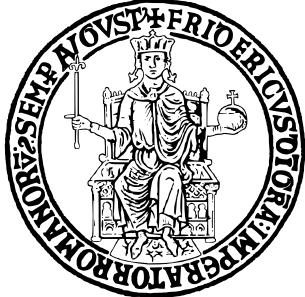


UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II



SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE
DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN INGEGNERIA
MECCATRONICA

TESI SPERIMENTALE IN ROBOTICA E AUTOMAZIONE
INDUSTRIALE

ARCHIMEDE

Relatore

Prof. Vincenzo LIPPIELLO

Candidato

Giulio VESTRI

P27/000143

Correlatore

Dott. Alessandro De Crescenzo

Anno Accademico 2023–2024

Indice

1	Introduzione	4
2	Struttura fisica del sistema	6
2.1	Elementi strutturali acquistati	6
2.2	Elementi progettati in CAD	7
3	Tracciamento della Posizione Solare	9
3.1	Librerie usate	9
3.2	Sviluppi futuri	10
4	Sensori e Attuatori	11
4.1	Sensori	11
4.2	Scelta dei motori e driver	12
5	Selezione Target	13
5.1	Controllo tramite Pulsantiera	13
5.2	Orientamento desiderato	15
6	Sistema di Controllo	16
6.1	Riduzione del rumore	16
6.2	PI	17
6.3	Segnale di controllo e PWM	19
7	Alimentazione	20
7.1	Batteria e ricarica	20
7.2	Modalità Low-Power	21
7.3	Modalità Notturna	21
8	Appendice	22
8.1	Schema elettrico	22
8.2	Codice sorgente	22
8.3	Riferimenti immagini	27
8.4	Riferimenti delle Librerie	27
8.5	Schema a blocchi	28

Capitolo 1

Introduzione

Il progetto di tirocinio aziendale, svolto presso Neabotics dal 7/2/2024 al 10/6/2024, si è concentrato sulla realizzazione di un pannello riflettente in grado di modificare periodicamente il proprio orientamento durante l'arco della giornata per riflettere i raggi solari verso un punto specifico desiderato, denominato Archimede e riportato nella Figura 1.1.

Neabotics è una startup innovativa e spin-off del laboratorio di robotica PRISMA Lab dell'Università Federico II di Napoli, specializzata nello sviluppo di soluzioni robotiche avanzate per l'ispezione e la manutenzione di infrastrutture nei settori dell'energia e dei trasporti. Durante i mesi di realizzazione del progetto, ho lavorato in un ambiente altamente stimolante e creativo, con l'opportunità di affrontare sfide nell'ambito dell'elettronica, della meccanica, dell'informatica e del controllo. Ho scelto questo progetto tra varie opzioni poiché, mi permetteva di realizzare un sistema completo che integrasse le diverse discipline della mia formazione.

Il sistema ideato richiedeva un movimento attorno a due assi, definito dagli angoli di azimuth ed elevazione, garantito da due motori DC con moto-riduzione per assicurare basse velocità ed elevate coppie.

Per analizzare i dati ed effettuare il controllo è stato necessaria la scelta di un microcontrollore. Ho scelto un Arduino Nano (Figura 1.2), una scheda elettronica basata sul microcontrollore ATmega328 che può lavorare fino ad una frequenza di 16 MHz. Il sistema include due sensori: un IMU solidale allo specchio per rilevare l'angolazione rispetto all'orizzonte (elevazione) e un magnetometro solidale alla struttura per rilevare l'angolo rispetto al nord magnetico terrestre (azimuth).

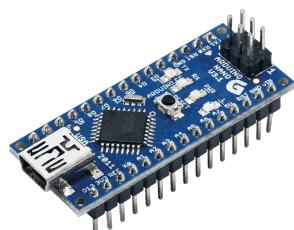


Figura 1.2:



Figura 1.1: Archimede

Gli angoli desiderati per la riflessione solare sono calcolati come media tra gli angoli della posizione del target e quelli della posizione del sole (Figura 1.3).

I segnali dei sensori sono filtrati attraverso un filtro digitale passa basso per ridurre il rumore. Il controllo di posizione è gestito da un PID, con il segnale di controllo filtrato digitalmente per evitare picchi che potrebbero danneggiare gli attuatori.

Infine, il sistema è dotato di una funzione di risparmio energetico, che permette di entrare periodicamente in modalità low-power per minimizzare gli sprechi di energia.

Tutte le considerazioni successive si basano sul sistema di riferimento preso in considerazione nell'immagine successiva (Figura 1.4) con:

- l'asse X (blu) sovrapposto alla normale dello specchio;
- l'asse Y (verde) che si sviluppa lungo l'asse longitudinale del sistema e
- l'asse Z (rosso) ortogonale agli ultimi due.

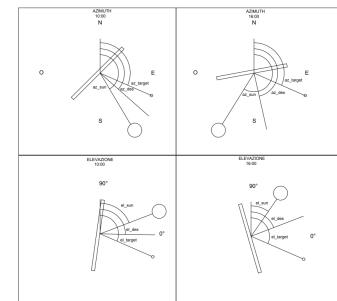


Figura 1.3:

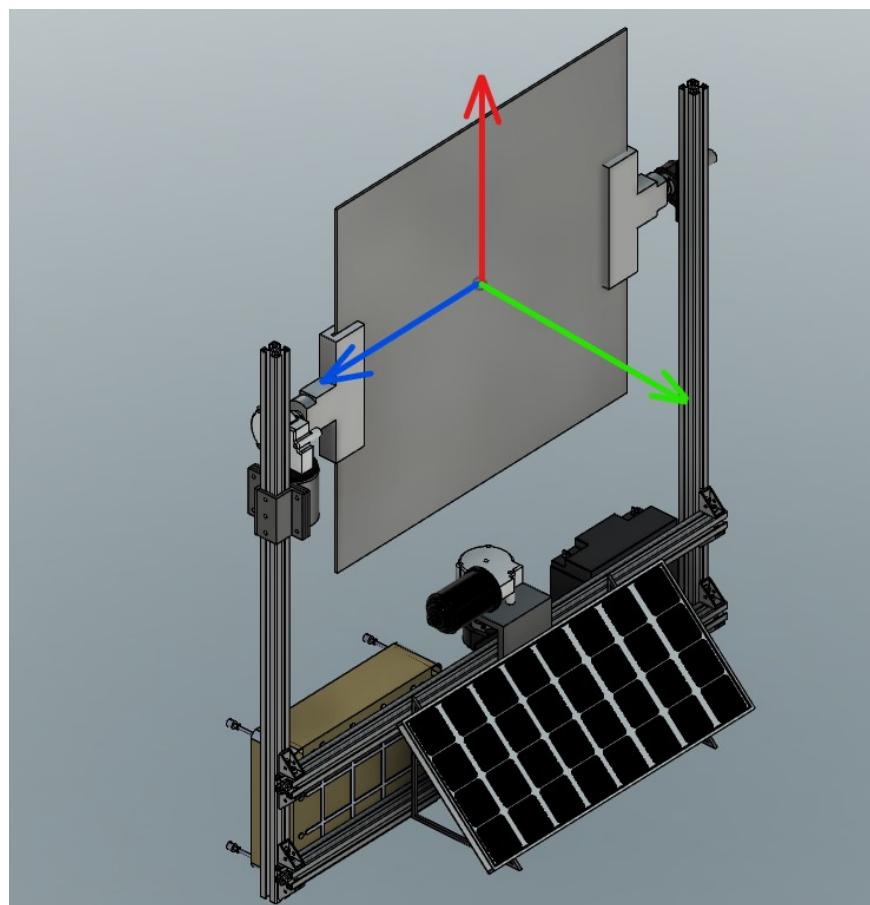


Figura 1.4: Sistema di riferimento adottato

Capitolo 2

Struttura fisica del sistema

La struttura del sistema doveva essere progettata in modo tale da garantire la rotazione su due assi perpendicolari e riuscire a reggere il peso di uno specchio che, da specifiche, doveva essere di dimensioni 50x60 cm. Per permettere alla struttura di essere leggera e robusta, ho optato per l'acquisto di elementi in alluminio e per la realizzazione delle componenti più complesse attraverso l'uso della stampa 3D.

2.1 Elementi strutturali acquistati

Sono state acquistate una serie di componenti sul sito di MISUMI:

- Profilati di alluminio HFS6-3030-900 (Figura 2.1)

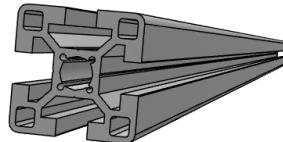


Figura 2.1:

- Staffe angolari HBLFSN6-5-C-SEP
- Supporti cuscinetto CPDR20

Altre componenti sono state acquistate su Amazon:

- Specchio 500x600 mm
- Motori DC
- Driver Motori

- Batteria
- Pannello solare
- Scatola elettrica per esterni

Ed altre sono state rinvenute in laboratorio:

- Alberi ϕ 25mm
- Step-down 5v
- 3.3v 5v level shifter
- Arduino Nano

2.2 Elementi progettati in CAD

Per permettere il montaggio di tutti gli elementi acquistati sulla struttura, è stato necessario progettare dei CAD 3D con il software Fusion360:

- Box per motore azimut (Figura 2.2)

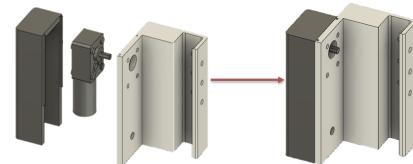


Figura 2.2:

- Box per motore elevazione (Figura 2.3)

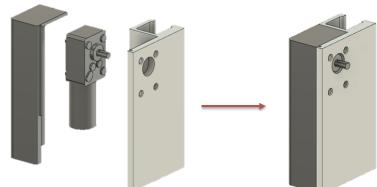


Figura 2.3:

- Box IMU e sostegno specchio
- Staffa per pannello solare
- Blocco di accoppiamento motore-profilato



Figura 2.4:



Figura 2.5:

- Blocco di accoppiamento profilato-albero di alluminio (Figura 2.4)
- Blocco di accoppiamento motore-albero di alluminio (Figura 2.5)

Prima del montaggio ho provato ad effettuare un pre-assemblaggio direttamente con il CAD, modificato durante il processo di costruzione (Figura 2.7).

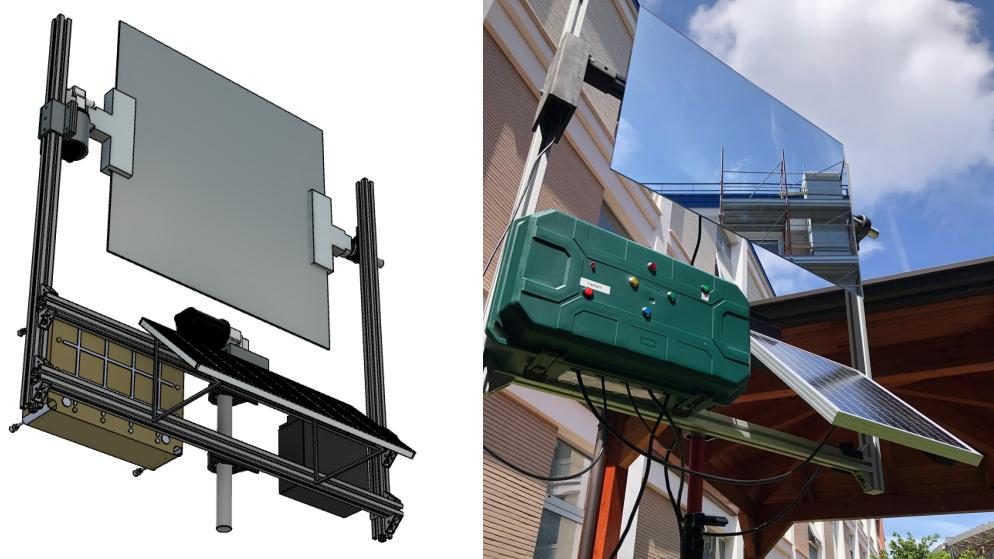


Figura 2.6: A sinistra, il pre assemblaggio effettuato sul CAD Fusion360; a destra il montaggio finale

Capitolo 3

Tracciamento della Posizione Solare

Il primissimo problema da risolvere era il riconoscimento della posizione del sole all'interno della sfera celeste. La soluzione adottata prevede l'utilizzo di una libreria che permette, attraverso l'uso di algoritmi astronomici, di calcolare la posizione del sole in Azimuth (angolo rispetto al nord) ed elevazione (angolo rispetto all'orizzonte). Questa libreria richiede informazioni relative alla posizione geografica e temporale del sistema.

3.1 Librerie usate

Per conoscere la posizione del sole in tempo reale, ho usato la libreria SolarCalculator.h, nello specifico la funzione calcHorizontalCoordinates(), che, attraverso la conoscenza di: latitudine, longitudine, data e ora convertiti in Universal Coordinated Time (UTC), restituisce in tempo reale la posizione del sole rispetto ad un osservatore in Azimuth ed Elevazione. Questi angoli rappresentano rispettivamente l'angolo formato rispetto al nord compreso nell'intervallo $[0^\circ ; 360^\circ]$ e l'angolo rispetto all'orizzonte compreso nell'intervallo $[+180^\circ ; -180^\circ]$ considerando a 0° la linea dell'orizzonte

```
1 #include <SolarCalculator.h>
2 ...
3 double az_sun, el_sun;
4 ...
5 calcHorizontalCoordinates(utc, latitude, longitude, az_sun, el_sun);
6 ...
```

I dati relativi alla posizione, latitudine e longitudine, sono inseriti manualmente;

```
1 const double latitude = 40.828660, longitude = 14.189816;
```

mentre quelli relativi alla data e all'ora sono ricavati tramite la libreria TimeLib.h che permette di calcolare attraverso la funzione now() i secondi trascorsi rispetto all'epoca (00:00- 9/9/1970) e di convertirli in UTC.

```
1 #include <TimeLib.h>
2 ...
```

```

3 time_t utc = now();
4 ...
5 setTime(toUtc(copileTime()));

1 /*-----CALCOLO DATA E ORA-----*/
2 time_t compileTime() {
3     const uint8_t COMPILE_TIME_DELAY = 8;
4     const char *compDate = __DATE__, *compTime = __TIME__, *months = "
5         JanFebMarAprMayJunJulAugSepOctNovDec";
6     char chMon[4], *m;
7     tmElements_t tm;
8     strncpy(chMon, compDate, 3);
9     chMon[3] = '\0';
10    m = strstr(months, chMon);
11    tm.Month = ((m - months) / 3 + 1);
12    tm.Day = atoi(compDate + 4);
13    tm.Year = atoi(compDate + 7) - 1970;
14    tm.Hour = atoi(compTime);
15    tm.Minute = atoi(compTime + 3);
16    tm.Second = atoi(compTime + 6);
17    time_t t = makeTime(tm);
18    return t + COMPILE_TIME_DELAY;
19 }
20 /*-----*/
21 /*----CONVERSIONE IN UTC----*/
22 time_t toUtc(time_t local) {
23     int utc_offset = 2;
24     return local - utc_offset * 3600L;
25 }
26 /*-----*/

```

3.2 Sviluppi futuri

L'inserimento manuale di latitudine e longitudine può risultare scomodo in caso di ampi spostamenti del sistema . può essere considerato quindi come sviluppo futuro l'implementazione di un GPS che non solo fornirebbe i dati relativi alla posizione , ma anche quelli relativi alla data riducendo così il lavoro computazionale del sistema.

Capitolo 4

Sensori e Attuatori

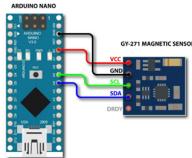
Il sistema, per poter funzionare, doveva conoscere la propria angolazione nello spazio ed essere in grado di modificarla. è stato dunque necessario scegliere due sensori che restituissero le letture dei due angoli interessati e due attuatori che permettessero al sistema di ruotare in maniera controllata.

4.1 Sensori

Ai sensori era richiesto di riconoscere Elevazione ed Azimuth.

L'azimuth è ricavato da un modulo bussola GY-271 (Figura 4.1) che monta un chip QMC583L, esso attraverso una comunicazione I2C permette, di ricavare l'orientamento. Per utilizzare correttamente il modulo GY-271 ho usato la libreria Wire.h (per permettere la comunicazione I2C) e la libreria QMC583L.h (relativa al chipset del magnetometro).

Figura 4.1:



```
1 #include <Wire.h>
2 #include <QMC5883LCompass.h>
3 ...
4 QMC5883LCompass compass;
5 ...
6 compass.getAzimuth();
```

Ottengo una lettura dell'azimuth attraverso la funzione compass.getAzimuth() che restituisce un valore compreso nell' intervallo [-180 ; +180], utilizzo perciò la funzione map() per trasporre la lettura nell' intervallo [0 ; 360] ed essere così coerente con i dati forniti dal tracciamento solare;

```
1 map(compass.getAzimuth(), -180, 180, 0, 360)
```

Salvo in fine il dato corrente in una variabile globale facendolo prima passare per un filtro digitale passabasso che vedremo nel dettaglio di seguito, per permettere la cancellazione del rumore

```
1 compass.read();
2 az_mis=LPF_az_mis.update(map(compass.getAzimuth(), -180, 180, 0, 360));
```

Per l'elevazione ho optato per un imu a 9 assi capace di restituire roll, pitch e yaw. Attraverso la lettura di uno di questi ho potuto ricavarmi il dato desiderato. Il sensore utilizzato è un WIT-IMU (Figura 4.2) per una lettura efficace dei dati ho usato la libreria JY-901, che consentiva un collegamento sia attraverso la seriale sia attraverso l'i2c. Al fine di mantenere la comunicazione seriale libera, e per non diminuire le prestazioni del sistema, ho scelto l'utilizzo del collegamento attraverso l'i2c. I dati ricavabili da questa scheda erano molteplici, al progetto serviva una lettura dell'angolo di rotazione attorno all'asse x del sistema.

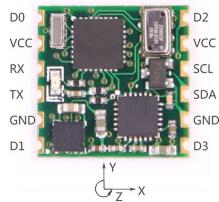


Figura 4.2:

Considerando la scheda solidale allo specchio e decisa la posizione di montaggio, si ritiene che il dato utile al progetto sia la lettura del rollio, possibile grazie alla funzione:

```
1 JY901.GetAngle();
2 el_mis=LPF_el_mis.update((float)JY901.stcAngle.Angle[0]/32768*180);
```

4.2 Scelta dei motori e driver



Figura 4.3:

Per permettere la rotazione su entrambi gli assi ho optato per la scelta di due motori DC da 12 V con vite senza fine, moto riduzione e velocità angolare massima di 10rpm (Figura 4.2), per garantire velocità controllate e coppie elevate. La corrente nominale è minore di 1.6A, con una torsione massima di 70 Kg*cm. I motori sono stati montati sulla struttura grazie a dei supporti stampati in 3d attraverso una accoppiamento diretto sugli assi di rotazione. Attraverso viti di montaggio ed elementi stampati è stato inoltre possibile effettuare l'accoppiamento albero-motore.

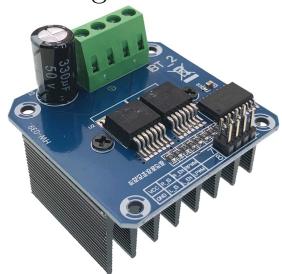


Figura 4.4:

Per pilotarli invece è stato necessario selezionare due driver che permettono di governare la velocità dei motori attraverso l'utilizzo di segnali PWM. I driver selezionati sono JZK BTS7960B (Figura 4.3) essi prendono in ingresso la tensione di alimentazione dei motori (12v), la tensione di alimentazione della logica di controllo (5v) ed i segnali PWM, moderando la potenza trasmessa ai motori in base a quest'ultimi.

Capitolo 5

Selezione Target

Per riuscire a riflettere i raggi del sole su di un punto specifico, bisogna conoscere la posizione di quest' ultimo. Per avere un sistema il più versatile possibile, ho implementato la possibilità di modificare manualmente questi parametri. Esiste quindi una funzione selezionabile in qualsiasi momento nel codice in cui attraverso dei comandi esterni, forniti da una pulsantiera, è possibile modificare elevazione ed azimuth del sistema salvando la nuova posizione nella memoria EEPROM del microcontrollore. Questi dati resteranno invariati per tutto il controllo e in caso di spegnimento.

5.1 Controllo tramite Pulsantiera

Per evitare l'utilizzo di eccessivi pin del microcontrollore, ho progettato un partitore di tensione collegato ad un singolo pin analogico di Arduino nano (Figura 5.1), premendo i pulsanti la lettura analogica varia modificando il risultato della funzione interna al firmware.

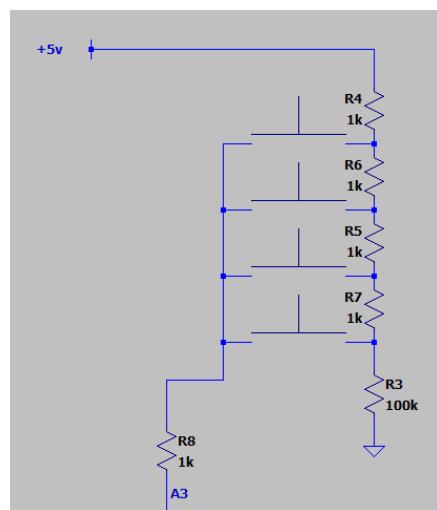


Figura 5.1: Pulsantier analogica

Ho implementato nel codice due funzioni per il funzionamento della pulsantiera: lettura_pulsante(), che permette la lettura del pin analogico e restituisce un valore diverso a seconda del pulsante premuto; e selezione_target(), che attiva i motori in corrispondenza dei pulsanti premuti e salva le nuove posizioni lette dai sensori nella memoria EEPROM.

```

1 int lettura_pulsante()
2 {
3     const int Az_up = 819, Az_dw = 612, El_up = 408 , El_dw= 203, delta = 10;
4     int Value = analogRead(A3);
5     if (Value <Az_up+delta && Value >Az_up-delta) return 0;
6     if (Value <Az_dw+delta && Value >Az_dw-delta) return 1;
7     if (Value <El_up+delta && Value >El_up-delta) return 2;
8     if (Value <El_dw+delta && Value >El_dw-delta) return 3;
9     if (Value <10+delta && Value >10-delta) return 4;
10 }
11
12 void controllo_target(){
13     if(digitalRead(PIN_TARGET)){
14         switch (lettura_pulsante()){
15             case 0:
16                 digitalWrite(DX_az, 80);
17                 digitalWrite(SX_az, 0);
18                 digitalWrite(SX_el, 0);
19                 digitalWrite(DX_el, 0);
20                 break;
21             case 1:
22                 digitalWrite(DX_az, 0);
23                 digitalWrite(SX_az, 80);
24                 digitalWrite(SX_el, 0);
25                 digitalWrite(DX_el, 0);
26                 break;
27             case 2:
28                 digitalWrite(DX_az, 0);
29                 digitalWrite(SX_az, 0);
30                 digitalWrite(DX_el, 50);
31                 digitalWrite(SX_el, 0);
32                 break;
33             case 3:
34                 digitalWrite(DX_az, 0);
35                 digitalWrite(SX_az, 0);
36                 digitalWrite(DX_el, 0);
37                 digitalWrite(SX_el, 50);
38                 break;
39             case 4:
40                 digitalWrite(DX_az, 0);
41                 digitalWrite(SX_az, 0);
42                 digitalWrite(SX_el, 0);
43                 digitalWrite(DX_el, 0);
44                 break;
45         }
46         compass.read();
47         az_mis= LPF_az_mis.update(map(compass.getAzimuth(), -180, 180, 0, 360));
48         az_des=az_mis;
49         az_target= (2*az_des)-az_sun;
50         JY901.GetAngle();
51         el_mis=LPF_el_mis.update((float)JY901.stcAngle.Angle[0]/32768*180);
52         el_des=el_mis;
53         el_target= (2*el_des)-el_sun;
54         EEPROM.put(address_az, az_target);
55         EEPROM.put(address_el, el_target);
56     }
57 }
```

Infine per attivare queste funzioni ho implementato un interruttore (PIN_TARGET) che se chiuso richiama la funzione di selezione.

5.2 Orientamento desiderato

La luce viene riflessa su uno specchio piano, il che implica che gli angoli di incidenza (θ_i) e di riflessione (θ_r) sono sempre uguali, indipendentemente dall'orientamento dello specchio (Figura 5.2). La normale alla superficie riflettente deve dunque coincidere con la bisettrice dell'angolo formato tra il raggio incidente e il raggio riflesso (Figura 5.3).

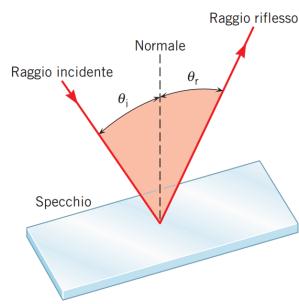


Figura 5.2:

Per garantire che la riflessione sia correttamente direzionata verso il target, è necessario dunque che la normale alla superficie riflettente coincida con la bisettrice dell'angolo formato tra la posizione del sole e quella del target. Essendo il sistema composto da due coordinate indipendenti, azimut ed elevazione, il ragionamento può essere effettuato per entrambe le misure in maniera analoga, ricavando così gli angoli desiderati. Nel codice la grandezze *az_des* e *el_des*, sono calcolate come l'angolo medio tra le misure del sole e del target, e fatti passare attraverso un filtro digitale passa basso per evitare modifiche impulsive alla lettura.

```
1 az_des = LPF_az_des.update (((az_sun + az_target) / 2.00));
2 el_des = LPF_el_des.update (((el_sun + el_target) / 2.00));
```

Questo approccio consente di ottenere gli angoli desiderati per allineare correttamente lo specchio in modo che la luce solare venga riflessa esattamente verso il target nonostante i movimenti del sole.

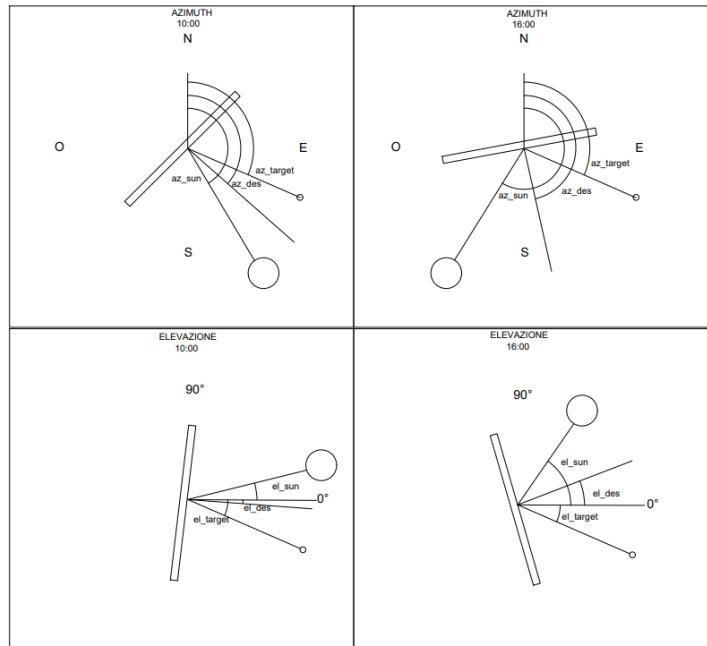


Figura 5.3: Gli angoli desiderati sono calcolati come l'angolo medio tra *_target* e *_sun*

Capitolo 6

Sistema di Controllo

6.1 Riduzione del rumore

I segnali uscenti dai sensori vanno ripuliti da interferenze aleatorie, questo processo può essere sviluppato attraverso la creazione di un filtro digitale passabasso, che permette di bloccare le alte frequenze caratteristiche del rumore. Il filtro è stato sviluppato come una classe da poter definire ed utilizzare nello script principale in maniera separata per ogni segnale. La definizione del filtro come classe permette inoltre di poterlo usare eventualmente anche in altri codici.

```
1 #ifndef _FILTER_H_
2 #define _FILTER_H_
3 #include <math.h>
4 class LPF{
5     private:
6         double _y_old;
7         double _y;
8         double _alfa;
9     public:
10        LPF(double f_c, double T_s); //Definizione Costruttore
11        double update(double x_raw);
12        void set_initial_condition (double y_0);
13    };
14 LPF::LPF(double f_c, double T_s){ //COSTRUTTORE implementazione
15     double tau_c= 1.0/(2.0*PI*f_c); //const. di tempo (data frequenza di taglio
16     _alfa = (T_s ) / (tau_c + T_s );
17     _y = 0.0;
18     _y_old=_y;
19 }
20 void LPF::set_initial_condition(double y_0){
21     _y=y_0;
22     _y_old=_y;
23 }
24 double LPF::update(double x_raw) {
25     _y = ((1 - _alfa) * _y_old) + ( _alfa * x_raw);
26     _y_old = _y;
27     return _y;
28 }
29#endif // _FILTER_H_
```

Il filtro è stato utilizzato sia sui valori misurati permettendo la cancellazione del rumore, sia sui valori desiderati impedendo così sbalzi impulsivi che avrebbero potuto danneggiare gli attuatori e garantendo movimenti più dolci, grazie ad

un'elevata costante di tempo necessaria per il raggiungimento ritardato del valore atteso.

```

1 az_miss=LPF_az_mis.update(map(compass.getAzimuth(), -180, 180, 0, 360));
2 el_miss=LPF_el_mis.update((float)JY901.stcAngle.Angle[0]/32768*180);
3 az_des =LPF_az_des.update (((az_sun + az_target) / 2.00));
4 el_des =LPF_el_des.update(((el_sun + el_target) / 2.00));

```

Il funzionamento del filtro è descritto attraverso l'acquisizione dei dati grezzi e filtrati relativi all'azimut mostrati nel grafico in Figura 6.1. Si noti come il segnale filtrato raggiunga il valore nominale in un tempo T, relativo alla costante di tempo, pari a $\sim 1.5s$.

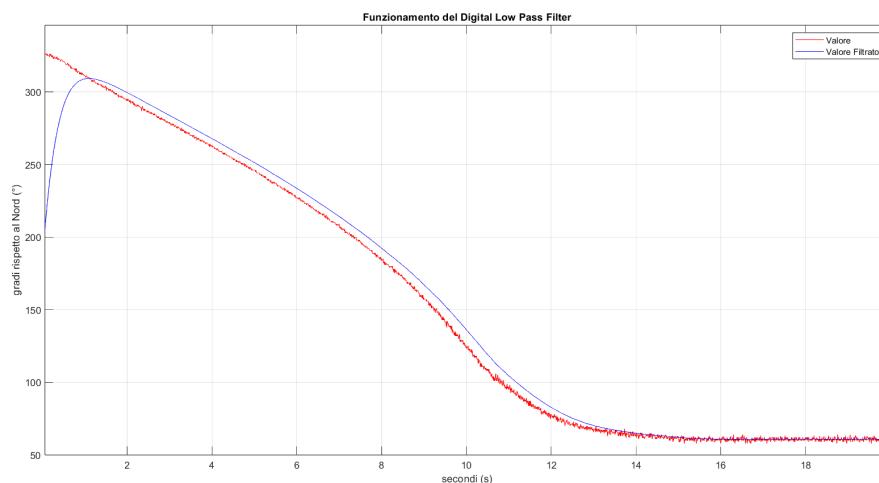


Figura 6.1: Enter Caption

6.2 PI

Il sistema di controllo utilizzato è il controllo PI. Il controllo PID in generale è un sistema di controllo che genera un segnale somma di funzioni proporzionale, integrata e derivata rispetto all'errore tra valore misurato e valore atteso. La parte proporzionale permette al sistema di raggiungere un errore prossimo allo zero, non riuscendo tuttavia ad annullarlo; la parte integrale permette di cancellare totalmente l'errore considerando non solo la differenza istantanea tra valore atteso e misurato, ma anche lo storico della funzione; la parte derivativa è utile per riferimenti dinamici, essendo in grado di prevedere i valori attesi futuri. Nel caso del sistema preso in considerazione, è stato sufficiente usare proporzionale ed integrato, in quanto i valori attesi sono sì variabili ma assimilabili a costanti. Anche il PI come il LPF è stato programmato come una classe con funzioni proprie.

```

1 #ifndef _PID_H_
2 #define _PID_H_
3 #include <math.h>
4 class PID{

```

```

5     private:
6         double _e;
7         double _u;
8         double _e_i;
9         double _delta;
10        double _Kp;
11        double _Ki;
12        double _Kd;
13    public:
14        PID(double soglia, double Kp, double Ki, double Kd );//Definizione
15        Costruttore
16        double signal(double X_des, double X_mis, double Sec);
17        double error(double a1, double a2);
18    };
19 PID::PID(double soglia, double Kp, double Ki, double Kd){//COSTRUTTORE
20     implementazione
21     _e = 0.00;
22     _u = 0.00;
23     _e_i=0.00;
24     _Kp = Kp;
25     _Ki = Ki;
26     _Kd = Kd;
27     _delta = soglia;
28 }
29 double PID::signal(double X_des, double X_mis, double Sec){//COSTRUTTORE
30     implementazione
31     _e = error(X_des, X_mis);
32     _e_i += _e * Sec;
33     _u = _e * _Kp + _e_i * _Ki;
34     if (_u > 1) {
35         _u = 1;
36         _e_i -= _e * Sec;
37     }
38     if (_u < - 1) {
39         _u = -1;
40         _e_i -= _e * Sec;
41     }
42     return _u;
43 }
44 double PID::error(double a1, double a2) {
45     a1 = a1;
46     a2 = a2;
47     double err = 0.00;
48     if (a1 > a2) {
49         err = a1 - a2;
50     } else if (a1 < a2) {
51         err = (a2 - a1) * -1;
52     }
53     if (abs(err) < _delta) err = 0.000;
54     return err;
55 }
56 #endif // _PID_H_

```

Il PI è utilizzato per generare i segnali di controllo (s_{az} e s_{el}) che serviranno a pilotare i motori presi in considerazione gli angoli desiderati, quelli misurati e il tempo di campionamento misurato in secondi.

```

1 s_az= PID_AZ.signal (az_des, az_mis, T_s_ctrl/1000);
2 s_el= PID_EL.signal (el_des, el_mis, T_s_ctrl/1000);

```

Per valutare il funzionamento del controllo, sono stati raccolti i dati relativi alla posizione desiderata e quella misurata sia per l'Azimuth che per l'Elevazione. I guadagni scelti per le due grandezze sono:

- K_p -El= 0.01
- K_i -El= 0.003

- $K_p\text{-Az} = 0.008$
- $K_i\text{-Az} = 0.001$

Si nota in Figura 6.1 che il controllo PI riesce efficacemente ad annullare l'errore e a portare il sistema all'orientamento desiderato partendo da una condizione iniziale diversa.

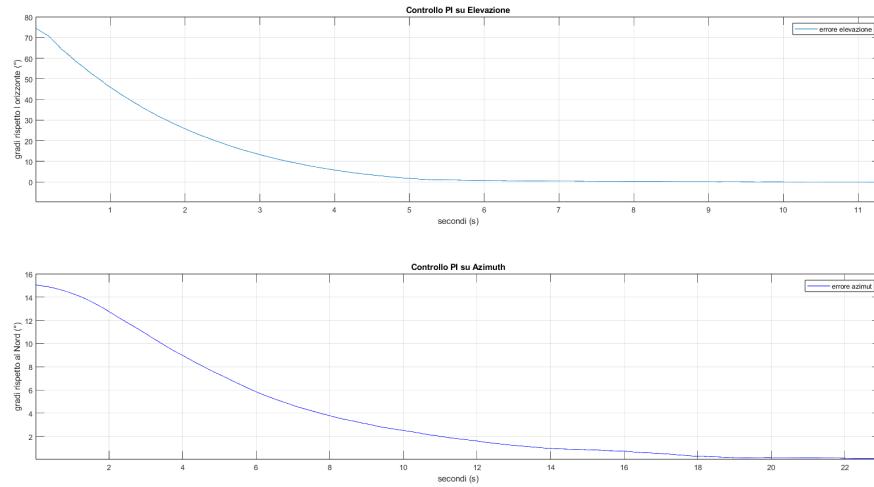


Figura 6.2:

6.3 Segnale di controllo e PWM

Il segnale di controllo generato dal PI compreso nell' intervallo $[-1, +1]$ viene usato come ingresso delle funzioni:

```
1 Controllo_az(s_az);
2 Controllo_el(s_el);
```

Esse generano un segnale PWM che pilota, attraverso i driver, i motori, permettendo loro di ruotare a velocità controllata in entrambe le direzioni.

```
1 void Controllo_az(double S){
2     if (S>=0){
3         analogWrite(DX_az, S*255);
4         analogWrite(SX_az, 0);
5     }else if (S<0){
6         analogWrite(SX_az, -S*255);
7         analogWrite(DX_az, 0);
8     }
9 }
10 void Controllo_el(double S){
11     if (S>=0){
12         analogWrite(DX_el, S*255);
13         analogWrite(SX_el, 0);
14     }else if (S<0){
15         analogWrite(SX_el, -S*255);
16         analogWrite(DX_el, 0);
17     }
18 }
```

Capitolo 7

Alimentazione

7.1 Batteria e ricarica

Il sistema totale consuma meno di 3 A di corrente al massimo carico. Pertanto, è sufficiente l'utilizzo di una batteria da 12V e 3A. È stata scelta una batteria LiFePO (Lithium Iron Phosphate) per evitare possibili complicazioni associate ad altri tipi di batterie, come le Li-ion, che richiedono una gestione più complessa per la sicurezza. La batteria LiFePO offre una maggiore stabilità termica e chimica, lunga durata e un ciclo di vita superiore.

Dato che il progetto è sempre esposto ai raggi del sole, è stato possibile installare un pannello solare e un regolatore di carica. Questo setup permette al sistema di funzionare costantemente e di mantenere la batteria carica, assicurando così un'operatività continua anche in condizioni di bassa illuminazione. Il collegamento del pannello solare, la batteria e il carico è stato effettuato attraverso un caricabatterie controllore per pannello solare (Figura 7.1).



Figura 7.1: caricabatterie controllore per pannello solare

7.2 Modalità Low-Power

Per ridurre il consumo, è stata implementata una modalità di risparmio energetico tramite una libreria specifica per Arduino. Questa libreria consente al microcontrollore di entrare in modalità sleep, riducendo significativamente il consumo di energia quando il sistema non necessita di eseguire operazioni intensive. Per determinare il duty cycle ottimale, è stata analizzata la velocità angolare del sole rispetto a un osservatore sulla Terra, essa è di circa 15 gradi all'ora sia per l'azimuth sia per l'elevazione. Per garantire un inseguimento efficace, il sistema è configurato per attivarsi periodicamente ogni due minuti al fine di sistemare la propria posizione per poi tornare in fase di sleep.

```

1 #ifndef _LOW_P_
2 #define _LOW_P_
3 #include <math.h>
4 #include <LowPower.h>
5 class LOW_P{
6     private:
7         int _T; // secondi
8     public:
9         LOW_P(int timer); //Definizione Costruttore
10        void ACTIVE ();
11    };
12 LOW_P::LOW_P(int timer){//COSTRUTTORE implementazione
13     _T=timer;
14 }
15 }
16 void LOW_P::ACTIVE (){
17     for (int t=0; t<_T; t++){
18         LowPower.powerDown(SLEEP_1S, ADC_OFF, BOD_OFF);
19     }
20 }
21 }
22 #endif // _LOW_P_

```

7.3 Modalità Notturna

Il sistema è progettato per entrare in modalità low-power durante la notte. Essendo l'elevazione il parametro che indica i gradi che forma il sole con l'orizzonte, quando questa lettura è negativa Archimede riconosce che il sole è calato ed entra in una modalità notturna monitorando in maniera periodica la posizione del sole.

```

1 while (el_sun<0){
2     LOW_P.ACTIVE();
3     delay(1000);
4     calcHorizontalCoordinates(utc, latitude, longitude, az_sun, el_sun);
5 }

```

Capitolo 8

Appendice

8.1 Schema elettrico

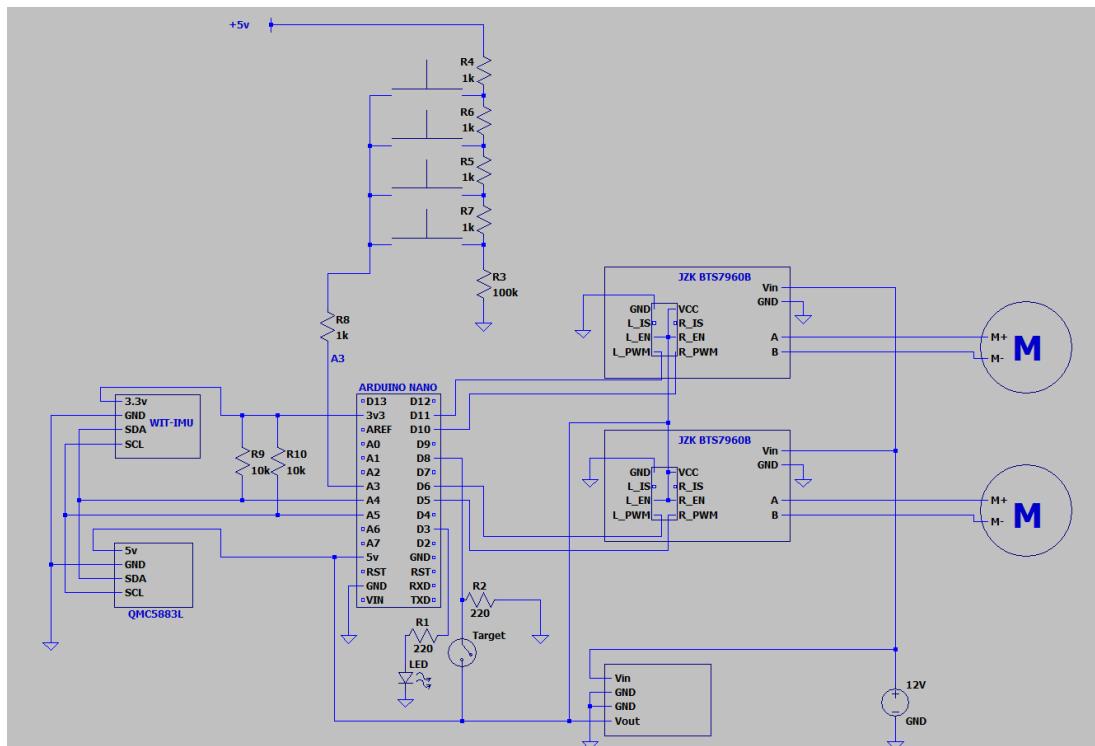


Figura 8.1: Schematico delle connessioni elettriche

8.2 Codice sorgente

```
1 #include <math.h>
2 #include <SolarCalculator.h>
3 #include <TimeLib.h>
4 #include <Wire.h>
5 #include <QMC5883LCompass.h>
```

```

6 #include <EEPROM.h>
7 #include <LowPower.h>
8 #include <JY901.h>
9 #include "DLPF.h"
10 #include "PID.h"
11 #include "LOW_P.h"
12
13
14 #define PIN_LED 3
15 #define PIN_TARGET 8
16 #define GRAD_TO_RAD PI/180.0
17 #define RAD_TO_GRAD 180.0/PI
18 #define F_C_MEAS 50.0 //Hz
19 #define F_C.Des 2.0 //Hz
20 #define T_S_CTRL 1.0/250.0 //s-->1/Hz
21 #define T_S_SUN 10.0 //s
22 #define T_S_TARGET 1.0/250.0
23 #define T_S_LOW_P 45.00 //s
24 #define SOGLIA 0.001
25 #define KP 0.01
26 #define KI 0.003
27 #define KD 0.00
28 #define LOW_P_TIME 60*2 //s
29
30 /*Oggetti*/
31 LPF LPF_az_mis(F_C_MEAS/25, T_S_CTRL);
32 LPF LPF_el_mis(F_C_MEAS, T_S_CTRL);
33 LPF LPF_az_des(F_C_Des, T_S_CTRL);
34 LPF LPF_el_des(F_C_Des, T_S_CTRL);
35
36 PID PID_AZ (SOGLIA, KP, KI, KD);
37 PID PID_EL (SOGLIA, KP, KI, KD);
38
39 LOW_P LOW_P(LOW_P_TIME);
40
41 QMC5883LCompass compass;
42
43 /*PARAMETRI*/
44 const int DX_el=5, SX_el=6, DX_az= 10, SX_az=11;
45 const double latitude = 40.828660 /* */, longitude = 14.189816 /* */;
46 const int address_az = 0, address_el = 100;
47
48 const double T_s_tsrgt = T_S_TARGET;
49 const double T_s_sun = T_S_SUN * 1000; //mS
50 const double T_s_ctrl = T_S_CTRL* 1000; //mS
51 const double T_s_low_p = T_S_LOW_P*1000; //ms
52 //
53
54 /*VARIABILI GLOBALE*/
55 double t_sun_old = 0.00, t_ctrl_old = 0.00, t_target_old = 0.00, t_lowp_old =
      0.00;
56 double az_sun, el_sun, az_target, el_target, az_des, el_des, az_mis, el_mis,
      s_az, s_el;
57 bool statos= HIGH;
58
59
60
61 void setup() {
62   delay (1000);
63   EEPROM.get(address_az, az_target);
64   EEPROM.get(address_el, el_target);
65   compass.init();
66   delay(500);
67   JY901.StartIIC();
68   Serial.begin(115200);
69   setTime(toUtc(compileTime()));
70   time_t utc = now();
71   delay(500);

```

```

72     pinMode(PIN_TARGET, INPUT_PULLUP);
73     pinMode(4, OUTPUT);
74     pinMode(PIN_LED, OUTPUT);
75
76     calcHorizontalCoordinates(utc, latitude, longitude, az_sun, el_sun);
77
78     LPF_az_des.set_initial_condition((az_sun + az_target) / 2.00);
79     LPF_el_des.set_initial_condition((el_sun + el_target) / 2.00);
80     LPF_az_mis.set_initial_condition(compass.getAzimuth() + 180);
81     LPF_el_mis.set_initial_condition((float)JY901.stcAngle.Angle[0]/32768*180);
82 }
83
84
85 void loop() {
86
87     if (millis() - t_sun_old >= T_s_sun) {
88         time_t utc = now();
89         calcHorizontalCoordinates(utc, latitude, longitude, az_sun, el_sun);
90         while (el_sun<0){
91             LOW_P.ACTIVE();
92             delay(1000);
93             calcHorizontalCoordinates(utc, latitude, longitude, az_sun, el_sun);
94         }
95         t_sun_old = millis();
96     }
97
98     if (millis() - t_ctrl_old >= T_s_ctrl ) {
99         compass.read();
100        az_mis= LPF_az_mis.update(map(compass.getAzimuth(), -180, 180, 0, 360));
101        JY901.GetAngle();
102        el_mis= LPF_el_mis.update((float)JY901.stcAngle.Angle[0]/32768*180);
103        az_des =LPF_az_des.update (((az_sun + az_target) / 2.00));
104        el_des =LPF_el_des.update(((el_sun + el_target) / 2.00)); //grad
105        s_az= PID_AZ.signal (az_des, az_mis, T_s_ctrl/1000);
106        s_el= PID_EL.signal (el_des, el_mis, T_s_ctrl/1000);
107        Controllo_az(s_az);
108        Controllo_el(s_el);
109        //stato = !stato;
110        //digitalWrite(4, stato);
111        t_ctrl_old = millis();
112    }
113
114    if (millis() -t_lowp_old >= T_s_low_p and (abs(az_des-az_mis)<=0.5) and (abs(
115        el_des-el_mis)<=1)){
116
117        analogWrite(DX_az, 0);
118        analogWrite(SX_az, 0);
119        analogWrite(DX_el, 0);
120        analogWrite(SX_el, 0);
121        LOW_P.ACTIVE();
122        delay(1000);
123        analogWrite(DX_az, 0);
124        analogWrite(SX_az, 0);
125        analogWrite(DX_el, 0);
126        analogWrite(SX_el, 0);
127        compass.read();
128        az_mis= LPF_az_mis.update(map(compass.getAzimuth(), -180, 180, 0, 360));
129        if (az_mis==360) az_mis=0;
130        JY901.GetAngle();
131        el_mis= LPF_el_mis.update((float)JY901.stcAngle.Angle[0]/32768*180);
132        t_lowp_old=millis();
133    }
134
135    if (millis() - t_target_old >= T_s_tsrget){
136        controllo_taget();
137        digitalWrite(PIN_LED,digitalRead(PIN_TARGET));
138        t_lowp_old=millis();

```

```
139     t_target_old = millis();
140 }
141 PRINT();
142
143 }
144
145
146
147
148
149
150 void controllo_taget(){
151     if(digitalRead(PIN_TARGET)){
152
153         switch (lettura_pulsante()){
154             case 0:
155                 analogWrite(DX_az, 80);
156                 analogWrite(SX_az, 0);
157                 analogWrite(SX_el, 0);
158                 analogWrite(DX_el, 0);
159                 break;
160             case 1:
161                 analogWrite(DX_az, 0);
162                 analogWrite(SX_az, 80);
163                 analogWrite(SX_el, 0);
164                 analogWrite(DX_el, 0);
165                 break;
166             case 2:
167                 analogWrite(DX_az, 0);
168                 analogWrite(SX_az, 0);
169                 analogWrite(DX_el, 50);
170                 analogWrite(SX_el, 0);
171                 break;
172             case 3:
173                 analogWrite(DX_az, 0);
174                 analogWrite(SX_az, 0);
175                 analogWrite(DX_el, 0);
176                 analogWrite(SX_el, 50);
177                 break;
178             case 4:
179                 analogWrite(DX_az, 0);
180                 analogWrite(SX_az, 0);
181                 analogWrite(SX_el, 0);
182                 analogWrite(DX_el, 0);
183                 break;
184         }
185
186         compass.read();
187         az_mis= LPF_az_mis.update(map(compass.getAzimuth(), -180, 180, 0, 360));
188         az_des=az_mis;
189         az_target= (2*az_des)-az_sun;
190         JY901.GetAngle();
191         el_mis=LPF_el_mis.update((float)JY901.stcAngle.Angle[0]/32768*180);
192         el_des=el_mis;
193         el_target= (2*el_des)-el_sun;
194
195         EEPROM.put(address_az, az_target);
196         EEPROM.put(address_el, el_target);
197
198     }
199
200 }
201
202
203 void Controllo_az(double S){
204     if (S>0){
205         analogWrite(DX_az, S*255);
206         analogWrite(SX_az, 0);
```

```

207     }else if (S<0){
208         analogWrite(SX_az, -S*255);
209         analogWrite(DX_az, 0);
210     }else if (S==0){
211         analogWrite(SX_az, 0);
212         analogWrite(DX_az, 0);
213     }
214 }
215 }
216 void Controllo_el(double S){
217     if (S>0){
218         analogWrite(DX_el, S*255);
219         analogWrite(SX_el, 0);
220     }else if (S<0){
221         analogWrite(SX_el, -S*255);
222         analogWrite(DX_el, 0);
223     }else if (S==0){
224         analogWrite(SX_el, 0);
225         analogWrite(DX_el, 0);
226     }
227 }
228 }
229
230
231 /*-----CALCOLO DATA E ORA-----*/
232 time_t compileTime() {
233     const uint8_t COMPILE_TIME_DELAY = 8;
234     const char *compDate = __DATE__, *compTime = __TIME__, *months = "
235     JanFebMarAprMayJunJulAugSepOctNovDec";
236     char chMon[4], *m;
237     tmElements_t tm;
238     strncpy(chMon, compDate, 3);
239     chMon[3] = '\0';
240     m = strstr(months, chMon);
241     tm.Month = ((m - months) / 3 + 1);
242     tm.Day = atoi(compDate + 4);
243     tm.Year = atoi(compDate + 7) - 1970;
244     tm.Hour = atoi(compTime);
245     tm.Minute = atoi(compTime + 3);
246     tm.Second = atoi(compTime + 6);
247     time_t t = makeTime(tm);
248     return t + COMPILE_TIME_DELAY;
249 }
250 /*-----CONVERSIONE IN UTC-----*/
251 time_t toUtc(time_t local) {
252     int utc_offset = 2;
253     return local - utc_offset * 3600L;
254 }
255 /*-----*/
256
257
258
259 int lettura_pulsante()
260 {
261     const int Az_up = 819, Az_dw = 612, El_up = 408, El_dw= 203, delta = 10;
262     int Value = analogRead(A3);
263     if (Value <Az_up+delta && Value >Az_up-delta) return 0;
264     if (Value <Az_dw+delta && Value >Az_dw-delta) return 1;
265     if (Value <El_up+delta && Value >El_up-delta) return 2;
266     if (Value <El_dw+delta && Value >El_dw-delta) return 3;
267     if (Value <10+delta && Value >10-delta) return 4;
268 }
269
270 void PRINT() {
271     Serial.print(millis());
272     Serial.print(", ");
273     Serial.print(az_sun);

```

```

274     Serial.print(",");
275     Serial.print(el_sun);
276     Serial.print(",");
277     Serial.print(az_des);
278     Serial.print(",");
279     Serial.print(el_des);
280     Serial.print(",");
281     Serial.print(az_mis);
282     Serial.print(",");
283     Serial.print(el_mis);
284     Serial.print(",");
285     Serial.print(az_des-az_mis);
286     Serial.print(",");
287     Serial.println(el_des-el_mis);
288
289
290 }
```

8.3 Riferimenti immagini

- **Figura 5.2:** Elementi di fisica di John D. Cutnell, Kenneth W. Johnson.

8.4 Riferimenti delle Librerie

Durante la realizzazione di questo progetto sono state utilizzate le seguenti librerie:

- **math.h:** Libreria standard del C per operazioni matematiche. Maggiori informazioni disponibili su <https://en.cppreference.com/w/c/numeric/math>.
- **SolarCalculator.h:** Utilizzata per il calcolo della posizione solare. Disponibile su GitHub: <https://github.com/jarzebski/Arduino-SolarCalculator>.
- **TimeLib.h:** Utilizzata per la gestione del tempo. Disponibile su GitHub: <https://github.com/PaulStoffregen/Time>.
- **Wire.h:** Libreria standard per la comunicazione I2C. Maggiori informazioni disponibili su <https://www.arduino.cc/en/Reference/Wire>.
- **QMC5883LCompass.h:** Utilizzata per l’interfaccia con la bussola QMC5883L. Disponibile su GitHub: <https://github.com/mprograms/QMC5883LCompass>.
- **EEPROM.h:** Libreria standard per la gestione della memoria EEPROM. Maggiori informazioni disponibili su <https://www.arduino.cc/en/Reference/EEPROM>.
- **LowPower.h:** Utilizzata per la gestione delle modalità di risparmio energetico. Disponibile su GitHub: <https://github.com/rocketscream/Low-Power>.
- **JY901.h:** Utilizzata per l’interfaccia con il modulo IMU JY901. Disponibile su GitHub: <https://github.com/ssalonen/JY901-arduino>.

8.5 Schema a blocchi

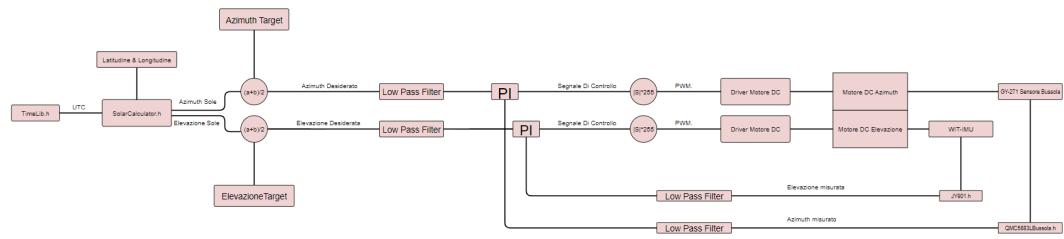


Figura 8.2:

Ringraziamenti

Ringrazio: Mamma, Papà, Marcello, Maria, e Bianca. In più i miei compagni di corso e di esami: Andrea e Davide.