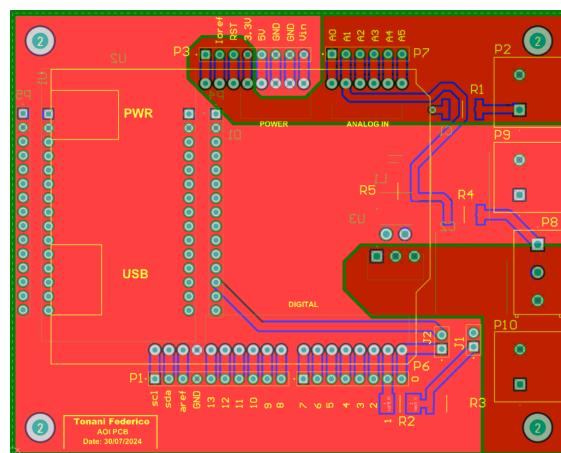


Figura 7.7: Alimentazione 5V

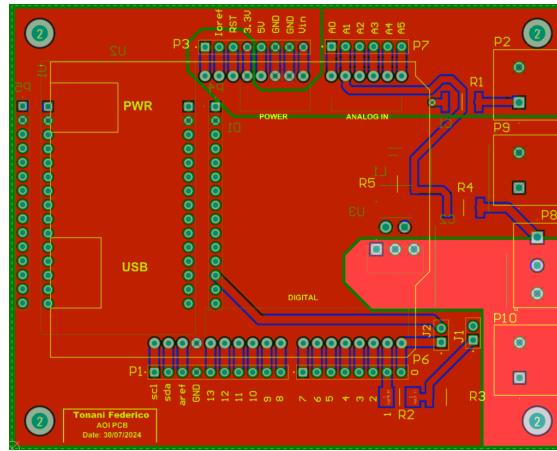


Figura 7.8: Alimentazione 24V

### 7.3.4 Layer GND

Il layer di massa (GND), come quello di alimentazione se inserito, riporta più o meno li stessi vantaggi, inoltre permette di avere un riferimento in comune comodo, di fatti, sia il layer di alimentazione che quello GND semplificano la gestione delle piste all'interno del PCB. Questo fattore rende il design del PCB più pulito e riduce il rischio di errori di connessione.

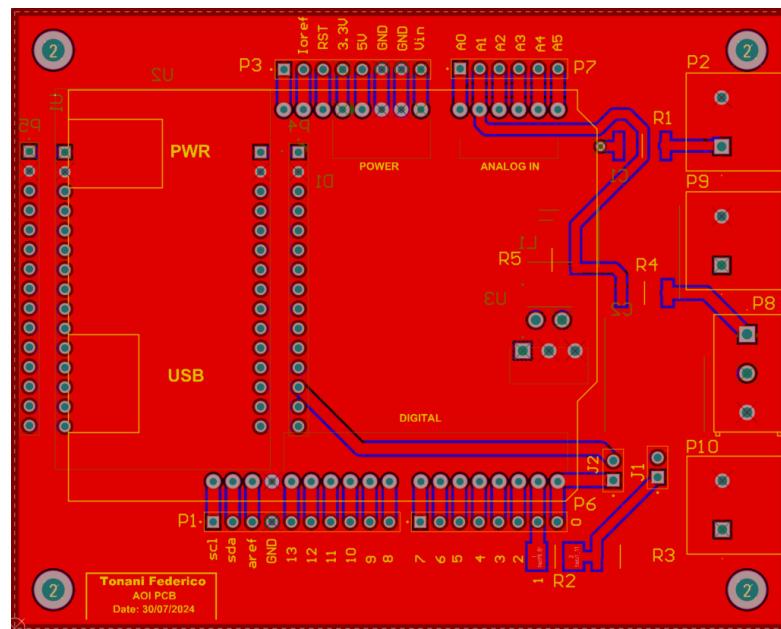


Figura 7.9: Layer GND

## Capitolo 8

# Test

In questo capitolo sarà illustrata tutta la parte inerente ai test quindi svolgimento del test, raccolta dei dati e analisi dei risultati.

### 8.1. Misura angolo d'incidenza

La misura fotovoltaica dell'angolo d'incidenza viene svolta misurando la **Isc (corrente di cortocircuito)** del modulo e si riferisce alla variazione della caratteristica della cella fotovoltaica in relazione all'angolo d'incidenza.

Questo metodo è utilizzato per comprendere come l'inclinazione dei pannelli fotovoltaici influenzi la loro capacità di produrre energia.

### 8.1.1 Dettagli Test

1. È stata centrata la struttura rispetto al simulatore solare.

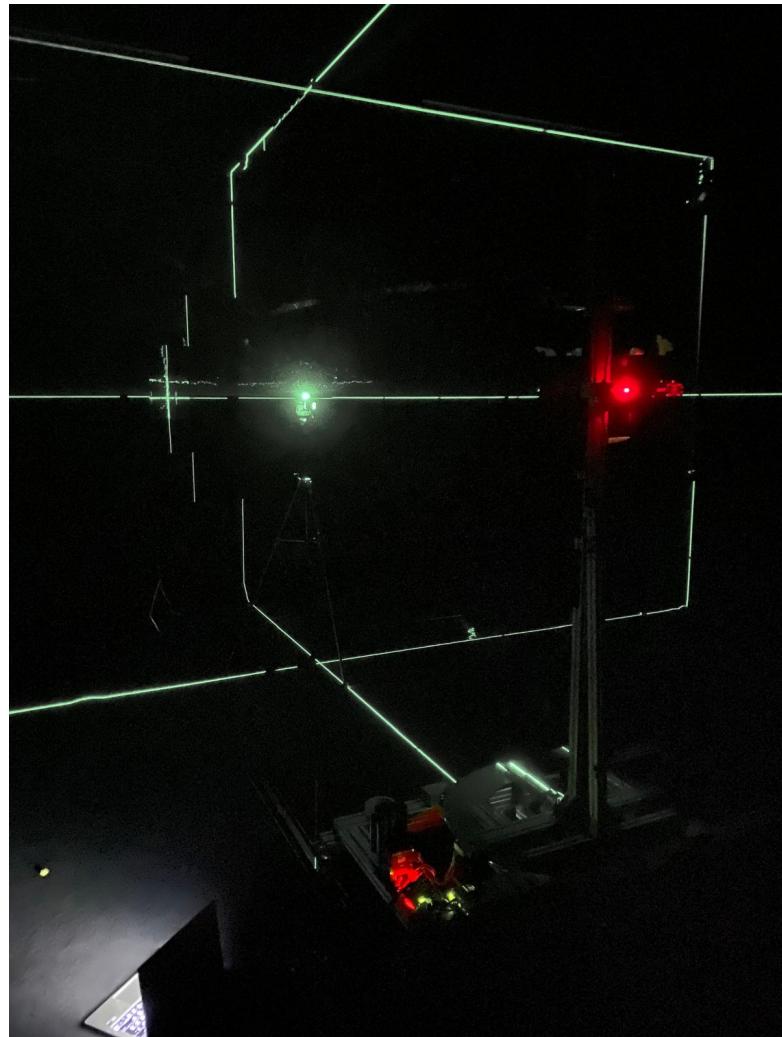


Figura 8.1: Centraggi struttura

2. Il secondo step è stato quello di montare il modulo fotovoltaico sulla struttura.

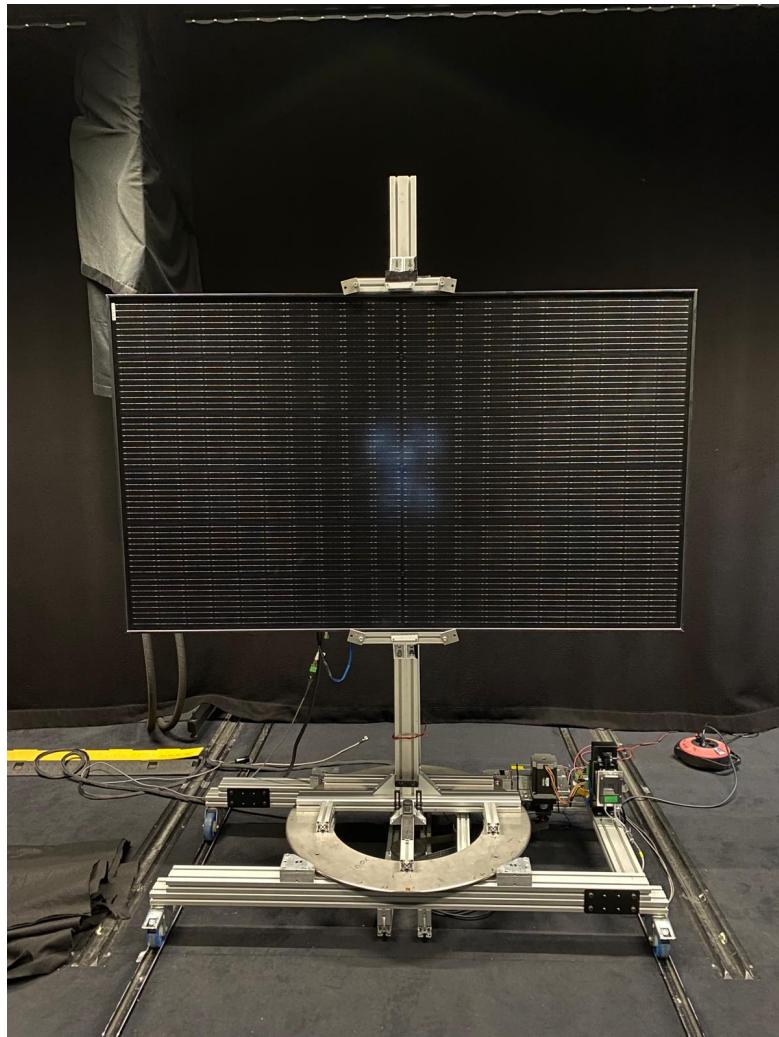


Figura 8.2: Montaggio Modulo Fotovoltaico

3. Per iniziare a ridurre le riflessioni è stata applicata un'apposita coperta sulla base della struttura, successivamente la struttura andrà verniciata con una vernice nera speciale per ridurre al minimo le riflessioni della luce.

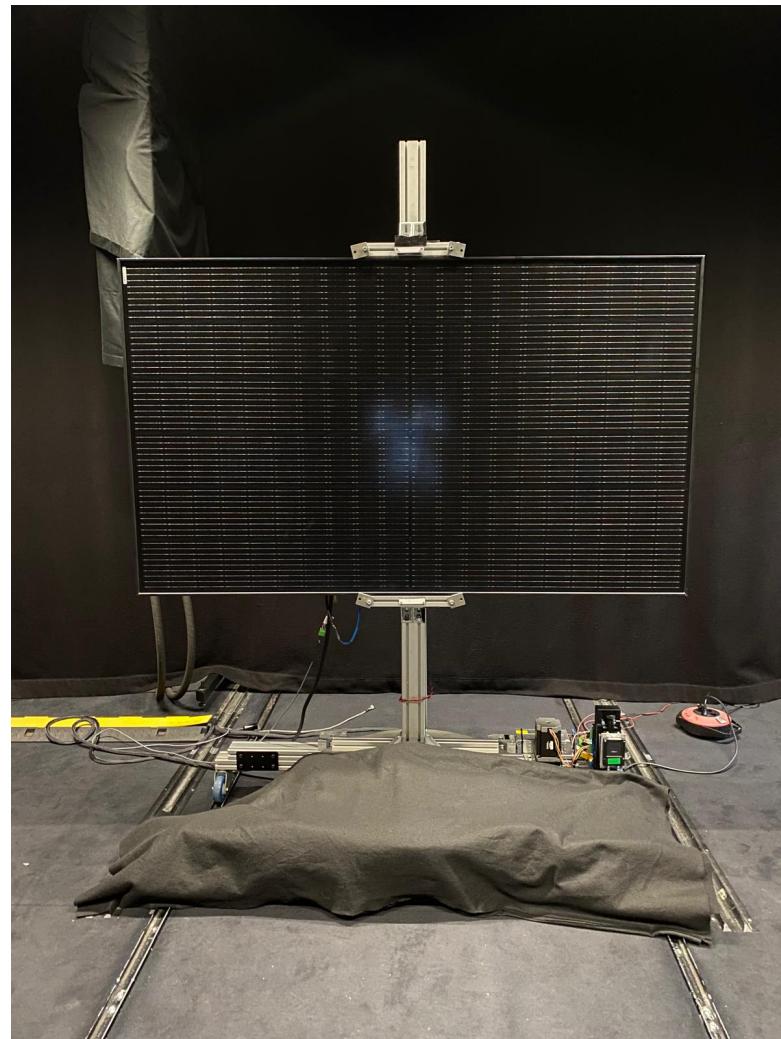


Figura 8.3: Coperta Struttura

4. Le misure sono state prese per i seguenti angoli: **0°, 10°, 20°, 30°, 40°, 50°, 60°, 65°, 70°, 75°, 80°, 85°, 90°** sia nel verso positivo che nel verso negativo. Le stesse misure sono state prese in modo analogo ombreggiando due celle del modulo.

### 8.1.2 Risultati Ottenuti e Analisi

Il test completo, quindi partendo dalla fase di acquisizione dati fino all'analisi di essi richiede tempo, le analisi vengono fatte da un software apposito e il processo di acquisizione dati risulta lungo. Di seguito uno dei grafici finali che riassume tramite un grafico excel i risultati del test.

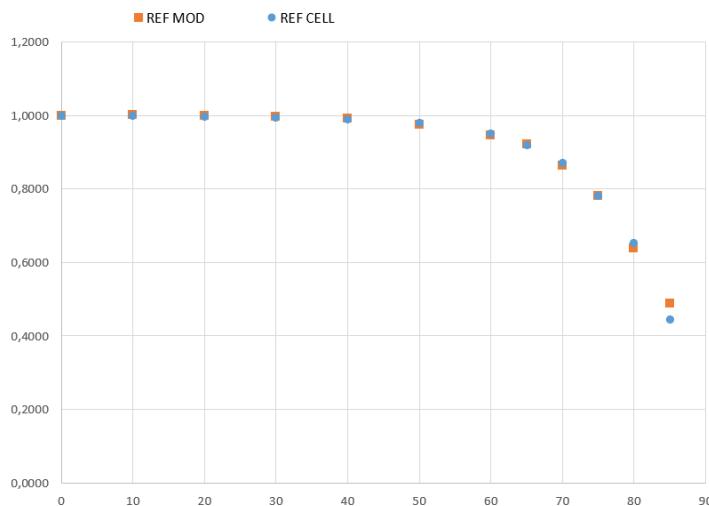


Figura 8.4: Grafico Risultati

In questo grafico vengono confrontate le misure del modulo fotovoltaico utilizzato nel test con quelle della cella di riferimento. Nello specifico viene analizzata **L'Incident Angle Modifier (IAM)** o modificatore dell'angolo d'incidenza, è un parametro utilizzato per descrivere come varia l'efficienza di raccolta della luce di un modulo fotovoltaico in funzione dell'angolo d'incidenza della luce.

Come possiamo notare dal grafico i risultati del modulo con quelli della cella di riferimento sono quasi sovrapponibili, questo indica che la struttura è adeguata per svolgere questo tipo di test.

Purtroppo come già specificato in precedenza il test risulta parecchio lungo di conseguenza è stato possibile svolgerne solamente uno documentato, gli altri verranno portati durante la difesa della tesi.



## Capitolo 9

# Conclusioni

La consegna iniziale del progetto prevedeva la realizzazione di una nuova struttura che fosse in grado di svolgere il test della misura dell'**angle of incidence (AOI)**. Per questioni di tempistiche e di budget non è stato possibile realizzarne una nuova, è stato quindi deciso di riadattare quella già presente con l'obiettivo di migliorarla quanto più possibile. La parte iniziale di progettazione di una nuova struttura, che purtroppo non ha portato risultati, ha comportato la perdita di 3 settimane di lavoro.

La precedente struttura è stata migliorata introducendo un encoder per poter utilizzare la tecnica del controllo con feedback, inoltre l'integrazione di un'interfaccia grafica wireless permette agli operatori di effettuare i test senza recarsi fisicamente a controllare nella camera oscura ad ogni nuovo posizionamento poichè i dati sono visualizzabili in real time direttamente dall'interfaccia, questo comporta un notevole risparmio di tempo per loro e quindi una riduzione del tempo necessario per svolgere l'intero test.

I risultati ottenuti nel Test di prova sono incoraggianti, purtroppo come specificato in precedenza il test completo richiede tempo e la camera oscura risulta molto spesso inagibile poichè già in uso. Altri test e valutazioni verranno fatti nei giorni a seguire e saranno portati in sede di discussione della tesi.



# Bibliografia

- [1] Automazione PLC. Encoder: cosa sono e differenze tra encoder assoluti e incrementali. <https://automazione-plc.it/encoders-differenze.html>, 2024. Ultimo accesso: 16 Agosto 2024.
- [2] Digikey Electronics. The fundamentals of proximity sensors: Selection and use in industrial automation. <https://www.digikey.it/it/articles/the-fundamentals-of-proximity-sensors-selection-and-use-industrial-automation>, 2023. Accessed: 2024-08-19.
- [3] Texas Instruments. Lm2596 datasheet. [https://www.ti.com/lit/ds/symlink/lm2596.pdf?ts=1724304205521&ref\\_url=https%253A%252F%252Fwww.mouser.ch%252F](https://www.ti.com/lit/ds/symlink/lm2596.pdf?ts=1724304205521&ref_url=https%253A%252F%252Fwww.mouser.ch%252F), 2024. Accessed: 2024-08-22.
- [4] RS Components. Arduino. <https://it.rs-online.com/web/content/discovery-blog/idee-suggerimenti/guida-uso-arduino>. Accessed: 2024-08-23.