

Relazione di laboratorio del corso di

# Progettazione e Prototipazione di Sistemi Elettronici

A.A. 2023/2024

**Docenti: Corrà Michele, Dalpez Stefano**

---

**Componenti del Gruppo:**

*Alessandro Bianchi Ceriani, Gerardo Chianucci, Emilian Manole, Daniele Visentin*

# Indice

1 - Introduzione.....	3
2 - Assemblaggio scheda.....	3
3 - Descrizione componenti.....	5
4 - Note implementative.....	6
5 - Funzionamento del sistema.....	8
6 - Struttura - Modello 3D.....	13
7 - Segnalazione luminosa - Alert di Stato.....	16
8 - PlatformIO e arduino-pico.....	16
9 - Descrizione del codice.....	18
10 - Conclusioni.....	24
11 - Bibliografia e Sitografia.....	25

# 1 - Introduzione

La presente relazione documenta il lavoro svolto nella realizzazione del progetto "**SunScanner**".

SunScanner consiste in un **sistema di rilevamento e puntamento del Sole**, basato su dati di geolocalizzazione forniti da una combinazione di **GPS e magnetometro**, coadiuvati dal rilevamento del rendimento di un **pannello solare**. Per il posizionamento preciso viene impiegato un sistema di **pan-and-tilt** che sfrutta l'azione di due **servomotori**.

Il cuore del progetto è rappresentato da una scheda programmabile basata sul microcontrollore **RP2040**, progettata da studenti di anni precedenti del corso di

"**Progettazione e Prototipazione di Sistemi Elettronici**".

Il lavoro è stato svolto principalmente in due fasi:

- **prima fase:**

popolazione della PCB con i componenti principali, che sono stati **montati** tramite pasta saldante con successiva messa in forno (metodo di saldatura a rifusione) e altri saldati a mano.

- **seconde fase:**

**programmazione** del microcontrollore, creazione **struttura** di supporto e **testing**.

L'obiettivo del progetto è stato quello di **acquisire esperienza** nell'ambito della progettazione sia hardware che software di dispositivi basati su microcontrollore RP2040.

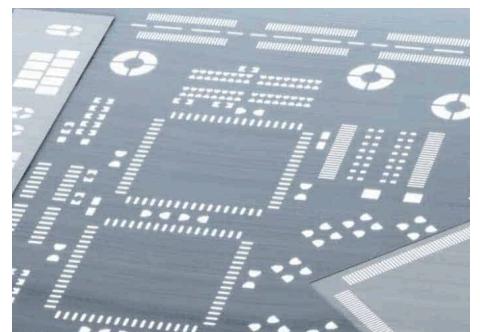
# 2 - Assemblaggio scheda

Per questo progetto ci è stata fornita una scheda non assemblata con lo **scopo di montare tutti i componenti** necessari al suo funzionamento.

L'assemblaggio della scheda ha richiesto diverse fasi:

## Fase n° 1 - Stesura della pasta saldante

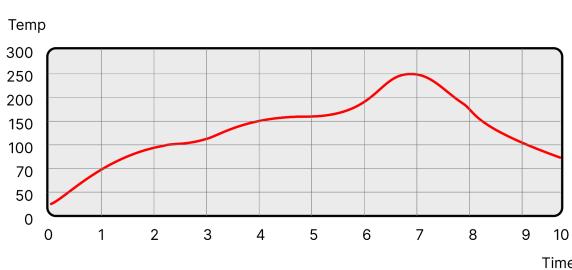
- Abbiamo costruito un telaio che ci ha permesso di **allineare perfettamente lo stencil dei footprint**, realizzato in acciaio con spessore di 120 micron, alla scheda sottostante.
- Successivamente abbiamo steso della **pasta saldante** sopra lo stencil, **spalmandola** con delle spatole affinché andasse a **coprire** uniformemente i **pad** di ogni footprint.
- Prima di estrarre la scheda dal telaio, è **sempre meglio verificare che tutti i footprint siano coperti da pasta saldante**; dopo un'attenta ispezione della stesura della pasta sui pad possiamo delicatamente estrarre la scheda dall'intelaiatura.



## Fase n° 2 - Posizionamento dei componenti

- In questa fase ci siamo occupati di **posizionare i vari componenti elettronici** con delle pinzette; all'occorrenza ci siamo serviti di uno stereoscopio, soprattutto nel fissare componenti di dimensioni ridotte con tanti pin (vedi il microcontrollore).
- Durante questa fase è fondamentale **seguire passo - passo lo schematico** dei componenti, segnando man mano i componenti appena posizionati.
- Visto che abbiamo impiegato più giorni per completare il posizionamento dei componenti, abbiamo **applicato nuovamente la pasta saldante** ove necessario (la pasta secca rende difficile il posizionamento e l'adesione dei componenti SMD alla scheda).

## Fase n° 3 - Saldatura scheda (metodo di saldatura per rifusione)

- Dopo aver messo tutti i componenti al loro posto e aver verificato il corretto allineamento dei pin sui pad si passa alla saldatura della scheda.
- La **saldatura viene fatta utilizzando un forno apposito per saldature**, che ha una curva di temperatura diversa da un comune forno domestico: la temperatura sale lentamente, raggiunge un plateau dove il valore rimane costante per dei minuti e poi ridiscende sempre lentamente con degli scaglioni di tempo.
- La prima parte della salita della temperatura serve per **eliminare, se presente, dell'umidità** nella pasta saldante, che potrebbe creare delle bolle all'interno se si fondesse immediatamente e quindi dei problemi.
- La seconda parte di stallo dove la temperatura si assesta all'incirca sui 250° C dove la **pasta si fonde saldando i componenti al PCB**.
- La terza e ultima parte dove la temperatura scende lentamente per fare in modo che **non si creino delle cricche** (microfratture) nella saldatura, che altrimenti potrebbero causare dei problemi.

## Fase n° 4 - Ispezione e controllo di qualità

- Una volta che la scheda è tornata a temperatura ambiente **è possibile verificare con lo stereoscopio se la saldatura** nel forno sia avvenuta con successo; in caso contrario è possibile aggiustare usando una punta saldante e dello stagno in filo.
- Alcuni componenti sono poi stati aggiunti e **saldati manualmente**, in quanto l'intero processo di assemblaggio (come già accennato) è stato svolto nell'arco di diverse sessioni di laboratorio.

## Fase n° 5 - Testing e debugging

- Una volta finito l'assemblaggio ci muniamo di **multimetro** e, all'occorrenza, di **oscilloscopio** per verificare i collegamenti.

- Fatto ciò la scheda è **terminata dal punto di vista hardware**; viene verificato il funzionamento anche a livello software programmandola con del semplice codice e modificandola dove fosse necessario.
- Una volta avuta la certezza dell'operatività della scheda in ogni suo componente, è possibile passare allo sviluppo di **codice** che sfrutti le **potenzialità hardware** specifiche e alla strutturazione di un **progetto** vero e proprio.

## 3 - Descrizione componenti

La scheda è composta da molteplici componenti, di seguito sono riportati i principali:

- **RP2040** - microcontroller di debutto dell'azienda Raspberry Pi (meglio conosciuta per i propri on-chip-computers) che fornisce alte prestazioni ad un costo contenuto e garantendo facilità d'utilizzo.

### Le features più notevoli di questa MCU sono:

- Architettura ARM-Cortex M0+ dual core
- 264kB di on-chip SRAM
- Supporto per memoria Flash esterna fino a 16MB con bus QSPI dedicato
- Bootloader UF2 in ROM
- 30 pin GPIO, 4 dei quali possono essere configurabili come analogici
- Ricco set di periferiche (che include I2C, SPI, UART ecc.)
- 2 blocchi di Programmable I/O (PIO), una feature particolare di questo microcontrollore: consistono in 4 macchine a stati programmabili per ogni blocco, con delle FIFO in input e output e altra circuiteria che permette di implementare particolari protocolli di comunicazione agendo direttamente sulle risorse hardware
- Consumi di potenza ridotti, supporto per modalità di funzionamento low-power

L'abbondanza di documentazione disponibile conferma la solidità di questa piattaforma.

- **ublox SAM-M8Q** - modulo antenna *ublox* con ricezione simultanea di dati da tre GNSS (GPS, Galileo e GLONASS). La sua **robustezza, costo competitivo** (relativamente alla performance) e fattore di forma ridotto lo rendono una **scelta ottimale** per una soluzione portatile che richieda una buona precisione di localizzazione - il modello in questione vanta una accuratezza di posizione orizzontale di 2.5m utilizzando GPS. I dati ricevuti vengono forniti nel **formato standard NMEA 0183**, estremamente popolare tra i sistemi di localizzazioni ad uso civile.
- **Display OLED SSD1306** - tipologia molto diffusa di schermi bicolore (blu e giallo) per applicazioni IoT e simili. Presenta una **risoluzione di 128x32 pixel** e sono disponibili svariate librerie driver ad alte prestazioni per facilitarne l'utilizzo.
- **Magnetometro/Bussola QMC583L** - Il modulo in questione consiste in un **sensore magnetico** multi-chip a tre assi e in una circuiteria dedicata per **signal processing**; garantisce **alta precisione** (1° e 2° di deviazione massimi nel direzionamento della

bussola), basso rumore, consumi ridotti, cancellazione degli offset e compensazione della temperatura. Inoltre, l'interfacciamento è semplice dato l'integrazione di un bus I2C. Questo modulo risulta **essenziale** per l'orientamento del pannello congiuntamente al ricevitore GNSS, in quanto permette di individuare la **direzione** del **Nord magnetico** terrestre con accuratezza.

- **Servomotori** - forza motrice vera e propria della struttura di supporto al pannello, rendono possibile il movimento su due **assi**, azimutale e polare (anche detti **pan** e **tilt**) rispettivamente di 360° e 180°. Le dimensioni e il peso ridotti sono ideali per un sistema mobile, mantenendo una buona coppia e una resistenza elevata.
- **LED RGB indirizzabili WS2812B** - sulla scheda è stata montata una serie di 8 LED RGB collegati in cascata, **ognuno** dei quali può essere **indirizzato individualmente**. Questo ci permette di pilotare ogni diodo indipendentemente, manipolando luminosità e colore.

## 4 - Note implementative

### **Partitori di tensione**

Nel design della scheda programmabile sono stati sfruttati dei semplici circuiti elettronici detti **partitori di tensione**; consistono in **due resistenze** configurate una in serie e una in parallelo al generatore/carico, e consentono di **fractionare la tensione** in uscita. La formula generica per il calcolo della tensione in uscita in funzione di quella in ingresso e del valore delle resistenze è ricavabile con delle semplici tecniche di analisi circuitale, ed è riportata di seguito:

$$V_{OUT} = \frac{R_2}{R_1 + R_2} * V_{IN} \quad \text{dove } R_2 \text{ è la resistenza su cui rilevo } V_{OUT}.$$

I due esempi più notevoli di questo circuito presenti sulla scheda si trovano nella parte di alimentazione, specificamente come parte della rete di **feedback del convertitore buck** e come parte del circuito di lettura della tensione della batteria.

Nel primo caso il circuito è stato implementato come da **indicazioni presenti nel datasheet** del convertitore di tensione;

nel secondo caso l'obiettivo è di abbassare la tensione della batteria in modo che possa essere letta da un pin "analogico" del microcontrollore. Facendo riferimento ai datasheet dell'RP2040, la **tensione massima** rilevabile dai pin ADC è **ADC\_AVDD**, ossia 3.3V nella nostra implementazione. Per questo motivo è stato installato un partitore con

$R_1 = 46k\Omega$ ,  $R_2 = 10k\Omega$ , in modo che data una  $V_{IN}^{MAX} = 12V$  si ottenga

$V_{OUT}^{MAX} = \frac{10k\Omega}{10k\Omega + 46k\Omega} * V_{IN}^{MAX} \approx 2.14V$ . In questo modo le tensioni in ingresso al microcontrollore rimangono tranquillamente entro i limiti consentiti.

**N.B.** i valori delle resistenze menzionati nello schematico sono sbagliati, in quanto venivano previste  $R_1 = 150k\Omega$ ,  $R_2 = 56k\Omega$  che avrebbero portato  $V_{OUT}^{MAX} \approx 3.2V$ , valore troppo vicino ai limiti consentiti che non lascia margine per possibili fluttuazioni.

## Pannello fotovoltaico

Nel caso del pannello è necessario **leggere una tensione analogica sulla scheda**, sfruttando uno dei pin ADC dell'RP2040. Nel design originale non era prevista una connessione libera ad uno dei 4 pin, perciò abbiamo deciso di **rimuovere il sensore di temperatura**, irrilevante per i nostri scopi.

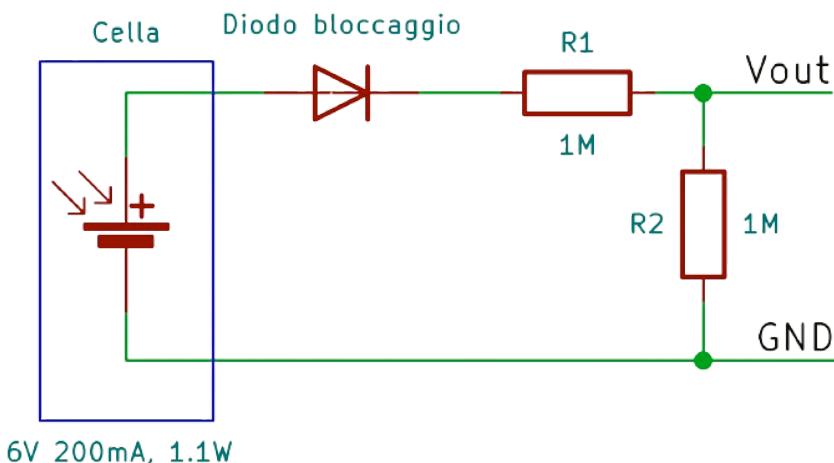
Per ottenere delle letture accurate va preso in considerazione il **filtro RC** passa-basso posto prima del sensore: la frequenza di cutoff  $f_c$  del filtro è calcolabile con la formula

$f_c = 0.5 * (\pi RC)^{-1} = 0.5 * (\pi * 470\Omega * 1\mu F) \approx 338.6 \text{ Hz}$ , che corrispondono ad un periodo  $T = 1/f_c \approx 2.9 \text{ ms}$ ; ciò significa che si potranno apprezzare soltanto variazioni di tensione con una frequenza inferiore a  $\sim 340 \text{ Hz}$ .

Per questo motivo, per i nostri scopi questo filtro **risulta comunque comodo** per eliminare picchi indesiderati o, nel peggiore dei casi, ininfluente.

Un altro aspetto da non sottovalutare è il **circuito di protezione** del pannello: questo previene ritorni di corrente al pannello stesso, tramite un **diodo di bloccaggio**, e abbassa la tensione letta dalla scheda mediante un **partitore a 1/2**. Come accennato in precedenza, i pin analogici del microcontrollore possono sopportare tensioni in ingresso non superiori alla tensione di alimentazione del modulo ADC, quindi 3.3V nel nostro caso. Di seguito viene fornito il diagramma circuitale e il calcolo della Vout con la formula del partitore. Da notare che sono state scelte delle resistenze da **1MΩ** in quanto è preferibile avere **alta impedenza in ingresso** (dal punto di vista del micro) per misurare tensioni.

$$V_{OUT} = \frac{R_2}{R_1 + R_2} * V_{Cella} = \frac{1M\Omega}{1M\Omega + 1M\Omega} * V_{cella} = \frac{1}{2} * V_{Cella}$$



*Circuito di protezione collegato al pannello solare*

## 5 - Funzionamento del sistema

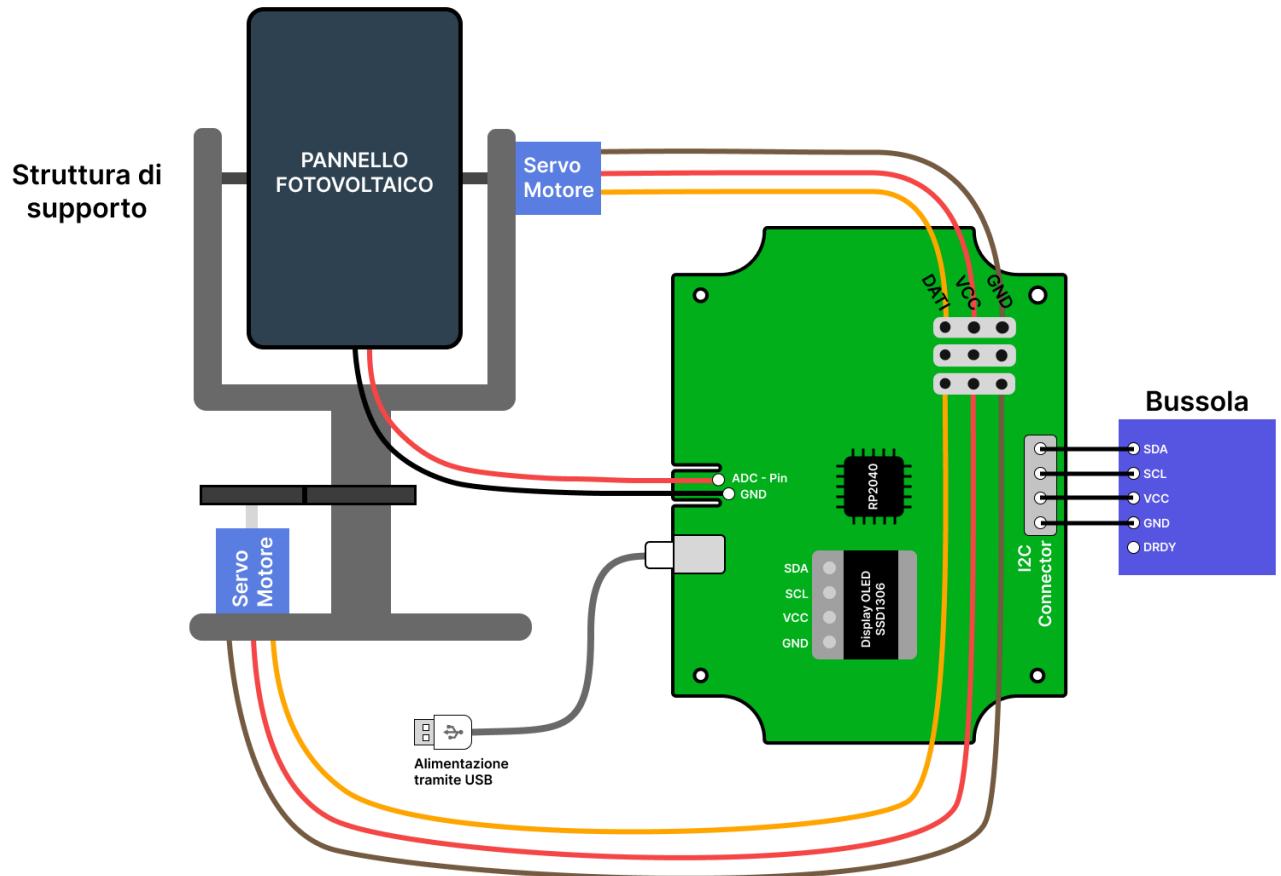
### Collegamenti elettrici

Come già esposto, il sistema “**SunScanner**” prevede l’uso di una serie di elementi elettromeccanici e sensori esterni, che si interfacciano in vari modi con la scheda programmabile.

Viene quindi fornita di seguito una visione schematica dei collegamenti elettrici, con annesse considerazioni pratiche.

- **Display OLED SSD1306:** collegato direttamente ai 4 pin femmina specificamente predisposti sulla board; il display **si connette al bus I<sup>2</sup>C sui pin GPIO0 e GPIO1** dell’RP2040.
- **Modulo bussola:** collegato all’header JST a 4 pin per l’espansione del bus I<sup>2</sup>C, indicato come “*I2C\_EXT*”.
- **Servomotori:** ogni servo occupa una delle tre serie di pin predisposti sulla scheda, indicati come “**SERVO**”.
- **Pannello fotovoltaico:** come delineato in precedenza, la lettura di una tensione ai capi del pannello richiede una connessione ai pad predisposti per il sensore di temperatura. Questa **viene effettuata saldando due cavi alla scheda**, sul pad 1 e 3 (rispettivamente V<sub>DD</sub> e GND riferendosi al footprint del sensore), che verranno connessi rispettivamente al polo positivo e negativo del pannello. È inoltre possibile saldare a questi ultimi due dei pin standard da 0.1” per facilitare i collegamenti.

## Schema complessivo di collegamento



L'**interfaccia utente** è presentata all'operatore attraverso il **display** SSD1306; da qui è possibile visualizzare le varie opzioni del menù e **interagire** con esso (e di conseguenza con il sistema “SunScanner”) utilizzando i **pulsanti** integrati nella scheda.

## Struttura del menù

Il menù, con il quale è possibile navigare fluidamente tra le varie modalità di utilizzo del sistema, si articola su 3 livelli:

- **Primo livello** - principale

Voci primarie di navigazione quali: **Specs, GPS, Move, History**

- **Secondo livello** - funzioni interne

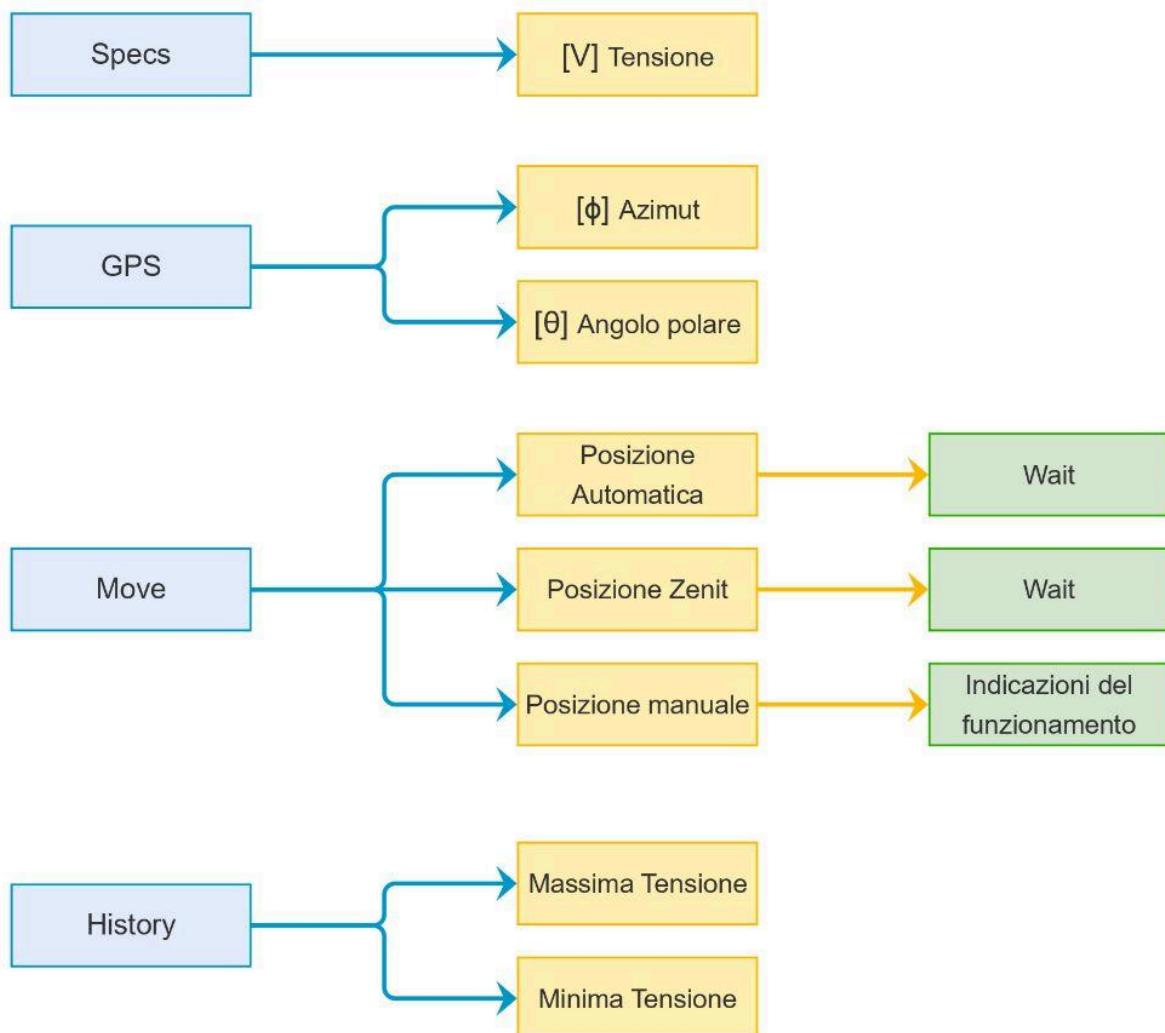
Per ogni opzione di navigazione principale sono previste da una a tre alternative

- **Terzo livello** - Opzioni di movimento della struttura

Menù di primo livello

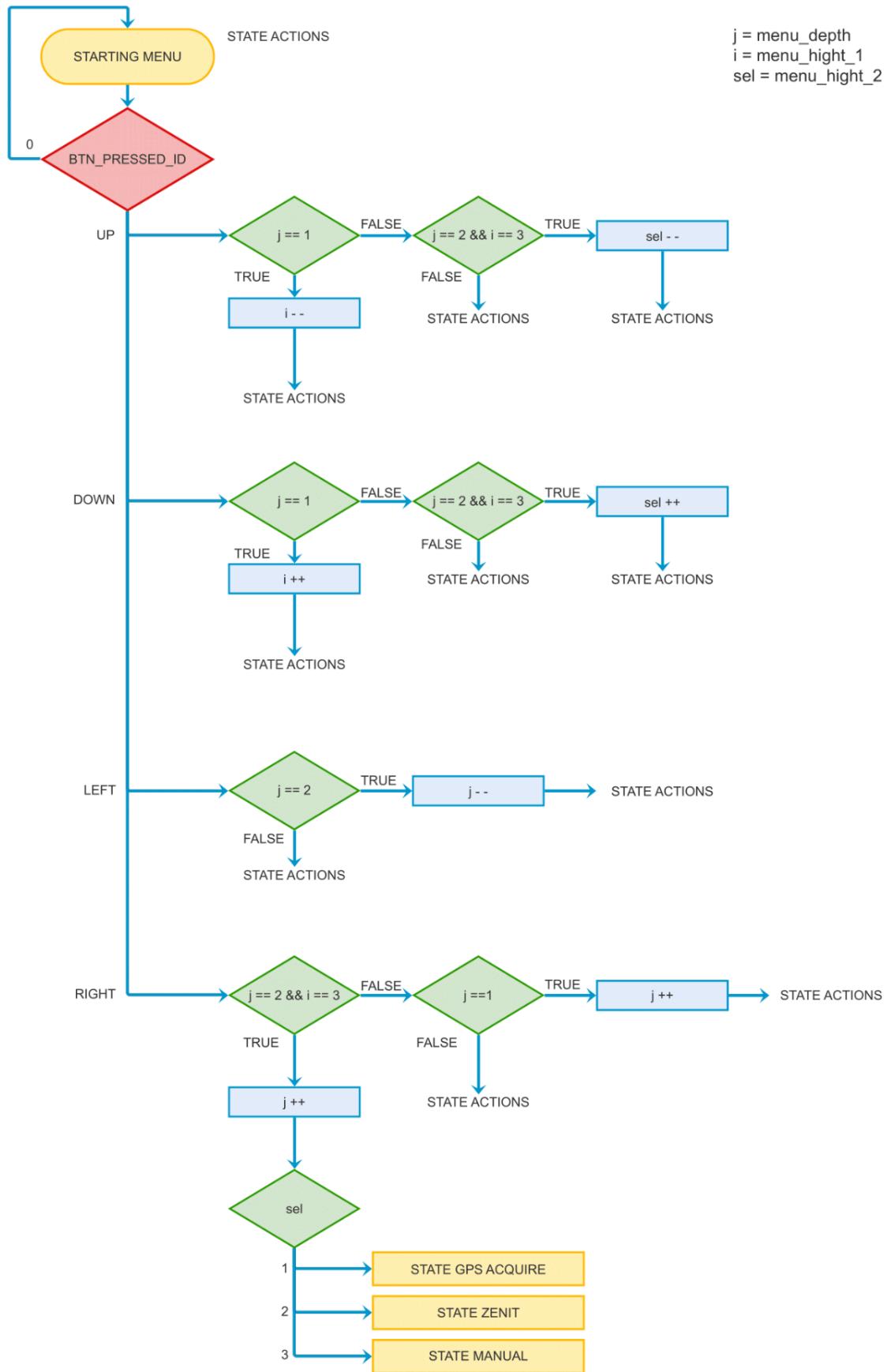
Menù di secondo livello

Menù di terzo livello

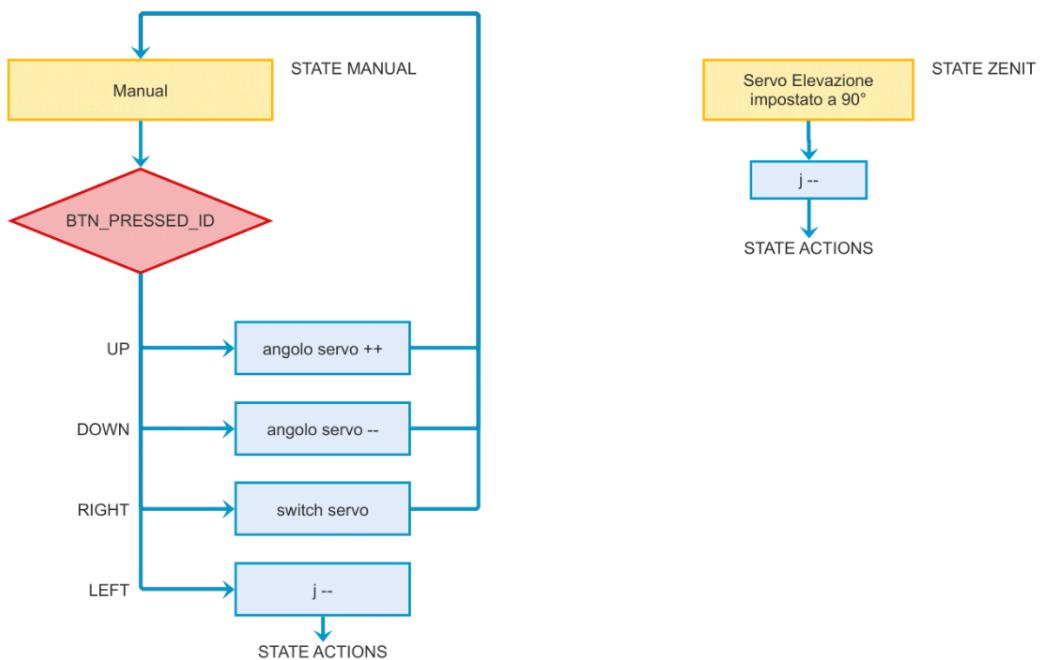
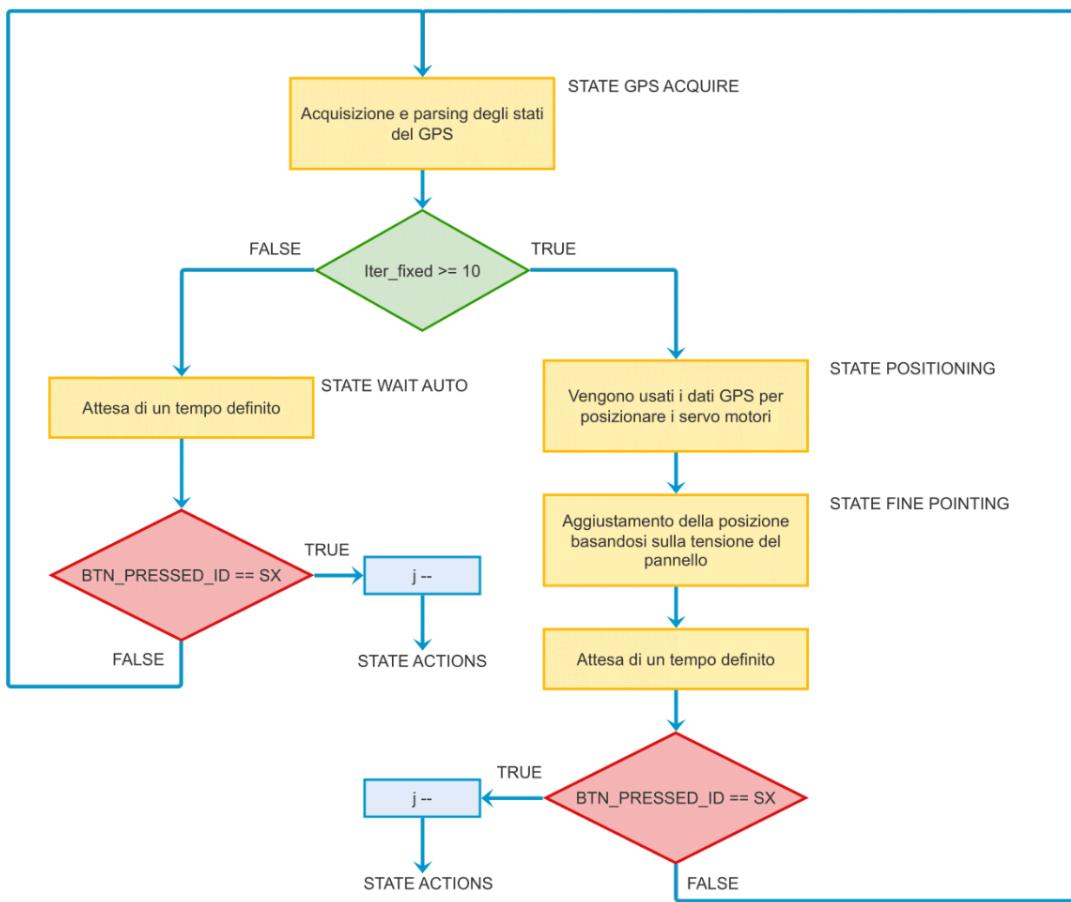


Per l'implementazione e funzionalità del display fare riferimento alla sezione 9 - Gerardo Chianucci.

## Diagramma di flusso - Navigazione nel menù

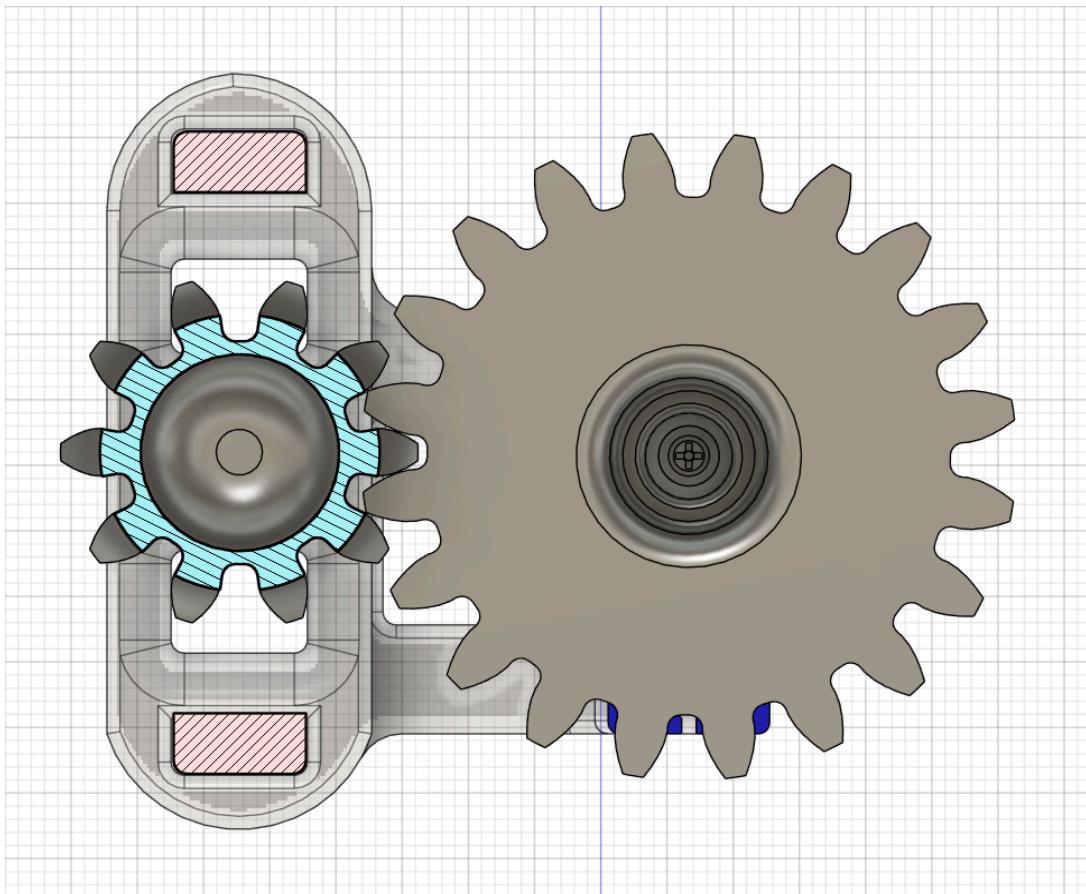


## Diagramma di flusso - Funzionamento del sistema



## 6 - Struttura - Modello 3D

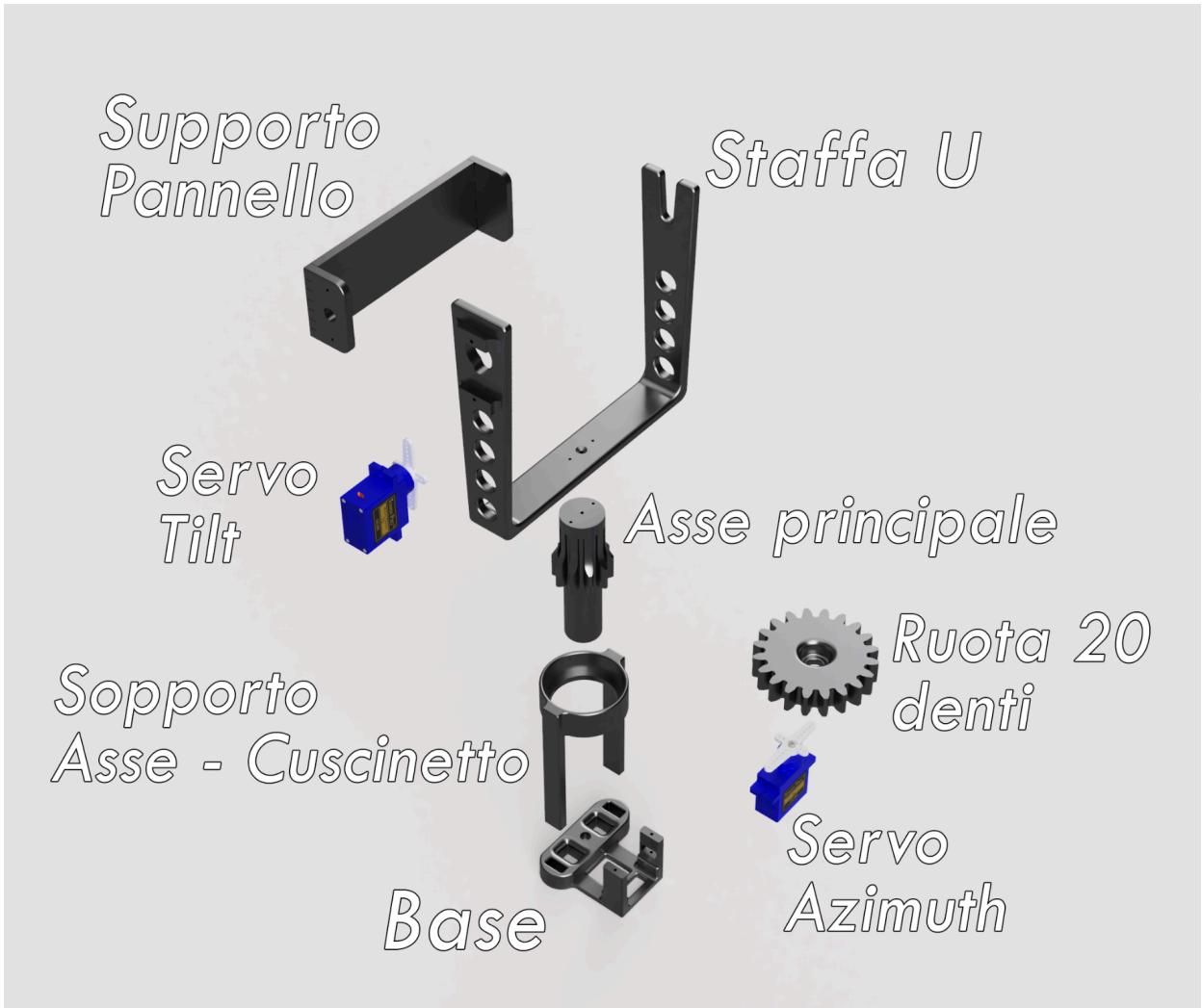
La struttura realizzata con Fusion360 è composta da un **asse principale** (mostrato a sx in figura sotto) sul quale è presente una ruota dentata a 10 denti che viene fatto ruotare da un servo dove è collocata una da 20 denti, ottenendo così un rapporto **2:1** sull'asse principale. Ciò è stato realizzato in modo tale da poter ruotare di **360 gradi** anche con la limitazione del **servo** utilizzato da 180 gradi. L'asse principale ruota su un perno sottostante e un cuscinetto a sfere nella parte superiore. Sopra di esso è presente una struttura a U dove con un secondo servo viene fatto ruotare il pannello solare.



A sx asse principale, a dx ruota dentata del servo.

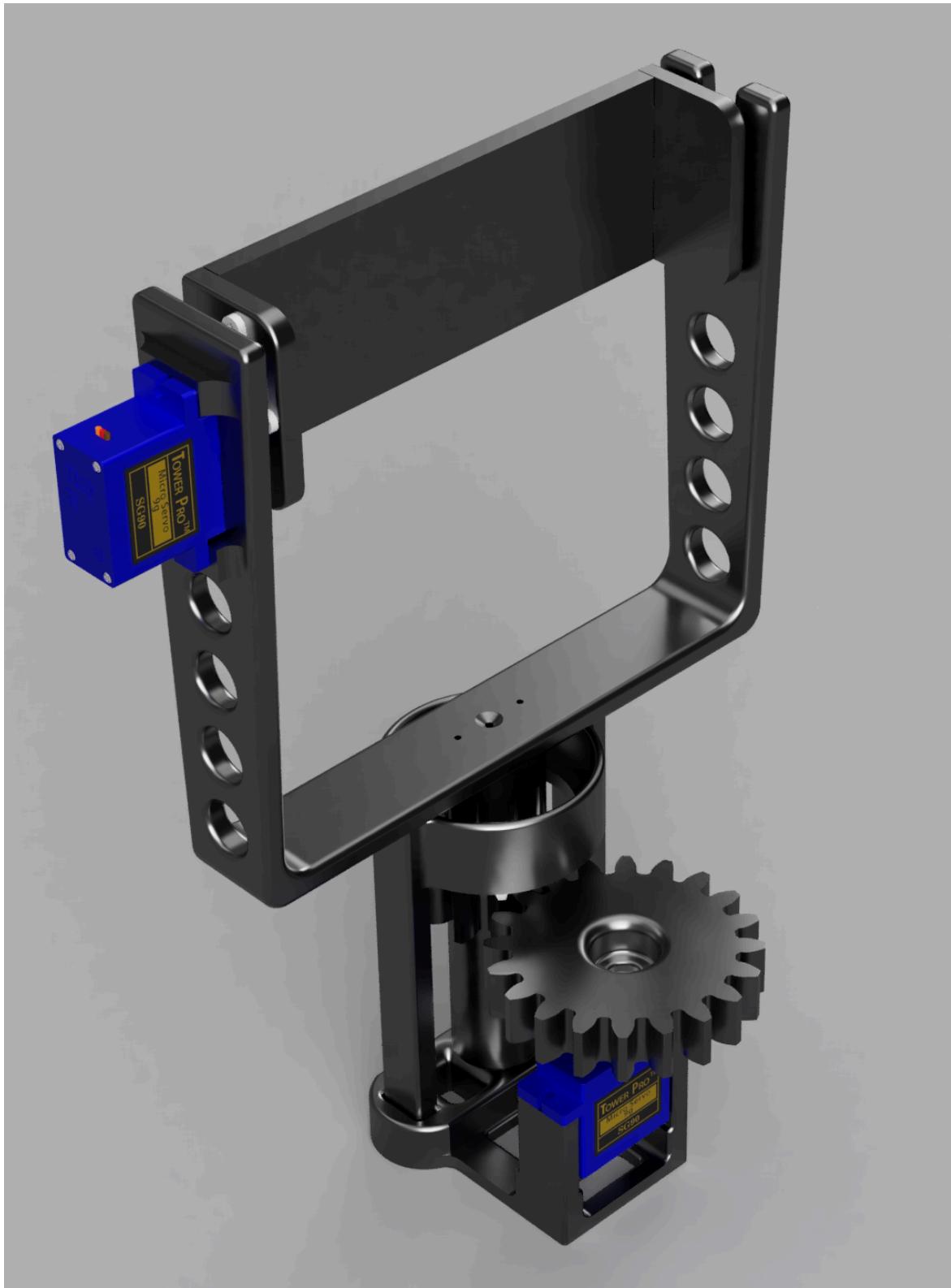
Con il primo servo, nella parte inferiore, viene fatto ruotare l'asse in modo tale da regolare l'**azimuth** (*pan*) del pannello solare, con il secondo posto nella parte superiore viene regolato invece l'**angolo polare** (*tilt*).

I pezzi sono stati successivamente prodotti con una stampante 3D tranne la base di appoggio a forma circolare e il piano orizzontale su cui poggia il pannello (realizzati in MDF con l'ausilio di una tagliatrice a laser).



Visuale esplosa della struttura sopra.

Un **cuscinetto a sfera** (non presente nei render) montato nell'alloggiamento circolare facilmente individuabile nella visuale esplosa, facilita la rotazione dell'asse principale su cui è montata la forcella, oltre ad agire da peso per stabilizzare la struttura.



Render finale della struttura.

## 7 - Segnalazione luminosa - Alert di Stato

Sulla scheda sono presenti **8 led** indirizzabili ( 2 per lato ) che sono utilizzati per dare un feedback diretto alla persona utilizzante il sistema, segnalando quello che è lo **stato**. Sono stati implementati 2 **alert** luminosi:

- **Blink di un led** per iterazione in senso orario: segnala l'avvenuta ricezione di messaggi NMEA dal satellite agganciato; un blink corrisponde a una ricezione, mentre il colore indica lo stato dell'agganciamento (verde -> fixed, arancione -> not fixed).
- Triplo **blink di tutti i led**: segnala la fine della fase di acquisizione dei dati GPS; come per i blink singoli, il colore (verde o arancione) riflette lo stato complessivo dell'agganciamento.

Per l'implementazione e funzionalità del led fare riferimento alla sezione 9 - *Emilian Manole*.

## 8 - PlatformIO e Arduino - Pico

La scelta di un ambiente di sviluppo software **verteva sulla sua facilità d'utilizzo**, sulla **flessibilità** in quanto sono state utilizzate diverse piattaforme hardware per testare le varie funzionalità del sistema, quali **Raspberry Pi Pico** e Development Kits basati su **ESP32** e sulla portabilità. I membri del team dispongono di **diversi ambienti di lavoro** sui quali eseguire gli applicativi. Alla luce di queste considerazioni abbiamo deciso di utilizzare congiuntamente **Arduino IDE**, il quale riteniamo non abbia bisogno di introduzioni e Visual Studio Code con l'estensione PlatformIO.

PlatformIO è un ecosistema che include un IDE **configurabile su Visual Studio Code** (il popolare editor di codice sorgente sviluppato da Microsoft) ed è stato concepito per essere utilizzabile con un vasto numero di piattaforme hardware. **Supporta numerosi framework**, un gran numero di board disponibili sul mercato (con la possibilità di aggiungere delle schede personalizzate) e più di diecimila librerie da integrare nei propri progetti. A **differenza dell'IDE di Arduino non presenta la stessa immediatezza di configurazione**, ma è superiore in termini di flessibilità, personalizzazione e ricchezza di funzionalità professionali (fondamentale il supporto nativo per Git e Github fornito da VS Code).

Di seguito vedremo il **processo per configurare l'ambiente di sviluppo** di PlatformIO all'interno di VS Code (*si fa riferimento alla guida ufficiale*):

1. **Installare Visual Studio Code** e, opzionalmente, l'estensione Serial Monitor di Microsoft per facilitare il debug delle comunicazioni seriali.
2. Aprire il VS Code **package manager** (label "Estensioni").
3. Cercare "PlatformIO IDE" e **scaricare il pacchetto ufficiale**.
4. Dalla home dell'IDE è possibile configurare un nuovo progetto, selezionando:
  - a. La board su cui stiamo lavorando (ad esempio, **Raspberry Pi Pico** per testing).
  - b. Il **framework** (ad esempio, Arduino).
  - c. Il percorso dove salvare il progetto.

All'interno della cartella di progetto i file sorgente andranno creati nella sottocartella “src” e gli header in “include”.

## **Arduino-pico**

Per poter prototipare il codice su schede basate sul microcontrollore RP2040 è **necessario configurare i nostri progetti PlatformIO** perché utilizzino il **framework arduino-pico** sviluppato da Earl E. Philhower (*si fa riferimento alla guida ufficiale, seguire i passaggi aggiuntivi per configurazione su macchina Windows*).

Arduino-pico è definita come una **variante del framework Arduino**, strutturata appositamente per programmare schede Raspberry Pi Pico: essa infatti integra le funzioni di basso-medio livello del pico-sdk (Software Development Kit) con i metodi propri di Arduino. Riteniamo che questo sia il modo migliore per sviluppare un'**applicazione general-purpose** per schede basate sul microcontrollore **RP2040** come la nostra, in quanto con la configurazione corretta (ad esempio specificando le funzioni associate ai vari pin) fornisce strumenti tanto potenti quanto facili da utilizzare sotto forma di **metodi e funzioni**.

Per una corretta configurazione è richiesta **l'ultima versione di git installata** sul nostro dispositivo; successivamente si dovrà modificare il file di inizializzazione “platformio.ini” nella directory del nostro progetto come segue:

Unset

```
[env:pico]
platform = https://github.com/maxgerhardt/platform-raspberrypi.git
board = pico
framework = arduino
board_build.core = earlephilhower
```

Fatto ciò **basterà aspettare il download della repository** e la riconfigurazione automatica del progetto.

A prescindere dall'approccio scelto per programmare la scheda, il metodo più efficiente e ottimizzato rimane l'utilizzo diretto delle funzioni del pico-sdk, in quanto vanno ad agire direttamente sui registri.

## 9 - Descrizione del codice

**Alessandro Bianchi Ceriani** - funzione per la validazione del checksum di una qualsiasi frase in formato NMEA 0183.

C/C++

```
bool validateChecksum(std::string mess_s, std::string& check_s) {  
    /*  
     * Flusso di lavoro per calcolare il checksum NMEA per un dato messaggio:  
     * 1. calcolare lo XOR di ogni carattere del messaggio compreso tra '$' e '*' con se  
     * stesso  
     * 2. dividere lo XOR in due nibble (campi di 4 bit) (che corrispondono ai due caratteri  
     * del checksum)  
     * 3. convertire i nibble nei caratteri esadecimali checkByte1 e checkByte2  
     * 4. confrontare i due caratteri della checksum con i checkByte calcolati  
    */  
  
    uint8_t XOR = 0;           // variabile che conterrà il risultato dell'XOR  
    size_t limit = mess_s.find('*'); // limit punta alla prima occorrenza di '*'  
    for (size_t i = 1; i < limit; ++i) { // inizio a i = 1 per saltare '$'  
        XOR ^= mess_s[i];          // computazione XOR  
    }  
  
    uint8_t nibble1 = (XOR & 0xF0) >> 4; // separazione dei nibble  
    uint8_t nibble2 = XOR & 0x0F;  
  
    uint8_t checkByte1 = (nibble1 <= 0x09) ? (nibble1 + '0') : (nibble1 - 10 + 'A'); // conversione  
    dei nibble in valori esadecimali  
    uint8_t checkByte2 = (nibble2 <= 0x09) ? (nibble2 + '0') : (nibble2 - 10 + 'A');  
  
    return (checkByte1 == check_s[0] || checkByte2 == check_s[1]); // ritorna vero se entrambi i  
    caratteri del checksum corrispondono a quelli computati  
}
```

La struttura delle frasi NMEA 0183 è la seguente: **\$aabbb,df1,df2,df3\*hh<CR><LF>**

Di seguito una breve spiegazione dei vari campi:

- \$ -> carattere che denota l'inizio del messaggio
- aabbb -> **sentence ID** (identificatore del messaggio); aa è l'identificatore del GNSS, bbb è il codice del tipo di messaggio (i più importanti per la navigazione a terra e la geolocalizzazione sono GGA, GLL, RMC)
- df1,df2... -> **data fields** (campi di dato); il numero di data fields dipende dal tipo di frase. I campi sono delimitati da ',' (tipo di formattazione detto CSV - Comma Separated Values)
- hh -> **checksum** a doppio valore esadecimale

- <CR><LF> -> carriage return e line feed; sono i **terminatori** del messaggio e facilitano la visualizzazione, soprattutto se si considera che il protocollo NMEA è stato ideato per comunicare dati puramente testuali.

I messaggi NMEA 0183 contano al massimo 80 caratteri, senza considerare i terminatori. Per la nostra applicazione abbiamo deciso di focalizzarci sui **messaggi RMC** (Recommended Minimum navigation information) in quanto forniscono i **dati essenziali** ai nostri scopi (data, ora, posizione tra gli altri). La ricezione di messaggi di altri tipi (GGA, GSV, GLL, VTG, GSA) è stata disattivata tramite l'invio di messaggi di **configurazione** (del tipo UBX-CFG-MSG), costruiti ad-hoc facendo riferimento all'**hardware integration manual** del produttore (*u-blox*). Inoltre, durante la fase di *parsing* (acquisizione e interpretazione) dei messaggi, i dati indesiderati dei messaggi RMC ricevuti vengono acquisiti ma subito scartati.

Per quanto riguarda la funzione sopra riportata, essa esegue la **validazione del checksum** di una qualsiasi sentence nello standard NMEA 0183, con le modalità indicate nel codice stesso. Congiuntamente al controllo del campo di stato (*status field*), costituisce un efficace meccanismo di verifica della “bontà” di un messaggio.

**Gerardo Chianucci** - di seguito sono riportate alcune funzioni che servono per stampare sul display OLED - SSD1306 le grafiche in bitmap con il relativo testo.

C/C++

```
//funzione che posiziona sia testo che bitmap sullo schermo
void drawMenuItem(int x, int y, const unsigned char *bitmap, char* name) {
    display.drawBitmap(x, y, bitmap, 16, 16, WHITE);
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(x + 28, y + 3);
    display.print(name);
    display.display();
}

-----
//Menu di primo livello
void drawMenu1(int i) {
    display.clearDisplay();
    switch (i) {
        case 1:
            drawMenuItem(4, 2, bitmap_history, home_menu_names[3]);
            display.drawRect(1, 22, 126, 20, WHITE);
            drawMenuItem(4, 24, bitmap_specs, home_menu_names[0]);
            drawMenuItem(4, 48, bitmap_GPS, home_menu_names[1]);
            break;

        case 2:
            drawMenuItem(4, 2, bitmap_specs, home_menu_names[0]);
            display.drawRect(1, 22, 126, 20, WHITE);
            drawMenuItem(4, 24, bitmap_GPS, home_menu_names[1]);
            drawMenuItem(4, 48, bitmap_move, home_menu_names[2]);
            break;

        case 3:
            drawMenuItem(4, 2, bitmap_GPS, home_menu_names[1]);
            display.drawRect(1, 22, 126, 20, WHITE);
            drawMenuItem(4, 24, bitmap_move, home_menu_names[2]);
            drawMenuItem(4, 48, bitmap_history, home_menu_names[3]);
            break;

        case 4:
            drawMenuItem(4, 2, bitmap_move, home_menu_names[2]);
            display.drawRect(1, 22, 126, 20, WHITE);
            drawMenuItem(4, 24, bitmap_history, home_menu_names[3]);
            drawMenuItem(4, 48, bitmap_specs, home_menu_names[0]);
            break;
    }
}
```

Questo codice disegna il menu sullo **schermo SSD1306 128x64 pixel**. La funzione drawMenuItem viene utilizzata per **disegnare ogni singolo elemento del menu**, che

**consiste in un'icona** (rappresentata come una bitmap) **e un nome associato**. Questa funzione prende le coordinate x e y dove l'elemento deve essere disegnato, la bitmap dell'icona, e il nome da visualizzare accanto all'icona.



Schermate del menù di primo livello - secondo le dimensioni dello schermo 128x32 pixel

La funzione **drawMenu1**, invece **stampa sullo schermo tutto il menù di primo livello**. Viene pulito lo schermo tramite il comando `display.clearDisplay()`; e poi disegna gli elementi del menu in base al valore passato come argomento ‘i’ alla pressione del tasto UP - DOWN.

Per far visualizzare sullo schermo le informazioni necessarie far muovere il pannello fotovoltaico e informative sono state incluse le librerie: `#include <Adafruit_GFX.h>` e `#include <Adafruit_SSD1306.h>` facendo in modo che vengano **visualizzati sia il testo sia le immagini (bitmap) sullo schermo**.

Per le icone invece è stato utilizzato il programma di editing di immagini: **Photoshop** realizzando un canvas impostato con la dimensione voluta (in questo caso **16x16 px**), una volta **pensata e creata l'icona** è stata **convertita con un tool** online di conversione dal formato PNG al formato Bitmap con il seguente sito: [image2cpp](#).



Icône all'interno del menù - dimensioni 16x16 pixel

In sintesi, questo codice serve a gestire la **visualizzazione di un menù a più livelli** su uno schermo, disegnando icone e nomi associati agli elementi del menù in base a un indice fornito utilizzando i pulsanti.

C/C++

```
//definizione lunghezza e pin della stringa di led
#include <NeoPixelConnect.h>
#define NUM_LED 8
#define PIN_RGB_LED 13

//inizializzazione della stringa di led
NeoPixelConnect p(PIN_RGB_LED, NUM_LED, pio0, 0);

//funzione per iterare sul singolo led interessato dato un messaggio in entrata (GPS)
void singleLedIfFixed(bool fixed, int num_iter) {
    num_iter%=NUM_LED;      //iterazione circolare su 8 elementi con INPUT anche >8
    if(fixed) {
        p.neoPixelSetValue(num_iter, 0, 255, 0, true);
        delay(100);
        p.neoPixelClear(true);
    } else {
        p.neoPixelSetValue(num_iter, 255, 128, 0, true);
        delay(100);
        p.neoPixelClear(true);
    }
}

-----  
//in ws2812.h

.wrap_target

bitloop:
    out x, 1           side 0 [T3 - 1] ; Side-set still takes place when instruction stalls
    jmp !x do_zero     side 1 [T1 - 1] ; Branch on the bit we shifted out. Positive pulse

do_one:
    jmp bitloop       side 1 [T2 - 1] ; Continue driving high, for a long pulse

do_zero:
    nop              side 0 [T2 - 1] ; Or drive low, for a short pulse
```

L'interfacciamento con i WS2812B avviene attraverso i **PIO** (Programmable I/O) con la libreria **“NeoPixelConnect”** basata su “**ws2812.h**” ufficiale della **raspberry-pi** ufficiale nella sezione PIO. In questo caso i led formano una “stringa” da 8 elementi consecutivi su cui si può iterare utilizzando il primo parametro delle funzioni disponibili nella libreria. Gli altri parametri riguardano il valore RGB (Red,Green,Blu) nel range di 0-255. Quando viene avviata la scheda l'RP2040 inizializza il **PIN\_RGB\_LED** 13 come PIO di output che viene aggiornato su **clock** del microcontrollore. Il controllo dei PIO viene fatto attraverso nove **istruzioni assembly** “**JMP**, **WAIT**, **IN**, **OUT**, **PUSH**, **PULL**, **MOV**, **IRQ**, **SET**”. In questo caso viene utilizzato “**OUT**” per mandare il bit sul PIN fino a quando non viene usato “**JUMP**” per cambiare l'istruzione (istruzioni che sono esplicitamente dichiarate per quanti clock durare).

**Daniele Visentin** - `set_servo` imposta il servo su un determinato angolo alpha mentre `auto_positioning` posiziona i due servo di controllo verso il sole.

C/C++

```
/*Funzione per posizionare il servomotore*/
void Servo::set_servo(double angle){
    double starting_alpha=alpha;
    int ms;

    alpha = rebound(angle); //sistema angoli fuori dal range
    duty_alpha = duty_calc(alpha); //calcola quanti secondi deve rimanere acceso
    ms = calc_iter_PWM(starting_alpha,alpha); //iterazioni per impostare il servo
    for(int i = 0; i < ms; ++i){
        digitalWrite(pin,HIGH);
        delayMicroseconds(duty_alpha);
        digitalWrite(pin,LOW);
        delayMicroseconds(20000-duty_alpha);
    }
}

-----
void ServoController::auto_positioning(tm* time_info_ptr, double Lat, double Lon,
double Alt, double compass_val, double* Az, double* El){
    int azpos;
    SolarAzEl(time_info_ptr, Lat, Lon, Alt, Az, El); //calcolo di azim e elev
    azpos=(Az);
    //aggiustamenti per far tornare gli angoli
    if(compass_val>180){
        azpos+=(360-compass_val);
    }else{
        azpos-=compass_val;
    }
    /* these adjustments work if the 0 angle of the servoAz face same direction of */
    /* compass */
    servoAz.set_servo(azpos);
    servoEl.set_servo(*(El));
}
```

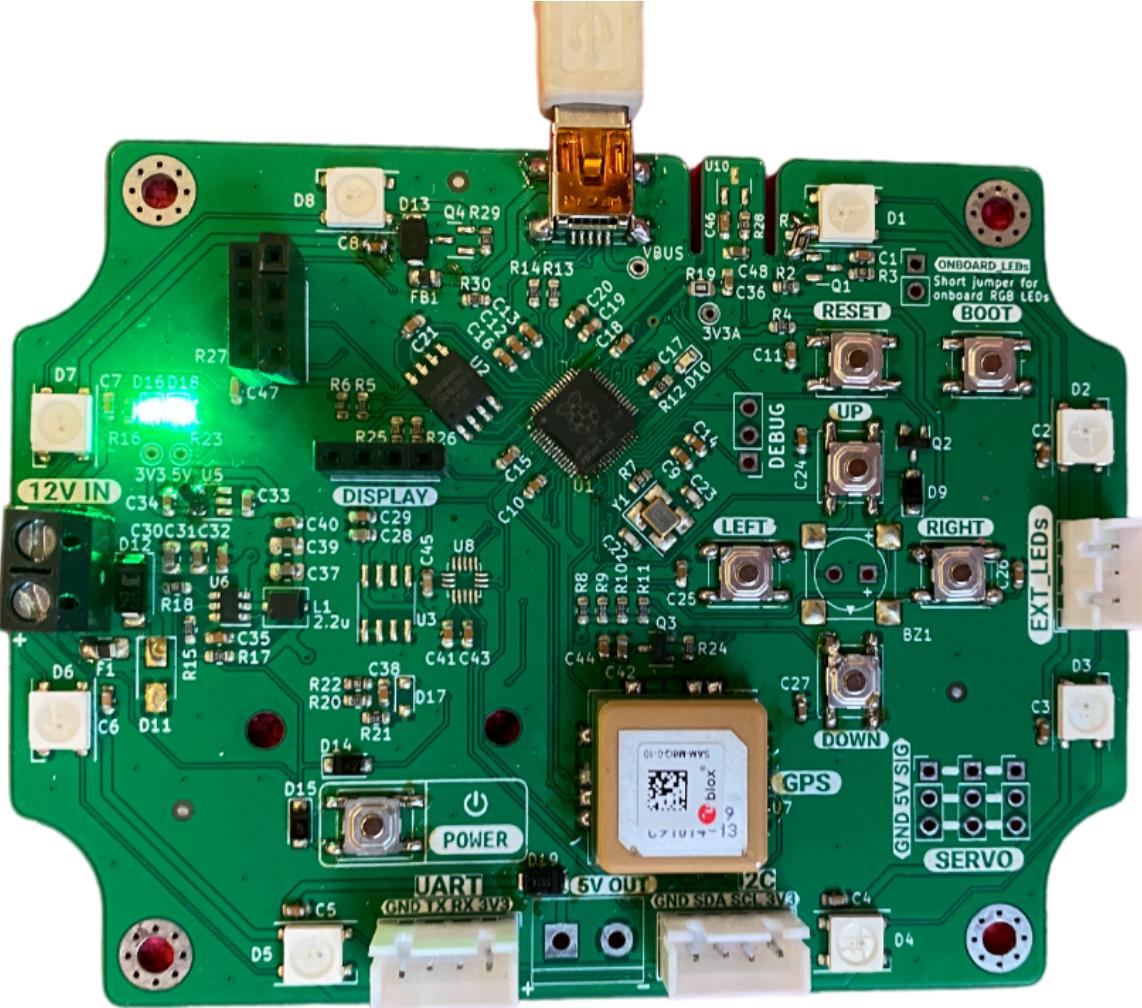
I **servomotori** utilizzati per muovere il pannello devono essere pilotati mediante una **pwm** a **50 Hz**, con duty-cycle variabile tra **500ms** e **2500ms** ( $0^\circ \rightarrow 180^\circ$ ).

Il **segnale** di **controllo** servo è generato mediante la funzione **set\_servo**, la quale dato un **angolo** come **input** genera un segnale **pwm** in **uscita** verso i servo. Questo avviene tante volte quanto basta a posizionare il servo.

La funzione **auto\_positioning** prima calcola le **coordinate altazimutali** del sole, poi modifica l'angolo di **azimut** basandosi sui dati della **bussola**

## 10 - Conclusioni

La foto seguente rappresenta la scheda finita e funzionante in tutte le sue parti.



Durante questo laboratorio, abbiamo affrontato **l'intero ciclo di sviluppo di una scheda**, partendo dall'idea iniziale del suo utilizzo, passando per la progettazione mediante l'uso di software CAD, fino ad arrivare alla fase di prototipazione e **creazione definitiva della scheda**.

L'ultimo passo è stato quello di **sviluppare il software** necessario per gestire tutte le periferiche e le funzionalità della scheda, **ogni componente del gruppo si è occupato di una parte specifica** per poi unire il tutto e creare il progetto definitivo.

Le maggiori problematiche che abbiamo riscontrato **sono state via software**, come creare un giusto parsing per far funzionare correttamente il GPS, capire i posizionamenti dei servomotori in base alle coordinate del GPS.

In sintesi, **l'obiettivo di questo laboratorio è stato quello di creare e realizzare** una scheda funzionante. Una volta programmata, questa scheda è in grado di fornire le indicazioni richieste, rappresentando quindi un **importante strumento applicativo**.

# 11 - Bibliografia e Sitografia

- Documentazione RP2040  
<https://www.raspberrypi.com/documentation/microcontrollers/rp2040.html>
- Sito ufficiale di PlatformIO  
<https://platformio.org/>
- Documentazione ufficiale di PlatformIO IDE  
<https://docs.platformio.org/en/latest/integration/ide/pioide.html>
- Documentazione ufficiale di Arduino Pico, integrazione con PlatformIO IDE  
<https://arduino-pico.readthedocs.io/en/latest/platformio.html>
- Hardware integration manual per SAM-M8Q  
[https://content.u-blox.com/sites/default/files/SAM-M8Q\\_HardwareIntegrationManual\\_%28UBX-16018358%29.pdf](https://content.u-blox.com/sites/default/files/SAM-M8Q_HardwareIntegrationManual_%28UBX-16018358%29.pdf)
- Conversione immagini da PNG a Bitmap  
<https://javl.github.io/image2cpp/>
- Datasheet dei componenti, schematici e resto del materiale fornito dai professori a lezione

---

<b>Alessandro Bianchi Ceriani</b>	[226663]
<b>Gerardo Chianucci</b>	[227071]
<b>Emilian Manole</b>	[228421]
<b>Daniele Visentin</b>	[227506]