Indice

Riassunto analitico

Il lavoro svolto si propone di realizzare, abbattendone drasticamente il costo di produzione, un sistema microrobotico di concentrazione solare, ovvero un sistema che sia in grado di riflettere i raggi solari su di un bersaglio mantenendo fisso il puntamento durante tutto l'arco della giornata. Questa funzione viene svolta grazie allo sviluppo, durante il lavoro di tesi, di un algoritmo di calcolo e di un sensore basato su fotodiodi, denominato "meridiana elettronica", che permettono di calcolare la posizione del sole relativamente allo specchio riflettente e, a partire dalla conoscenza delle coordinate altazimutali del sole ricavate con un algoritmo di calcolo numerico, anche la posizione del bersaglio e dello specchio rispetto allo stesso sistema di riferimento altazimutale. L'orientazione dello specchio avviene in modalità biassiale tramite due motori elettrici in corrente continua pilotati da un microcontrollore su cui è stato implementato un algoritmo di controllo di tipo proporzionale. Il lavoro di ricerca si inserisce in un'attività di laboratorio, contribuendo allo sviluppo di progetti di ricerca già iniziati nel laboratorio stesso.

Prefazione

Questo studio nasce con l'intento di portare un elemento di innovazione nell'attuale panorama delle energie rinnovabili, in particolare quelle relative alla produzione di energia fotovoltaica tramite concentrazione dei raggi solari. Attualmente infatti, la tecnologia fotovoltaica e solare a concentrazione sta assumendo un ruolo sempre più rilevante rispetto al fotovoltaico o ai collettori solari tradizionali, grazie ai notevoli vantaggi che offre in termini di efficienza di conversione e di sfruttamento dell'energia solare disponibile. Uno dei principali elementi che limitano la diffusione di questa innovativa tecnologia è senz'altro l'elevato costo di produzione dell'impianto, che pertanto risulta economicamente appetibile soltanto se di grossa taglia, ovvero per elevate produzioni di energia elettrica (da 1 MW di picco in su [2]). A questo riguardo risulta di grande importanza la riduzione del costo del sistema di puntamento solare, parte fondamentale di un sistema a concentrazione, la cui incidenza sul costo finale dell'impianto è determinante ($\cong 25\%$ [2]). L'obiettivo di questo studio è quindi di realizzare un sistema di puntamento e di focalizzazione dei raggi solari minimizzandone il costo.

Sommario

Nel capitolo 1 viene presentato lo stato dell'arte delle tecnologie a concentrazione, fotovoltaica e solare, con un focus sulle componenti di costo più importanti di un sistema a concentrazione;

Nel capitolo 2, dopo aver definito cosa si intende per coordinate altazimutali, vengono confrontati alcuni algoritmi di calcolo numerico utilizzati per la determinazione della posizione del sole;

Nel capitolo 3 viene illustrato il sensore fotoelettrico messo a punto nel lavoro di tesi e denominato meridiana elettronica, attraverso il quale è possibile ricavare le coordinate altazimutali del sole;

Nel capitolo 4 si passa all'analisi del sistema di concentrazione solare, scopo di questa tesi, evidenziandone le caratteristiche hardware, software e infine il costo complessivo;

Nel capitolo 5 si descrivono le prove sperimentali eseguite per valutare l'efficacia del sistema a concetrazione solare.

Introduzione

In un mondo in cui l'approvvigionamento energetico risulta uno dei fattori centrali per lo sviluppo delle attività umane e per gli assetti geopolitici del pianeta, la possibilità di sganciarsi dalla dipendenza da fonti fossili, in via di esaurimento e proprietà di pochi Paesi nel mondo, assume un'importanza vitale. Lo sviluppo e la ricerca nel campo delle energie rinnovabili risulta pertanto inevitabile e lungimirante, anche per garantire un sviluppo ecosostenibile, riducendo la produzione di anidride carbonica causata dall'impiego di fonti fossili.

Negli ultimi anni le tecnologie solare e fotovoltaica sono state e sono tutt'oggi al centro delle maggiori ricerche nel campo delle energie rinnovabili, perché dimostrano di avere ampi margini di miglioramento per quanto riguarda l'efficienza di conversione dell'energia solare in energia termica od elettrica. Una delle più significative innovazioni degli ultimi anni nel campo del fotovoltaico è la tecnologia fotovoltaica a concentrazione (CPV), che dimostra come ci sia ancora spazio per sviluppare la tecnologia fotovoltaica classica, incrementandone l'efficacia.

Il presente lavoro di tesi si inserisce proprio all'interno del filone di ricerca delle tecnologie a concentrazione solare tentando di portare un elemento di innovazione sul sistema di puntamento e focalizzazione, riducendone drasticamente il costo di produzione al fine di poter rendere fruibile la tecnologia a concentrazione anche in contesti di produzione di piccole potenze (sotto 1 MW di picco).

Capitolo 1

Tecnologie a concentrazione solare

L'attuale panorama delle tecnologie solari e fotovoltaiche è caratterizzato da due principali categorie, che si distinguono per l'impiego, o meno, di un sistema per la focalizzazione dei raggi solari. Esistono, da un lato, i tradizionali pannelli fotovoltaici per la produzione di energia elettrica e i collettori solari per il riscaldamento dell'acqua, il funzionamento dei quali prevede lo sfruttamento della luce diretta sul pannello stesso, mentre dall'altro lato sta assumendo importanza crescente la tecnologia fotovoltaica e solare a concentrazione, per la quale il sistema di focalizzazione dei raggi solari riveste un ruolo fondamentale. Il motivo per cui attualmente l'attenzione della ricerca è rivolta verso i sistemi a concentrazione è perchè essi permettono di incrementare in modo significativo l'efficienza di conversione dell'energia solare in energia elettrica, attraverso una migliore esposizione del cella durante tutto l'arco di una giornata. Ciò nonostante questa tecnologia presenta alcune criticità, in particolare gli elevati costi fissi, che ad oggi ne hanno limitato la diffusione alle sole grosse produzioni industriali (da 1 MW di potenza in su).

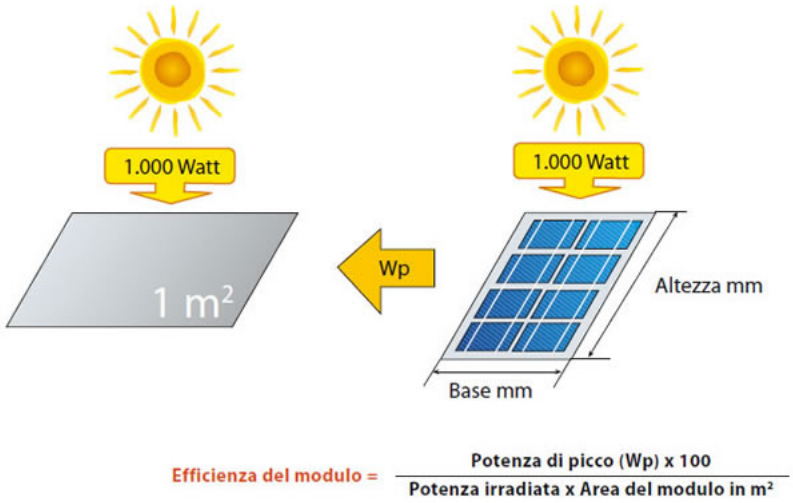

Figura 1.1: Definizione di efficienza di conversione del modulo fotovoltaico [15]

1.1 Fotovoltaico a concentrazione

La tecnologia fotovoltaica a concentrazione (CPV, "Concentrating Photo Voltaic") si differenzia dalla tecnologia fotovoltaica "classica" non tanto per la modalità con cui viene convertita l'energia solare in energia elettrica (effetto fotovoltaico della giunzione PN), quanto per l'utilizzo di:

- celle multigiunzione particolarmente sofisticate (più efficienti, ma anche molto più costose di quelle tradizionali al silicio mono/policristallino);
- un apparato di focalizzazione dei raggi solari;
- un sistema di inseguimento solare.

L'elemento fondamentale che giustifica l'utilizzo di questa costosa tecnologia è l'incremento dell'efficienza di conversione, che passa dal $15 \div 22$ % delle celle in silicio al $38 \div 40$ % delle celle multigiunzione ($9 \div 13$% per le celle in tecnologia film sottile), dal $15 \div 19$% al $26 \div 31$% per quanto riguarda l'efficienza dei moduli, mentre prendendo in considerazione il sistema completo

(comprendente quindi anche batterie di accumulo, inverter, etc...) l'efficenza si attesta sul $22 \div 27$ % , rispetto al $14 \div 17\%$ delle sistema fotovoltaico basato su silicio, al $7 \div 8\%$ del sistema in tecnologia film sottile e al $13 \div 19\%$ del solare termico.

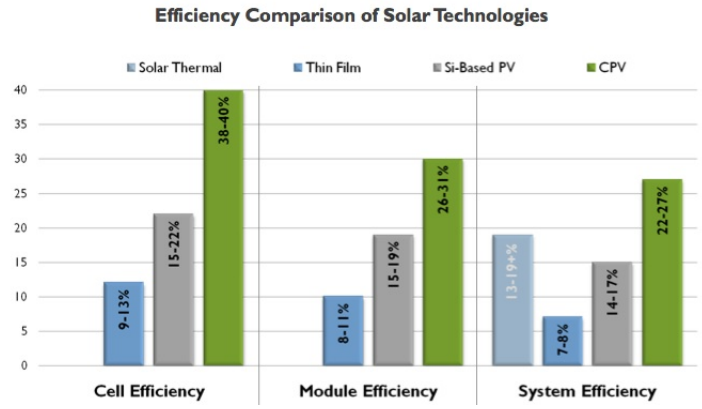

Figura 1.2: Comparativa efficienza di conversione fra fotovoltaico a concentrazione, fotovoltaico basato su celle al silicio, celle a film sottile e solare termico [2]

Ci sono poi altri vantaggi, di tipo ambientale, che ne favoriscono l'impiego a scapito della tecnologia fotovoltaica tradizionale, ad esempio negli impianti a terra evitano l'ombreggiamento permamente di una porzione di terreno.

Il corrispettivo da pagare per tale miglioramento di prestazioni è un consistente aumento del costo di produzione delle celle fotovoltaiche, rendendone conveniente l'impiego soltanto associandovi un sistema di concentrazione solare. Diventa pertanto di vitale importanza lo sfruttamento di superfici di celle solari quanto più piccole possibili a fronte di una potenza installata equivalente a quella ottenibile con le celle solari tradizionali e questo è reso possibile grazie appunto ad un sistema di focalizzazione, che concentra su di un'area molto piccola la luce incidente su di una superficie inizialmente molto più estesa (fig. 1.3). Questa operazione può essere svolta tramite l'impiego

di elementi rifrattivi come, ad esempio, le lenti di Fresnel, oppure mediante specchi riflettenti.

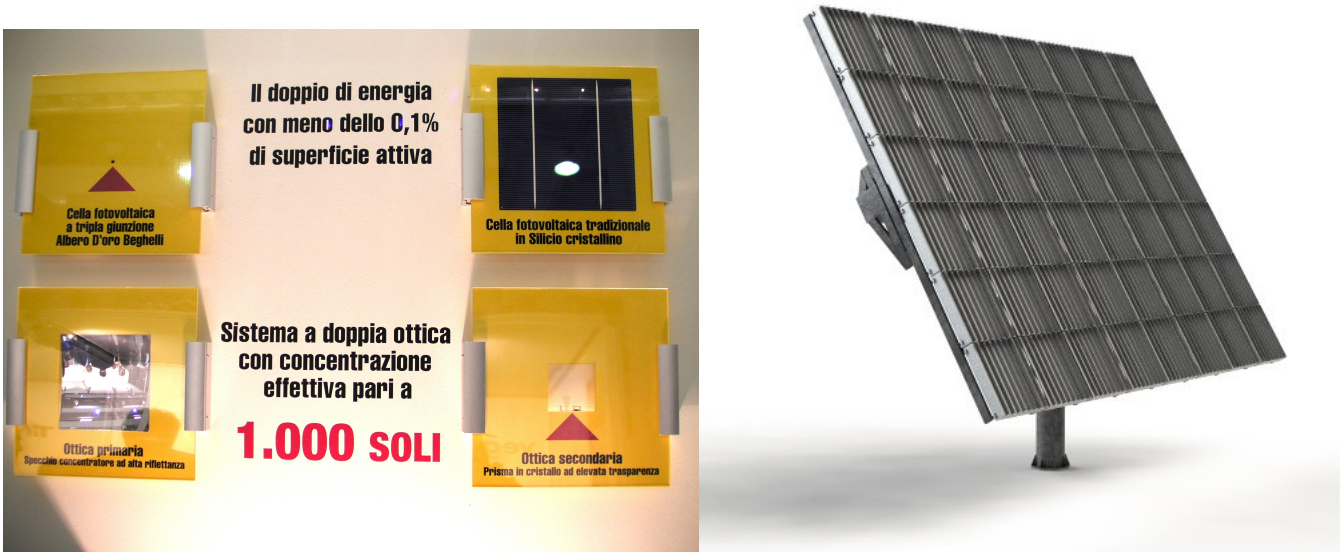

Figura 1.3: Superfice attiva e ottiche del sistema fotovoltaico a concentrazione "Albero d'Oro" della Beghelli [2]

Ultimo, ma altrettanto fondamentale, è il sistema di inseguimento solare, che tramite la movimentazione del pannello in modalità monoassiale (per i sistemi meno efficienti) o biassiale (nell'evoluzione più recente) consente di massimizzare la ricezione di energia solare durante tutto l'arco della giornata.

1.1.1 Inseguitori ad un grado di libertà

Gli inseguitori ad un solo grado di libertà sono caratterizzati da un asse di rotazione passante per lo zenith-nadir e pertanto muovono il pannello da EST a OVEST. Per massimizzare l'efficienza del sistema il pannello deve essere inclinato manualmente di un angolo, detto di *tilt*, rispetto al terreno, in base

alla stagione dell'anno. L'incremento di produzione elettrica rispetto ad un sistema fisso è approssimativamente pari al 25%.

1.1.2 Inseguitori a due gradi di libertà

Gli inseguitori più sofisticati dispongono di due gradi di libertà, con cui si prefiggono di allineare perfettamente e in tempo reale l'ortogonale dei pannelli fotovoltaici con i raggi solari. L'operazione di inseguimento viene fatta impiegando due motori elettrici e una struttura meccanica più complessa, pertanto risulta più costosa rispetto al caso monoassiale. Con questi inseguitori si registrano aumenti di produzione elettrica che raggiungono anche il $35 \div 40$ %.

1.2 Solare a concentrazione

L'impianto solare a concentrazione, a differenza di quello fotovoltaico, sfrutta la componente termica dell'energia solare per produrre energia elettrica, con un processo che si compone essenzialmente di due fasi:

1. riscaldamento ad alte temperature (centinaia di gradi centigradi) di un fluido termovettore opportuno;

2. conversione dell'energia termica accumulata nel fluido in energia elettrica mediante generatore di vapore e turboalternatore, o impiego diretto del calore ottenuto (ad es. per uso sanitario).

A questo proposito è possibile ad oggi distinguere tre tipologie di impianti a concentrazione:

- Impianti con specchi parabolici lineari;
- Impianti a torre centrale;
- Impianti con specchio a disco parabolico.

1.2.1 Impianti con specchi parabolici lineari

Nell'impianto a specchi parabolici lineari l'elemento riflettente ha una superficie semicilindrica con sezione parabolica (detta anche paraboloide lineare) (fig. 1.4). L'orientazione della struttura avviene lungo un solo asse e inoltre si compone di un ricevitore, posto nel fuoco del paraboloide, il quale non è altro che un condotto nel quale scorre un fluido termovettore sottoposto alla concentrazione solare. Esso viene quindi vaporizzato e inviato a una turbina per la conversione dell'energia termica in elettrica. Il fluido ha pertanto un ruolo fondamentale perché deve soddisfare contemporaneamente due requisiti:

- deve vaporizzare alle temperature di esercizio (qualche centinaio di gradi centigradi) ;
- deve permettere la conservazione prolungata del calore accumulato, in modo da poterne usufruire anche in mancanza di luce (di notte o quando il cielo è coperto).

Mentre il primo requisito può essere ottenuto senza particolari problemi viste le elevate temperature raggiungibili con la concentrazione solare, il secondo risulta più complicato da rispettare e si ottiene soltanto con l'impiego di un fluido composto di sali fusi. Attualmente è questa la tecnologia solare a concentrazione più diffusa.

Figura 1.4: Schema di un impianto a specchi parabolici lineari e foto dell'impianto di Kramer Junction [4]

1.2.2 Impianti con specchio a disco parabolico

In questo caso l'elemento concentratore ha una forma a disco parabolico con il ricevitore sempre posto in corrispondenza del fuoco, ma con la differenza che la movimentazione del sistema è biassiale, ovvero sia verticale che orizzontale, in modo da ottenere un inseguimento solare ottimale.

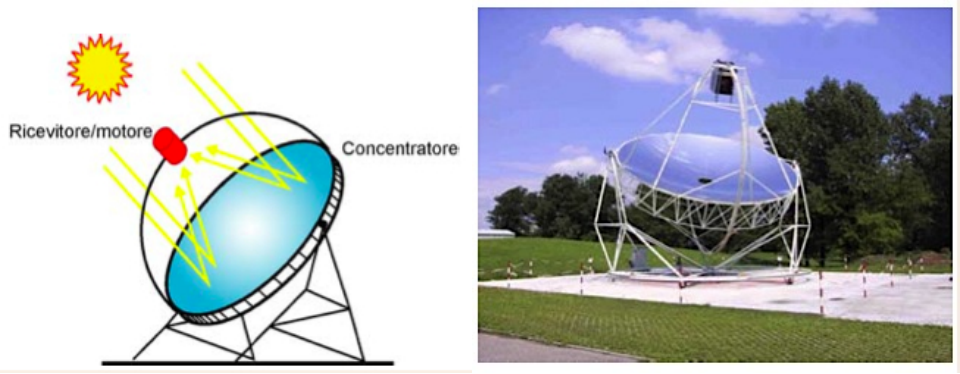

Figura 1.5: Schema impianto a disco parabolico e foto esemplificativa [4]

1.2.3 Impianti a torre centrale

L'impianto a torre centrale non utilizza specchi di forma parabolica, bensì centinaia di specchi piani movimentati ciascuno da 2 motori di precisione (movimentazione biassiale), che ne consentano la corretta orientazione automatizzata durante tutto l'arco di una giornata, in modo da convogliare la luce del sole verso la sommità della torre sulla quale è posizionato il collettore. A differenza delle altre due tecnologie appena presentate, in generale gli elementi riflettenti non si orienteranno in modo da essere affacciati al sole (questo accadrà soltanto se il bersaglio si trova proprio sulla congiungente sole-specchio), bensì ognuno di essi dovrà orientarsi in maniera differente a seconda della propria posizione relativa rispetto al bersaglio. Le temperature raggiungibili sono in questo caso più elevate rispetto agli altri due impianti precedentemente descritti, ma i costi di produzione della struttura sono nettamente più alti a causa soprattutto del sistema di movimentazione che comprende i già sopracitati 2 motori elettrici per specchio, nonché l'elettronica necessaria per il controllo dei motori. Ciò fa sì che, ancor più che negli altri casi, gli ingenti costi fissi di produzione non permettano a questo tipo di impianti un impiego per piccole produzioni domestiche.

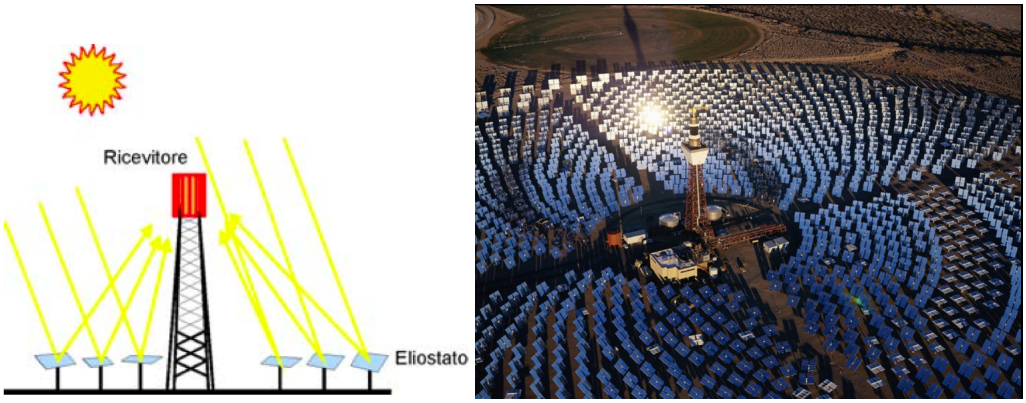

Figura 1.6: Schema impianto a torre centrale e foto dell'impianto Gemasolar in Spagna da 20 MW [4] [14]

1.2.4 Analisi dei costi di un sistema a concentrazione

In figura 1.7 è possibile osservare l'incidenza che i vari fattori di costo di un sistema CPV hanno sull'ammontare totale: la produzione e l'assemblaggio delle celle solari ovviamente concorrono in maniera molto rilevante nella determinazione del costo finale dell'impianto ma, nel caso dei sistemi a concentrazione, hanno importanza determinante anche il sistema di tracking e le lenti di focalizzazione (assieme contano più del 25%). Se pertanto si riuscisse a ridurre il costo associato a queste due ultime componenti si otterrebbe un notevole vantaggio, non solo per i sistemi CPV, ma anche per i sistemi a concentrazione solare come gli impianti a torre centrale appena descritti. Questa riduzione di costo potrebbe altresì rendere conveniente un impiego di tali tecnologie per piccole potenze installate.

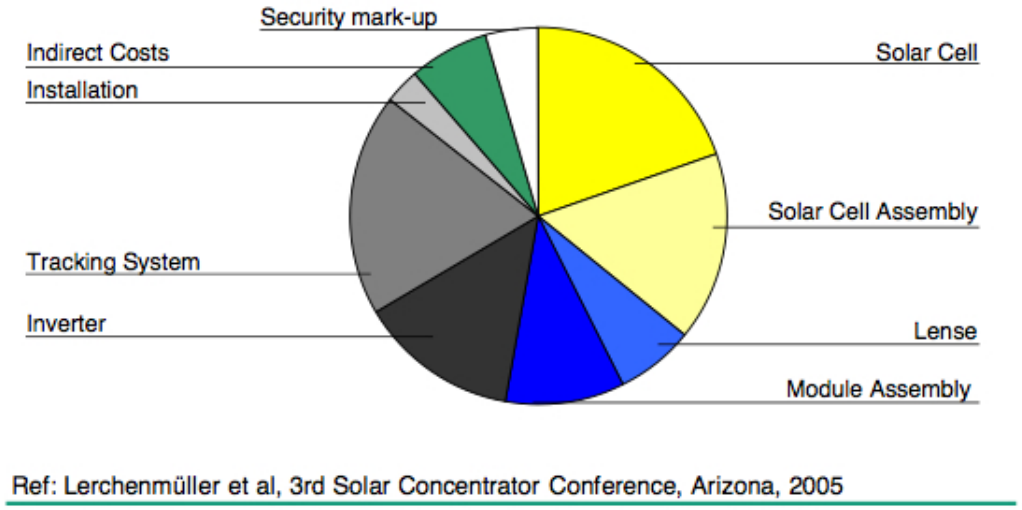

Figura 1.7: Analisi dei costi di produzione di un sistema fotovoltaico a concentrazione (CPV) [2]

Per quanto riguarda le meccaniche di movimentazione degli specchi o dei pannelli stessi, sono i motori elettrici a rappresentare il "collo di bottiglia": essi devono avere caratteristiche di elevata accurattezza di posizionamento

(sotto il grado), di elevata coppia (sia in fase di spostamento, sia in fase di mantenimento della posizione) e di affidabilità. Non è invece importante per queste applicazioni la velocità di rotazione, in quanto il sole si "muove" soltanto di alcune frazioni di grado al minuto. Il motore che più di ogni altro risponde a queste esigenze è quindi quello di tipo stepper, con un passo che sia il più piccolo possibile. Peculiarità di questo tipo associate ad un motore passo-passo ne fanno lievitare il prezzo a centinaia di euro.

Gli elementi rifrattivi o riflessivi, a seconda che si tratti di lenti o specchi, sono anch'essi una fonte di costo non trascurabile. Questo è dovuto principalmente alle particolari geometrie che devono assumere per adempiere alla propria funzione (fattore di concentrazione di $500 \div 1000$) e che ne determinano una certa complessità realizzativa in fase di produzione.

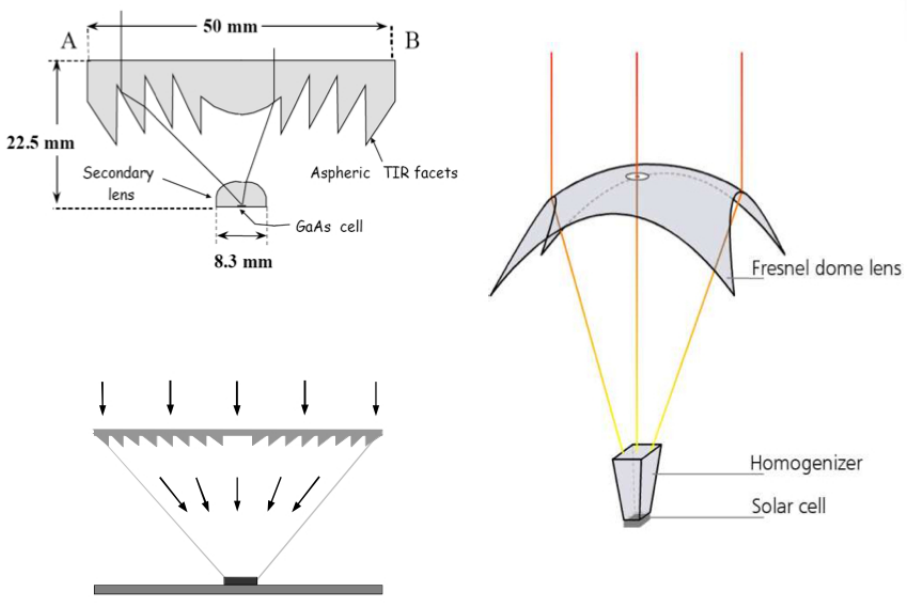

Figura 1.8: Lenti di Fresnel [2]

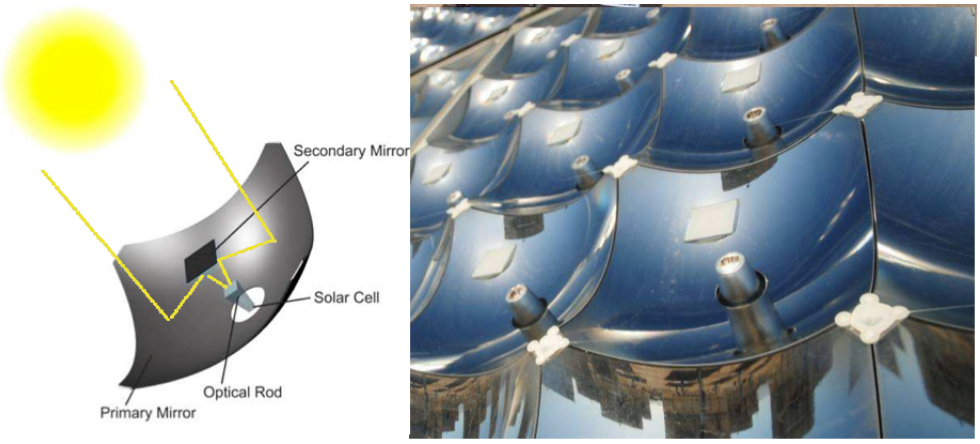

Figura 1.9: Specchietti parabolici [2]

L'idea che sta alla base di questa tesi è quindi quella di impiegare componenti economici per realizzare un sistema di focalizzazione e inseguimento. Essi saranno pilotati da una elettronica "intelligente" in modo da ottenere risultati comparabili a quelli raggiungibili utilizzando tecnologie più costose.

Capitolo 2

Determinazione della posizione solare

Per ottenere un puntamento solare ottimale è necessario conoscere con la massima accuratezza sia la posizione del bersaglio su cui riflettere i raggi solari, sia la posizione del sole nella sfera celeste durante tutto l'arco di una giornata, 365 giorni all'anno. L'importanza di conoscere le coordinate astronomiche del sole, per la nostra applicazione, risiede nel fatto di avere un riferimento attraverso il quale determinare la posizione degli specchi e, attraverso relazioni di tipo trigonometrico, anche quella del bersaglio. É quindi innanzitutto indispensabile definire un sistema di riferimento appropriato, che semplifichi i calcoli e riduca l'errore di posizionamento.

I sistemi di riferimento più usati in astronomia sfruttano le coordinate sferiche e si basano sulla definizione di un asse passante per l'osservatore e di un piano fondamentale, perpendicolare a questa direzione, la cui intersezione con la sfera celeste individua l'orizzonte astronomico.

2.1 Sistema altazimutale

Il sistema di riferimento scelto per la nostra applicazione è il sistema altazimutale, in cui l'asse scelto come direzione fondamentale è la verticale del luogo di osservazione e il piano fondamentale è l'orizzonte astronomico. Esso è caratterizzato dalle seguenti grandezze:

Zenit: è il punto di intersezione fra la verticale nel punto di osservazione e la sfera celeste. L'angolo zenitale è quindi l'angolo tra la verticale dell'osservatore e la semiretta che parte dall'osservatore e passa per il corpo celeste, misurando l'angolo a aprtire dallo Zenit. L'altezza è invece l'arco complementare;

Azimut: chiamato anche angolo azimutale è l'ascissa sferica di un punto sulla sfera celeste. Viene definito come l'angolo, sul piano equatoriale, compreso tra la direttrice NORD-SUD e l'intersezione con l'equatore celeste del meridiano passante per l'astro e lo Zenit, misurato in senso orario a partire dal NORD.

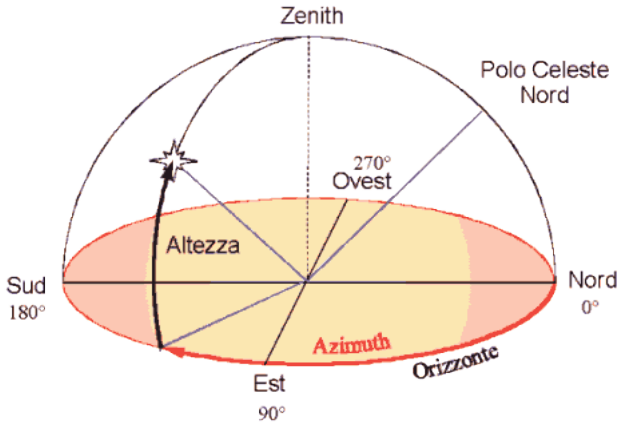

Figura 2.1: Sistema di coordinate altazimutali [16]

Questo tipo di sistema di riferimento ha il vantaggio di permettere di determinare in modo semplice e immediato le coordinate di un punto sulla

sfera celeste ma, essendo dipendente dall'orizzonte astronomico e dallo Zenit, esse risulteranno diverse a seconda della posizione dell'osservatore.

2.2 Coordinate altazimutali del sole

Le coordinate altazimutali del sole non dipendono però soltanto dal luogo di osservazione, ma anche dall'istante di misura. Infatti esse variano nell'arco di una stessa giornata a causa del moto di rotazione della Terra, mentre il moto di rivoluzione terrestre ne provoca la variabilità in base al periodo dell'anno solare in cui ci si trova a fare la misura. Moto di rotazione e di rivoluzione fanno sì che dalla Terra il sole ci appaia come se si muovesse lungo una traiettoria detta eclittica.

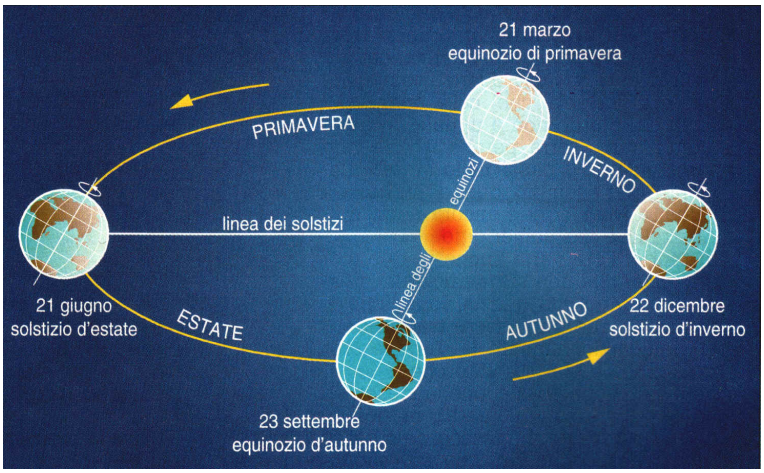

Figura 2.2: Rivoluzione terrestre [18]

2.2.1 Algoritmi numerici per il calcolo della posizione del sole

Per determinare le coordinate altazimutali del sole esistono molteplici algoritmi di tipo numerico, che a partire dalle informazioni su data (anno, mese,

giorno e orario), sulla latitudine e longitudine del luogo, riescono a ricavare le coordinate astronomiche del sole con una certa accuratezza. Maggiore è l'accuratezza con cui si vuole conoscere la posizione del sole e maggiore sarà la complessità computazionale dell'algoritmo. Le tecniche classificabili come "veloci" dal punto di vista computazionale riescono tipicamente a mantenere l'errore massimo dell'ordine di 0.01°: fra le più note possiamo ricordare la formula di Spencer (Spencer, 1971), caratterizzata da un errore maggiore di 0.25°; l'algoritmo di Pitman and Vant-Hull (Pitman and Vant-Hull, 1978) che ha ridotto l'errore a 0.02°; quello di Walraven (Walraven, 1978) che, con successive ottimizzazioni (Walraven, 1979; Archer, 1980; Wilkinson, 1981, 1983; Muir, 1983), ha raggiunto un errore massimo di 0.013°; la tecnica di Michalsky (Michalsky, 1988), 0.011°; infine l'algoritmo PSA (Blanco-Muriel et al., 2001), con un errore che non supera gli 0.008°. Tutte queste tecniche sono valide per intervalli di tempo limitati: 1950–2050 per l'algoritmo Michalsky e 1995–2015 per la formula PSA, soltanto per citarne alcuni.

Se invece è forte l'esigenza di grande accuratezza vi sono algoritmi astronomici complessi che permettono di raggiungere errori massimi di 0.0003°, al prezzo di dover eseguire calcoli molto più pesanti. Fra questi è possibile menzionare la formula di Meeus (1988), che è stata rivista per applicazioni legate alla concentrazione solare da Reda e Andreas (2004) ed è quindi conosciuta come SPA (Solar Position Algorithm). L'errore massimo in questo caso è di 0.0003°, inoltre l'intervallo temporale di validità è molto ampio (2000 A.C. – 6000 D.C.).

Per la nostra tesi abbiamo fatto uso di un algoritmo numerico, formulato nel 2007 da Roberto Grena del Centro Ricerche ENEA [1] che, nonostante la complessità computazionale sia paragonabile agli algoritmi cosiddetti "veloci", riesce a garantire un errore massimo pari a 0.0027°, valore sicuramente appropriato per l'applicazione di nostro interesse e, in generale, per le applicazioni legate alla concentrazione solare, che richiedono un errore massimo inferiore a 1°. Il periodo di validità di questo algoritmo è limitato a 2003-

2022, dopodichè si renderebbero necessarie delle modifiche per mantenere l'errore entro i limiti prefissati.

Capitolo 3

Meridiana elettronica

La meridiana elettronica è il dispositivo centrale del lavoro di tesi. Consiste essenzialmente in un sensore fotoelettrico che, associato ad un semplice algoritmo di calcolo ideato dal prof. Giovanni Pennelli, permette di determinare gli angoli azimutale e zenitale del sole, facendo uso di componenti fra i più economici in commercio e quindi con un costo di realizzazione molto basso. Essa è costituita da 2 unità principali: il blocco sensore che fornisce i dati e il blocco di elaborazione che li riceve, li elabora e li comunica a un PC per successive analisi.

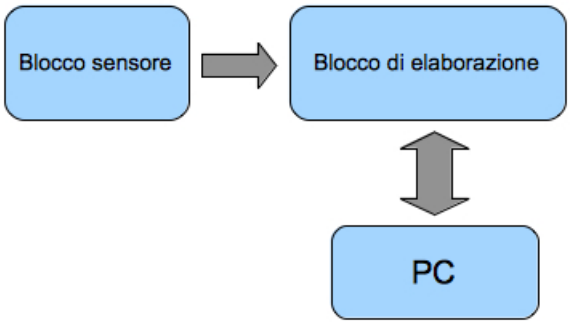

Figura 3.1: Schema a blocchi della meridiana elettronica

3.1 Sensore

Il blocco sensore è rappresentato da una scatolina di plastica nera alla quale deve essere praticato un foro, di forma e dimensioni geometriche ben precise, che serve per permettere alla luce di entrare e incidere su 4 fotodiodi posti in una configurazione a 4 quadranti e inseriti in un semplice circuito analogico. Il foro deve essere di forma quadrata, o più in generale rettangolare a seconda della posizione dei fotodiodi, in modo che le proiezioni dei vertici del foro sul piano dei fotodiodi cadano al centro di ciascun fotodiodo e deve avere spessore quanto più piccolo possibile (molto più piccolo della distanza dei fotodiodi dal foro), compatibilmente con le esigenze di stabilità meccanica della struttura. Quest'ultima condizione è necessaria perchè l'algoritmo di calcolo degli angoli si basa sull'ipotesi che la proiezione del foro sul piano dei fotodiodi mantenga sempre la forma del foro, indipendentemente dall'inclinazione dei raggi solari.

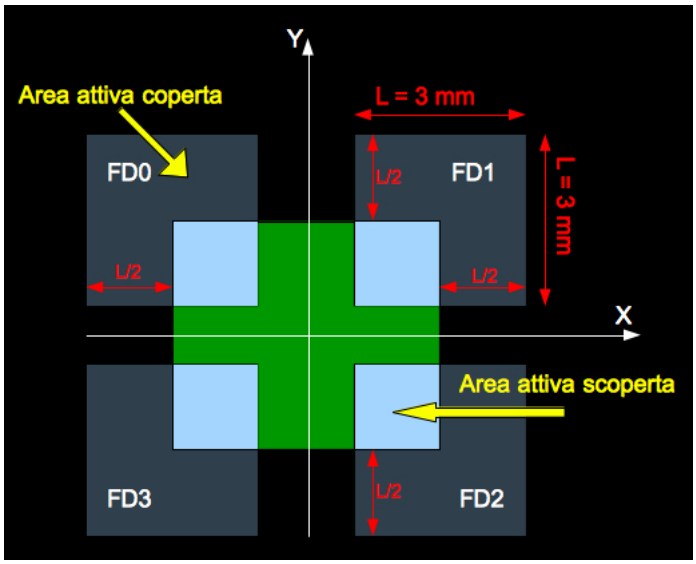

Figura 3.2: Posizionamento e geometrie dei fotodiodi e del foro

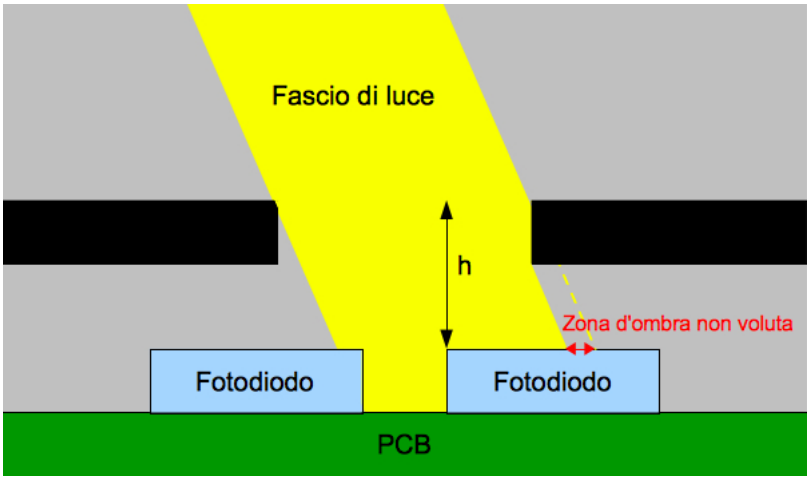

Figura 3.3: Zona d'ombra non voluta nel caso di foro spesso

L'impossibilità di realizzare in laboratorio un foro di forma rettangolare, ci ha costretto a pensare ad un'altra strategia che prevede l'utilizzo di una maschera proiezionale. La maschera è realizzata con un'etichetta adesiva nera tagliata in modo opportuno per ottenere la geometria rettangolare voluta ed è applicata ad una sottile (1mm) lastra di vetro trasparente. Essa viene interposta tra fotodiodi e superficie interna della scatolina (fig. 3.4), ottenendo in questo modo il soddisfacimento di tutti i requisiti geometrici che ci eravamo posti, in particolare la distanza fotodiodi-maschera risulta controllata e di 1mm. L'allineamento dei fotodiodi alla maschera è un'operazione importante e delicata ed è stata svolta manualmente stringendo 2 bulloni inseriti nelle rispettive viti (vedi Fig. 3.5).

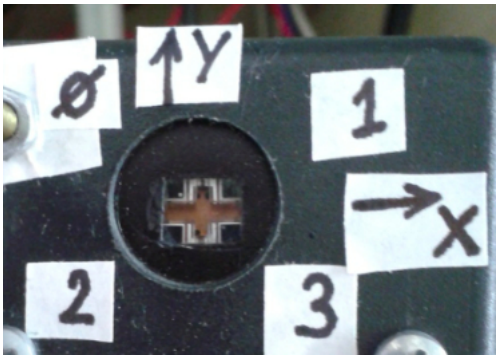

Figura 3.4: Foto configurazione fotodiodi

Il circuito elettrico in cui sono inseriti i fotodiodi amplifica e converte in tensione la corrente prodotta da ciascun fotodiodo mediante un amplificatore transresistivo, in modo da adattare la dinamica del segnale a quella del convertitore A/D del microcontrollore. A questo scopo è importante la scelta delle resistenze, poiché esse determinano l'entità dell'amplificazione del segnale il cui valore non deve superare il limite di tensione massimo che il convertitore A/D è in grado di convertire (5 V). Poichè l'intensità luminosa del sole non è costante tutto l'anno, è da verificare se sia necessario modificare il valore delle resistenze per garantire un migliore sfruttamento della dinamica del convertitore A/D, pertanto si è provveduto a inserire degli zoccoletti per agevolare la sostituzione dei resistori. In figura 3.5 è inoltre visibile un cavo multipolare a 7 poli che, oltre a trasportare i 4 segnali dei fotodiodi, provvede ad alimentare il circuito fornendo una tensione duale di ± 12 V e la massa.

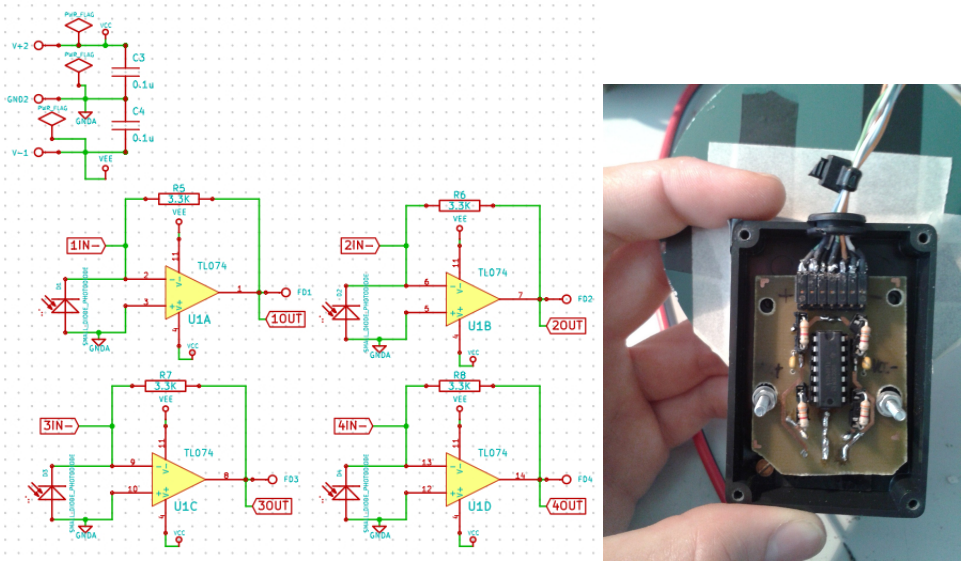

Figura 3.5: Schema elettrico da Kicad e foto del circuito realizzato

3.2 Algoritmo di calcolo delle coordinate altazimutali

L'algoritmo che ci permette di ricavare le coordinate altazimutali del sole si basa sul calcolo del "baricentro" delle 4 tensioni dei fotodiodi. Infatti la singola tensione prodotta da un fotodiodo non ci dà nessuna informazione sulla direzione di provenienza dei raggi solari, se non altro perchè dipende dall'intensità luminosa del sole nell'istante di misura, intensità che non è prevedibile in quanto, ad esempio, può essere influenzata dalla presenza di nuvole. L'idea è quindi di usare 4 fotodiodi uguali posizionati ognuno in un quadrante diverso del piano e osservare lo sbilanciamento che si crea fra le 4 tensioni quando il sole non è perfettamente a picco sui fotodiodi.

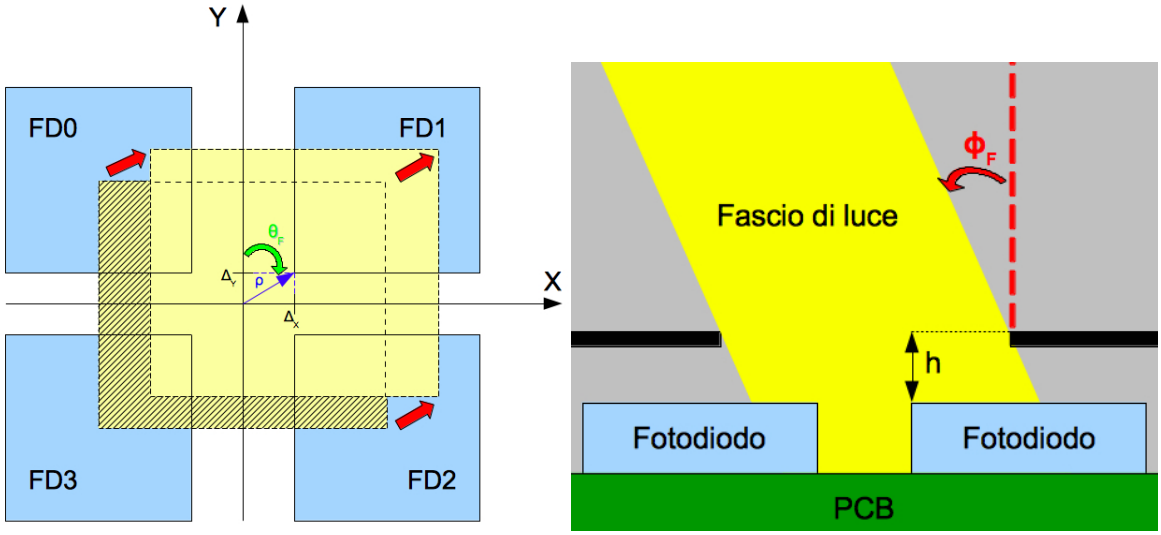

Figura 3.6: Il baricentro delle tensioni ρ e gli angoli θ_F e ϕ_F

Seguendo la numerazione presente in figura 3.6 e indicando con V_i la tensione prodotta dall' i-esimo fotodiodo si procede calcolando lo sbilanciamento Δ_X sull'asse X e Δ_Y sull'asse Y:

$$\begin{cases} \Delta_X = & \frac{L}{2} \times \frac{V_1+V_2-V_0-V_3}{V_0+V_1+V_2+V_3} \\ \Delta_Y = & \frac{L}{2} \times \frac{V_0+V_1-V_2-V_3}{V_0+V_1+V_2+V_3} \end{cases}$$

Traducendo questa informazione in coordinate polari, il modulo del vettore baricentro risulta $|\rho| = \sqrt{\Delta_X^2 + \Delta_Y^2}$, mentre l'angolo, misurato in senso orario a partire dall'asse Y, si ottiene come $\theta_F = \arctan\left(\frac{\Delta_X}{\Delta_Y}\right)$, tenendo presente quale sia il quadrante corretto da scegliere in base al segno di Δ_X eΔ_Y. L'angolo di inclinazione dei raggi solari rispetto alla perpendicolare al piano si ricava come $\phi_F = \arctan\left(\frac{|\rho|}{h}\right)$, dove $h = 1\,mm$ è la distanza fotodiodi-maschera e con ϕ_F appartenente al primo quadrante.

É importante notare che il funzionamento da "meridiana elettronica" si ha soltanto posizionando il blocco sensore in modo tale da avere l'asse Y orientato a NORD e la perpendicolare al piano dei fotodiodi diretta verso lo

zenit; in questa condizione infatti si ha che θ_F rappresenta l'angolo azimutale del sole (a meno di un angolo di 180°), mentre ϕ_F l'angolo zenitale.

I grafici delle figure 3.7 e 3.8 mostrano l'andamento degli angoli azimutale e zenitale del sole, dalle 11:30 alle 15:30 circa, calcolati con la meridiana elettronica raffrontandoli con i valori forniti dall'algoritmo numerico di Roberto Grena. Com'è possibile osservare, gli andamenti degli angoli calcolati con la meridiana elettronica approssimano molto bene quelli ricavati con l'algoritmo numerico, a meno di un errore che difficilmente supera i 2°; nel caso specifico di ϕ_F esso appare come un semplice offset, pertanto risulterebbe facile una correzione via software.

I dati ricavati danno prova delle ottime potenzialità di questo metodo di misura completamente analogico, inoltre probabilmente gli errori di calcolo sono imputabili a imperfezioni dei fotodiodi, che per la dispersione dei parametri non saranno perfettamente identici, a imprecisioni realizzative della maschera e a incertezze di posizionamento della scatolina, eseguito a mano con tecniche abbastanza rudimentali (bussola e livella). Ci sono quindi ampi margini di miglioramento, potendo contare su tecniche realizzative più sofisticate e su componenti più affidabili.

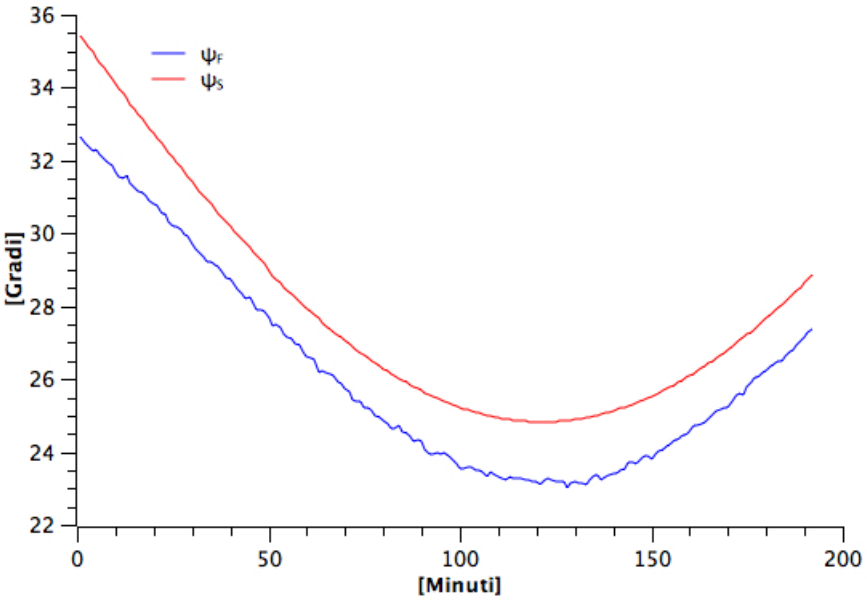

Figura 3.7: Angolo zenitale calcolato con la meridiana elettronica (ϕ_F) e con l'algoritmo numerico di R. Grena (ϕ_S)

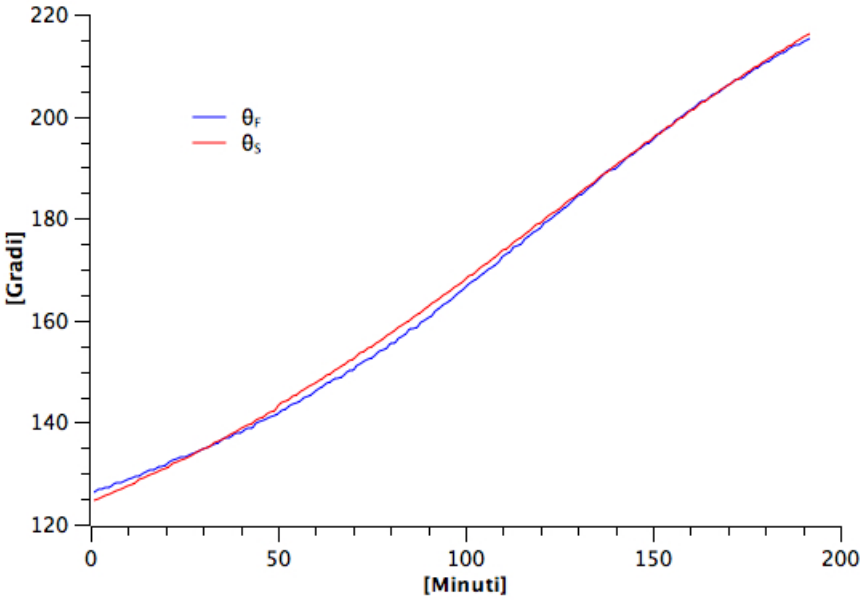

Figura 3.8: Angolo azimutale calcolato con la meridiana elettronica (θ_F) e con l'algoritmo numerico di R. Grena (θ_S)

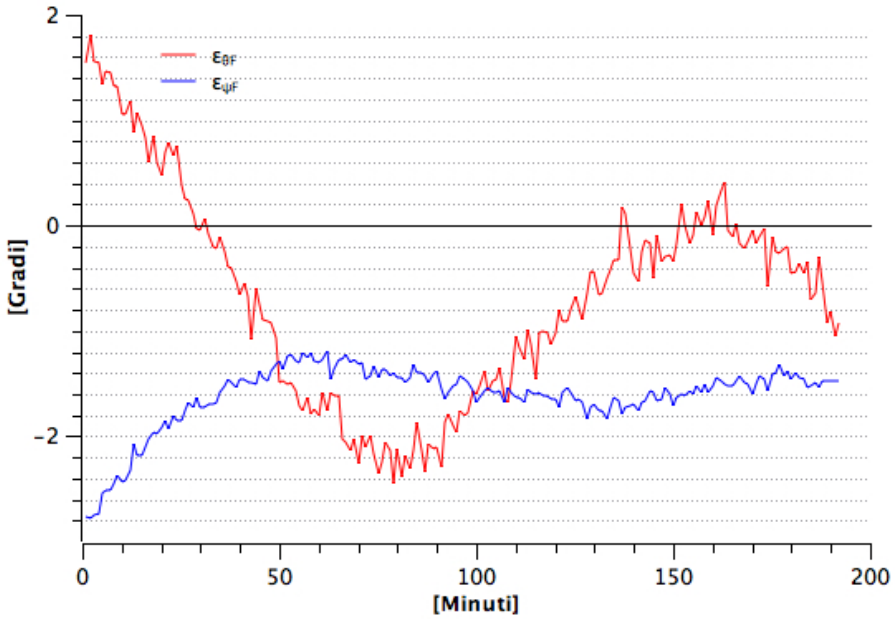

Figura 3.9: Errore commesso su θ_F e su ϕ_F

É importante mettere in evidenza come la forma geometrica della maschera sia determinante per la riuscita dell'algoritmo appena presentato: la scelta di una geometria di tipo circolare, per esempio, avrebbe comportato l'introduzione di non linearità influendo in maniera significativa sull'accuratezza dei risultati. Questo comportamento si verifica non appena ci si discosta dalla situazione di sole a picco sui fotodiodi ($\phi_F = 0°$) poiché, nel caso di geometria circolare, l'incremento di area illuminata su 1 o 2 fotodiodi non è bilanciato da un equivalente incremento di ombra sui restanti fotodiodi. Questo inconveniente non si verifica nel caso di apertura rettangolare, perché come è possibile osservare in figura 3.10, ad un aumento di area attiva illuminata corrisponde una diminuzione di pari valore su un'altra area attiva.

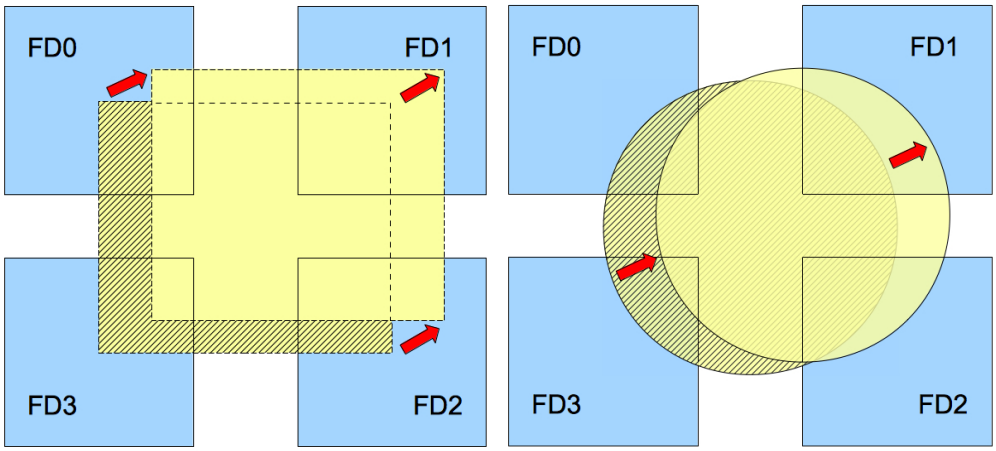

Figura 3.10: Proiezione nel caso di apertura rettangolare e circolare

3.3 Limiti di funzionamento

Il corretto funzionamento di questa tecnica è garantito sino a che i raggi solari illuminano contemporaneamente tutti e 4 i fotodiodi, quindi dipende dalla distanza che separa la proiezione della maschera, nella condizione di sole picco, con il bordo più vicino di uno dei fotodiodi e dallo spessore della lastra di vetro. Ciò determina in particolare un limite superiore sul calcolo dell'angolo di inclinazione ϕ_F, il cui valore limite, per $h = 1mm$, risulta :

$$\begin{cases} \phi_{F_{MAX}} = \arctan\left(\frac{|\rho_{MAX}|}{h}\right) \times \frac{180}{\Pi} = \arctan\left(\frac{L/2}{h}\right) \times \frac{180}{\Pi} = \arctan\left(\frac{1.5}{1.0}\right) \times \frac{180}{\Pi} \approx 56.31^\circ \\ |\rho_{MAX}| = \sqrt{\triangle^2_{X_{MAX}} + \triangle^2_Y} = \sqrt{\left(\frac{L}{2}\right)^2} = \frac{L}{2} \\ \triangle_{X_{MAX}} = \frac{L}{2} \times \frac{V_1+V_2-V_0-V_3}{V_0+V_1+V_2+V_3} = \frac{L}{2} \\ \triangle_Y = 0 \end{cases}$$

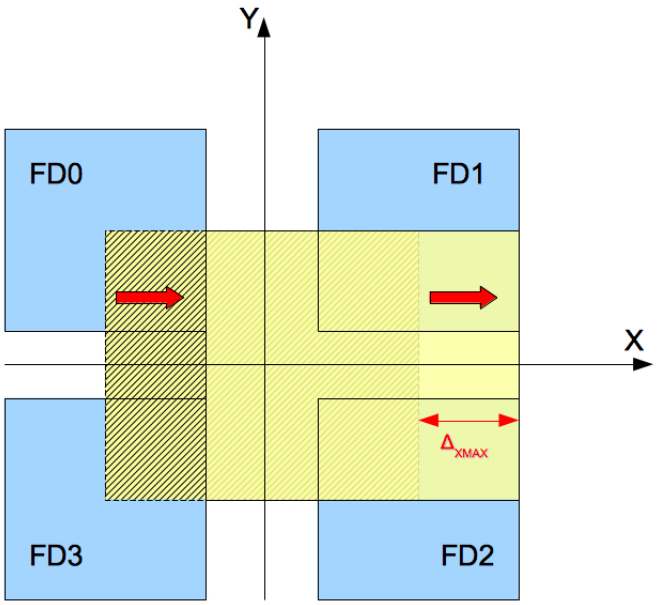

Figura 3.11: Condizione limite di funzionamento

Questa è la condizione limite perché se la proiezione si spostasse ancora lungo l'asse X non si avrebbe nessun cambiamento sul valore di $|\rho|$, di conseguenza ϕ_F rimarrebbe uguale a $\phi_{F_{MAX}}$. Ciò comporta un range di funzionamento della meridiana dipendente dall'altezza del sole sull'orizzonte, la quale oltre a variare durante una stessa giornata, raggiunge valori diversi durante l'anno e in base alla latitudine del luogo di misura. Alle nostre latitudini e nel mese di maggio, ad esempio, ϕ_S è più grande di $\phi_{F_{MAX}}$ prima delle 9:00 e dopo le 17:30 circa, quindi non abbiamo riscontrato particolari problemi durante i nostri esperimenti, essendo l'intervallo di validità molto ampio. Qualora fosse necessario estendere il range di funzionamento della meridiana elettronica, ciò sarebbe possibile utilizzando fotodiodi con aree attive più estese, in modo tale da incrementare contemporaneamente il $\triangle_{X_{MAX}}$ e il $\triangle_{Y_{MAX}}$, oppure impiegando spessori più piccoli.

3.4 Blocco di elaborazione

Il blocco di elaborazione dei dati ha la funzione di convertire in digitale i dati provenienti dal blocco sensore, eseguire alcune operazioni su di essi e infine inviarli al PC per successive analisi.

Il cuore di questa unità è il microcontrollore che è stato scelto perché garantisse il soddisfacimento dei requisiti tecnici che ci eravamo prefissati, in particolare:

- Convertitore A/D da almeno 10 bit di risoluzione e almeno 4 ingressi, per garantire una sufficiente sensibilità alle variazioni del segnale e per poter ricevere i 4 segnali dal blocco sensore;
- Sufficiente memoria per poter eseguire semplici operazioni sui dati (qualche Kbyte);
- Interfaccia RS232, per comunicare con il PC;

In ultima istanza, ma forse la più importante fra i requisiti, il microcontrollore doveva essere economico, pertanto la nostra scelta è ricaduta sul microcontrollore PIC18F4431 realizzato dalla Microchip, caratterizzato da un costo di vendita di soli 3.58 €.

L'altro componente importante del blocco di elaborazione è il chip UM232R, ovvero l'interfaccia USB-RS232 della FTDI, che ci ha permesso di:

- gestire in modo semplice e veloce la comunicazione fra PC e microcontrollore, sfruttando le funzioni di libreria che vengono fornite dal costruttore;
- fornire al microcontrollore l'alimentazione di 5 V una volta collegato al PC tramite cavo USB;
- fornire il clock di sistema (8 MHz), rendendo superfluo l'impiego di un quarzo esterno;

É importante evidenziare che l'utilizzo del (costoso) modulo UM232R è previsto soltanto in questa prima fase di debugging, in modo da potersi liberare momentaneamente delle problematiche connesse alla gestione dell'interfaccia RS232 e concentrare gli sforzi sugli altri aspetti della tesi.

La scheda, realizzata in tecnologia PCB, è visibile in figura 3.12 ed è stata realizzata con il proposito di essere *general purpose*, rendendo disponibili i pin del microcontrollore per un eventuale utilizzo a posteriori, in particolare prevedendo l'impiego del microcontrollore anche per il controllo dei motori elettrici del sistema di concentrazione. Sulla scheda sono presenti anche altri componenti (4 condensatori ceramici, 1 condensatore elettrolitico, 1 induttore, 3 resistori, 1 diodo, 1 jumper, 1 tasto e 1 connettore a 5 pin) con le funzionalità evidenziate in figura ed è inoltre possibile notare i 4 fili di diverso colore che permettono la ricezione dei 4 segnali dei fotodiodi del blocco sensore.

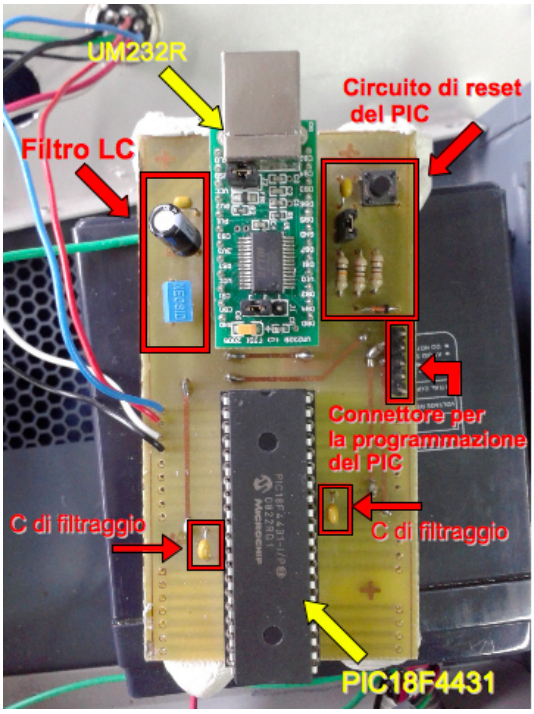

Figura 3.12: Foto scheda PCB del blocco di elaborazione

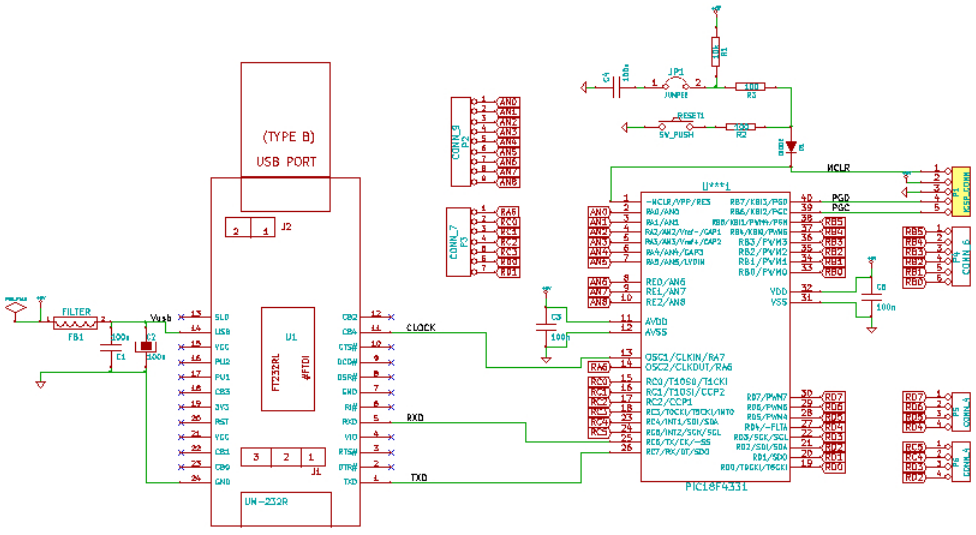

Figura 3.13: Schematico scheda *general purpose* realizzato con Kicad

3.5 Software

La funzionalità da meridiana elettronica è stata ottenuta effettuando una programmazione su due fronti:

- codice firmware del microcontrollore;
- codice lato PC.

3.5.1 Firmware

Il codice firmware è stato scritto in linguaggio C nell'ambiente di programmazione C18 della Microchip, che fornisce le librerie utili per la gestione dei propri dispositivi e permette il caricamento del codice nella memoria del microcontrollore tramite un accessorio, il *programmer*, che è venduto a parte ad un costo contenuto di circa 30 €. Nel nostro caso è stato impiegato il PICKit2, ancora valido per i nostri scopi, ma oramai non più in vendita, perché sostituito dalla versione successiva denominata PICKit3.

Le operazioni richieste al microcontrollore sono le seguenti:

1. Settaggio dei registri per consentire le operazioni successive, in particolare ci si è preoccupati configurare il convertitore A/D e i registri che regolano la comunicazione con il PC;
2. Attesa del comando di start da parte del PC per avviare il campionamento e la conversione A/D a 10 bit dei 4 segnali provenienti dai fotodiodi;
3. Al fine di ottenere una maggiore accuratezza di campionamento, per ogni canale vengono acquisiti 30 campioni su cui viene eseguita la media;
4. Invio al PC dei 4 campioni risultanti dalle medie;

5. Ritorno nello stato di attesa del comando di avvio da parte del PC per eseguire delle nuove acquisizioni (punto 2).

3.5.2 Codice lato PC

Il codice lato PC è stato sviluppato in linguaggio C++ per la gestione delle funzionalità della meridiana elettronica e per settare le modalità di comunicazione con il modulo UM232R della FTDI. I passi seguiti sono:

1. Verifica della connessione con la scheda di elaborazione;
2. Impostazione del baud rate opportuno (2400 baud);
3. Scelta del formato dei dati scambiati (8 bit);
4. Attesa inserimento da tastiera del tempo di attesa fra un'acquisizione e la successiva (60 s);
5. Invio comando di start al microcontrollore;
6. Ricezione dei 4 campioni;
7. Calcolo delle 4 tensioni prodotte dai fotodiodi, degli sbilanciamenti Δ_X e Δ_Y e degli angoli θ_F e ϕ_F;
8. Salvataggio dei risultati su file di testo in modo tale da renderli disponibili per successive analisi ed elaborazioni (ad es. grafici);
9. Passaggio nello stato di *sleep* per un numero di secondi pari al tempo di attesa inserito precedentemente da tastiera;
10. Ritorno al punto 5.

Capitolo 4

Concentratore solare

Una volta validato il funzionamento della meridiana elettronica, il passo successivo è stato pensare a come poterla sfruttare per raggiungere l'obiettivo di realizzare un concentratore solare. Per indirizzare i raggi del sole su di un bersaglio fisso, oltre alla posizione del sole è necessario conoscere sia la posizione del bersaglio, sia l'orientazione dello specchio secondo uno stesso sistema di riferimento. Secondo l'idea di base, sviluppata dal prof. Giovanni Pennelli, le 2 informazioni che ci servono possono essere ricavate prendendo come riferimento il sole, di cui sappiamo con elevata accuratezza le coordinate altazimutali, e servendoci della meridiana elettronica posizionata solidalmente allo specchio. In questa modalità di funzionamento, infatti, la meridiana elettronica consente di rilevare l'angolo ϕ_F di inclinazione dei raggi solari rispetto alla perpendicolare allo specchio e l'angolo θ_F, che fornisce indicazione sulla direzione di provenienza dei raggi rispetto al sistema di riferimento X-Y definito sul piano dei fotodiodi, che coincide con il piano dello specchio (fig. 4.1).

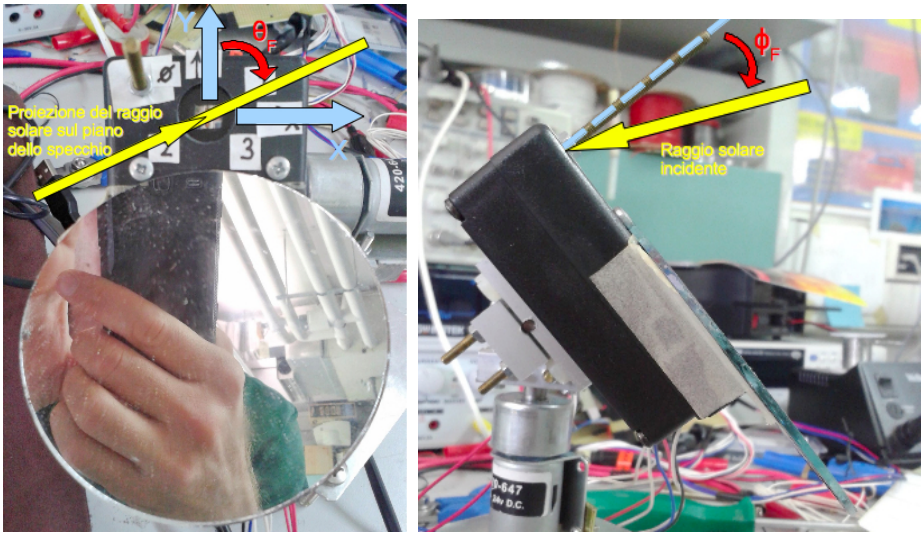

Figura 4.1: Angolo θ_F e angolo ϕ_F

Questi dati, assieme alla conoscenza dell'angolo azimutale θ_S e zenitale ϕ_S del sole, sono sufficienti per determinare la posizione del bersaglio. Ciò è fattibile applicando un algoritmo di "setup", che fa uso delle leggi della riflessione e dei noti teoremi di trigonometria sferica conosciuti come teoremi del seno e del coseno. La procedura preliminare da seguire è la seguente:

1. Orientare lo specchio in modo che punti la luce sul bersaglio: questa operazione può essere svolta azionando manualmente i motori, o in modo automatico servendosi di 4 fotodiodi piazzati sul bersaglio, che indicheranno il corretto puntamento soltanto quando sono tutti illuminati;

2. Acquisire i dati dalla meridiana elettronica per ricavarsi θ_F e ϕ_F;

3. Calcolare θ_S e ϕ_S mediante algoritmo numerico;

4. Applicare l'algoritmo di setup che, a partire da θ_F, ϕ_F, θ_S e ϕ_S, permette di ricavare θ_B e ϕ_B, ovvero le coordinate azimutali del bersaglio.

Essendo il bersaglio fisso, θ_B e ϕ_B rimarranno costanti nel tempo, mentre col passare dei minuti le coordinate altazimutali del sole cambieranno progressivamente, pertanto si verificherà che i raggi riflessi saranno sempre meno centrati sul target. Per riottenere la corretta concentrazione solare si dovrà operare la movimentazione dello specchio in modo da orientarlo opportunamente e a tal fine ci viene in aiuto un altro algoritmo, cosiddetto "operativo", che in sostanza compie il procedimento opposto rispetto all'algoritmo precedente, partendo dalla conoscenza di θ_B, ϕ_B, θ_S e ϕ_S per ricavarsi gli angoli $\theta_{F_{OK}}$ e $\phi_{F_{OK}}$, che la meridiana elettronica dovrà misurare una volta riposizionata tramite i motori elettrici. Il riposizionamento dello specchio viene fatto in modo automatizzato, grazie all'implementazione su microcontrollore di un algoritmo di controllo che, avendo in ingresso gli angoli θ_F e ϕ_F misurati contestualmente dalla meridiana elettronica, determina l'azionamento dei motori fino al raggiungimento della condizione $\theta_F = \theta_{F_{OK}}$ e $\phi_F = \phi_{F_{OK}}$, a meno di una certa soglia di errore. Vediamo ora nel dettaglio l'algoritmo di setup.

4.1 Algoritmo di setup

La trigonometria sferica viene impiegata dagli astronomi per effettuare cambi di sistemi di riferimento sulla sfera celeste. Lo strumento geometrico fondamentale in questo senso è il triangolo sferico, definito sulla sfera celeste come la figura racchiusa fra 3 archi generati dall'intersezione di 3 circoli massimi.

Sfera celeste: sfera di raggio arbitrario al cui centro è posta la Terra e sulla cui superficie viene visualizzata l'intersezione con la semiretta congiungente il centro della terra con l'oggetto celeste.

Circolo massimo: risultato dell'intersezione fra la superficie della sfera celeste e un piano passante per il suo centro; è detto "massimo" perchè è la più grande circonferenza ottenibile sulla superficie della sfera celeste.

I triangoli sferici godono delle seguenti proprietà:

1. I lati sono degli archi, pertanto si misurano in gradi;
2. La somma degli angoli ai vertici è sempre maggiore di 180° e sempre minore di 540°.

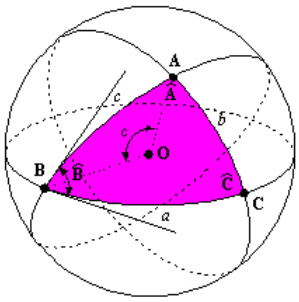

Figura 4.2: Triangolo sferico con vertici A, B e C [17]

Il lato di un triangolo sferico non è specificato pertanto dalla sua lunghezza lineare, bensì dall'angolo che sottende rispetto al centro O, ovvero è l'angolo piano definito fra le semirette passanti per il centro O e gli estremi del lato. Per angoli di un triangolo sferico si intendono invece gli angoli diedri compresi fra i piani dei circoli massimi che formano i lati del triangolo sferico.

Prendendo a riferimento la figura 4.3, caso in cui il sole si trova a destra del bersaglio secondo la prospettiva dell'osservatore nel punto O, per il calcolo di θ_B e ϕ_B si considera il triangolo sferico i cui vertici sono lo zenit (V), il punto di intersezione fra raggio solare passante per lo specchio e sfera celeste (S) e il punto di intersezione fra raggio riflesso e sfera celeste (B); inoltre è possibile individuare un ulteriore punto generato dall'intersezione con la perpendicolare al piano dello specchio (F) che, se collegato a V con un arco di cerchio massimo, divide il precedente triangolo sferico in altri 2 triangoli sferici: $S\hat{V}F$ e $F\hat{V}B$. Notare che gli archi $\overset{\frown}{SF}$ e $\overset{\frown}{FB}$ hanno stessa lunghezza

ϕ_F: questo è spiegato dalla seconda legge della riflessione, la quale afferma che l'angolo di incidenza e l'angolo di riflessione rispetto alla normale alla superficie riflettente sono uguali tra loro.

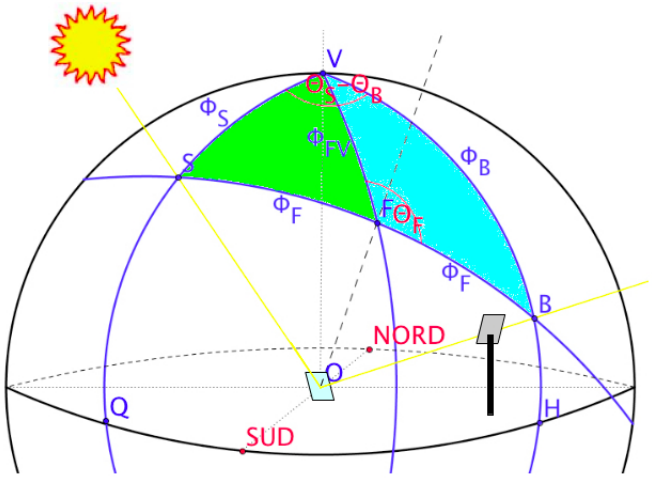

Figura 4.3: Triangoli sferici presi in considerazione nella definizione dell'algoritmo di setup

Arriviamo dunque ad enunciare l'algoritmo di setup. Come prima cosa si ricava l'arco ϕ_{FV} avvalendosi del teorema dei coseni della trigonometria sferica, il quale asserisce che *il coseno di un lato è uguale al prodotto dei coseni degli altri due lati più il prodotto dei seni degli stessi due lati per il coseno dell'angolo opposto al primo lato*, ovvero riportando il tutto in linguaggio matematico e applicandolo al triangolo sferico $S\hat{V}F$ di fig. 4.3:

$$\cos(\phi_S) = \cos(\phi_F) \times \cos(\phi_{FV}) + \sin(\phi_F) \times \sin(\phi_{FV}) \times \cos(\Pi - \theta_F) =$$

$$= \cos(\phi_F) \times \cos(\phi_{FV}) - \sin(\phi_F) \times \sin(\phi_{FV}) \times \cos(\theta_F)$$

Come si può notare la nostra incognita non è ottenibile in modo immediato, perché è presente sia nell'argomento del coseno, sia in quello del seno. Questo inconveniente può essere superato riscrivendo l'equazione in questo modo:

$$\cos(\phi_S) = m \times \left(\frac{\cos(\phi_F)}{m} \times \cos(\phi_{FV}) - \frac{\sin(\phi_F) \times \cos(\theta_F)}{m} \times \sin(\phi_{FV}) \right) =$$

$$= m \times [\cos(\xi) \times \cos(\phi_{FV}) - \sin(\xi) \times \sin(\phi_{FV})] = m \times \cos(\xi + \phi_{FV})$$

ϕ_{FV} è quindi ricavabile come:

$$\phi_{FV} = \arccos\left(\frac{\cos(\phi_S)}{m}\right) - \xi$$

con

$$\begin{cases} m = \sqrt{\cos^2(\phi_F) + \sin^2(\phi_F) \times \cos^2(\theta_F)} \\ \xi = \arccos\left(\frac{\cos(\phi_F)}{m}\right) \end{cases}$$

Sempre sfruttando il teorma del coseno è possibile infine ricavare ϕ_B e θ_B:

$$\begin{cases} \phi_B = \arccos[\cos(\phi_F) \times \cos(\phi_{FV}) + \sin(\phi_F) \times \sin(\phi_{FV}) \times \cos(\theta_F)] \\ \theta_B = \theta_S - \arccos\left[\frac{\cos(2\times\phi_F) - \cos(\phi_S)\times\cos(\phi_B)}{\sin(\phi_S)\times\sin(\phi_B)}\right] \end{cases}$$

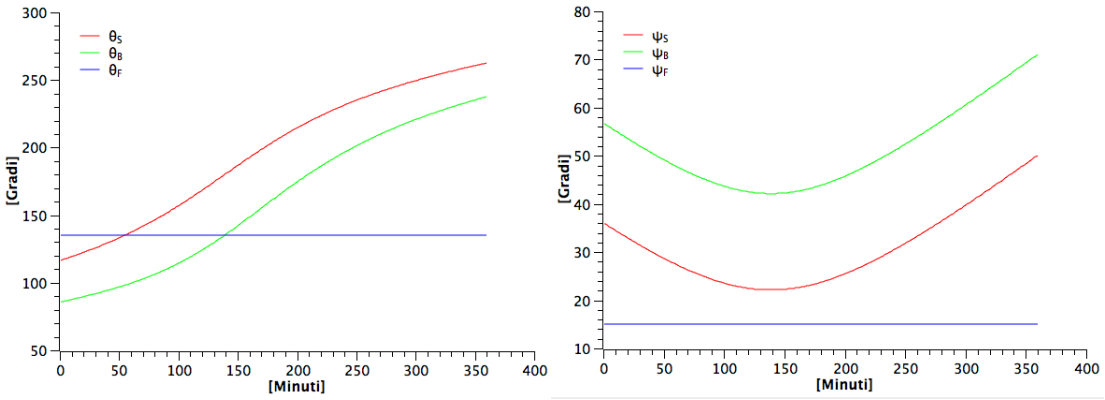

Figura 4.4: Simulazione a partire dalle 11:00 dell'algoritmo di setup per θ_F e ϕ_F costanti al variare di θ_S e ϕ_S

4.2 Algoritmo operativo

L'algoritmo operativo permette di calcolare gli angoli $\phi_{F_{OK}}$ e $\theta_{F_{OK}}$ che la meridiana elettronica dovrebbe misurare per far sì che lo specchio rifletta correttamente la luce del sole sul bersaglio. Ancora una volta si applica il teorema del coseno avendo a disposizione ϕ_S, θ_S, ϕ_B e θ_B:

$$\cos(2 \times \phi_{F_{OK}}) = \cos(\phi_S) \times \cos(\phi_B) + \sin(\phi_S) \times \sin(\phi_B) \times \cos(\theta_S - \theta_B)$$

pertanto

$$\phi_{F_{OK}} = \frac{\arccos[\cos(\phi_S) \times \cos(\phi_B) + \sin(\phi_S) \times \sin(\phi_B) \times \cos(\theta_S - \theta_B)]}{2}$$

il procedimento per arrivare a $\theta_{F_{OK}}$ parte dal calcolo degli angoli ai vertici del triangolo sferico $S\hat{V}B$, B e S, avvalendosi del teorema dei seni, il quale afferma che *il rapporto fra il seno di un angolo ed il seno del lato opposto è costante*:

$$\frac{\sin(B)}{\sin(\phi_S)} = \frac{\sin(\theta_S - \theta_B)}{\sin(2 \times \phi_{F_{OK}})}$$

$$\sin(B) = \frac{\sin(\phi_S) \times \sin(\theta_S - \theta_B)}{\sin(2 \times \phi_{F_{OK}})}$$

$$B = \arcsin\left[\frac{\sin(\phi_S) \times \sin(\theta_S - \theta_B)}{\sin(2 \times \phi_{F_{OK}})}\right]$$

$$\phi_{FV} = \arccos\left[\cos(\phi_{F_{OK}}) \times \cos(\phi_B) + \sin(\phi_{F_{OK}}) \times \sin(\phi_B) \times \cos(B)\right]$$

$$\theta_{F_{OK}} = \arcsin\left[\frac{\sin(\phi_B) \times \sin(B)}{\sin(\phi_{FV})}\right]$$

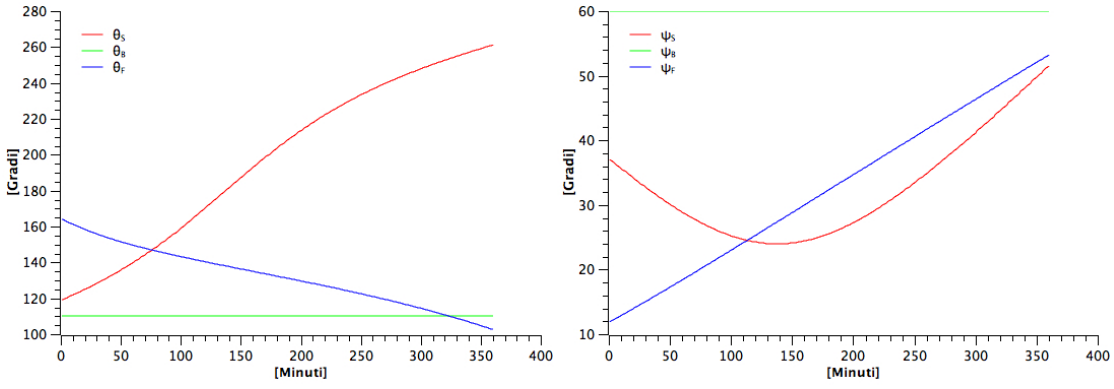

Figura 4.5: Simulazione a partire dalle 11:00 dell'algoritmo operativo per θ_B e ϕ_B costanti al variare di θ_S e ϕ_S

4.3 Descrizione del sistema microrobotico

Con l'ausilio degli algoritmi di setup e operativo è quindi possibile conoscere in modo automatico i valori $\theta_{F_{OK}}$ e $\phi_{F_{OK}}$ che la nostra meridiana elettronica deve "raggiungere". Il raggiungimento della condizione viene dettato da un algoritmo di controllo implementato su microcontrollore ed eseguito da 2 motoriduttori in corrente continua montati in modalità biassiale (fig. 4.6).

Figura 4.6: Specchio, meridiana elettronica e sistema di movimentazione biassiale

4.3.1 Motori

La scelta dei motori è stata particolarmente discussa, poichè le condizioni indispensabili per la nostra applicazione erano che i motori:

1. Avessero un passo il più piccolo possibile (in ogni caso inferiore a 1°), condizione necessaria per poter calibrare al meglio il puntamento;

2. Avessero una sufficiente coppia di esercizio per riuscire a muovere la struttura specchio più meridiana;

3. Conservassero la posizione raggiunta con una sufficiente coppia di mantenimento senza dover essere alimentati;

4. Non richiedessero troppa corrente per funzionare.

Gli ultimi due requisiti hanno particolare importanza in quanto, se non soddisfatti, potrebbero portare a un consumo di potenza tale da annullare i benefici del sistema a concentrazione. Con queste esigenze il motore che più di ogni altro sembrava adatto era quello di tipo stepper con passo di 0.9° o inferiore, trovando conferma nel fatto che è la tipologia attualmente più diffusa nei sistemi ad inseguimento solare. Ovviamente prestazioni di questo livello associate ad un motore di tipo passo passo si pagano con un elevato prezzo di acquisto: il costo in questo caso raggiunge facilmente le centinaia di euro. La soluzione è stata adottare un motore in continua, che sicuramente non nasce per applicazioni di precisione, ma che associato ad un riduttore 1024:1 e pilotato opportunamente diventa un valido sostituto del motore stepper. La riduzione inoltre presenta un doppio vantaggio:

1. Riduce la velocità angolare in uscita di 1024 volte, consentendo una maggiore accuratezza di posizionamento;

2. Aumenta consistentemente la coppia di esercizio e di mantenimento.

Il costo di un motore di questo tipo acquistato su RS è di soli 13.14 € (ma è possibile trovarne anche di più economici a 6 € circa), decisamente più basso del costo di qualsiasi motore stepper con passo da 0.9° in commercio, inoltre a suo vantaggio va a anche un consumo di corrente molto basso di 0.3 A nominali su un range di tensioni di alimentazione 12÷24 V per lo sfruttamento di tutta la coppia disponibile.

Un primo motore è stato montato perpendicolarmente ad un piedistallo di alluminio, mentre il secondo è stato fissato all'asse del primo in modo che

risultasse parallelo al piedistallo; la meridiana elettronica è stata avvitata all'asse del secondo motore e infine lo specchio è stato incollato alla superfice della scatolina della meridiana per garantire che il piano X-Y dei fotodiodi coincidesse con quello definibile sulla superficie dello specchio.

4.3.2 Scheda di elaborazione e controllo dei motori

Per il pilotaggio dei motori e la elaborazione dei dati viene riutilizzata la scheda *general purpose* impiegata per la validazione della meridiana elettronica. Per il nostro scopo infatti è sufficiente saldare 4 ulteriori fili per il pilotaggio dei motori e aggiungere 2 ponti H, che servono per ovviare alla ridotta corrente erogabile da ciascun pin del microcontrollore, insufficente al fine di alimentare i motori. Essi infatti necessitano di una corrente di 0.3 A, mentre la capacità dei pin del microcontrollore è limitata a 25 mA.

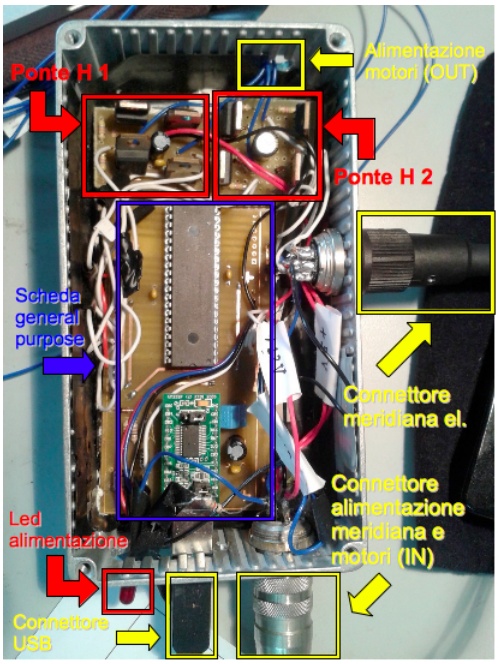

Figura 4.7: Scheda di elaborazione dati e controllo dei motori

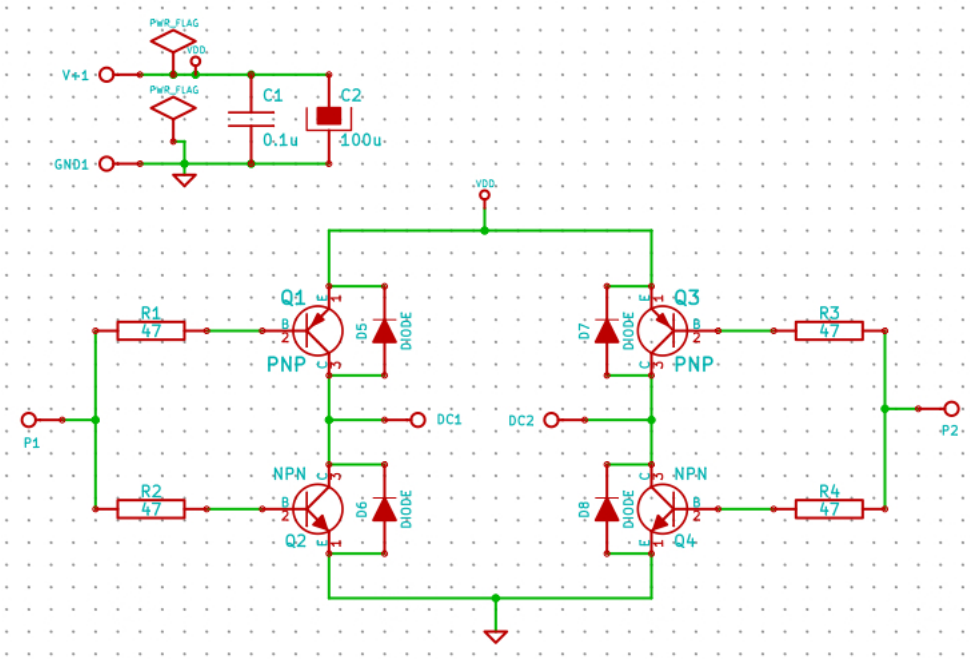

Figura 4.8: Schematico del ponte H utilizzato

Come si può notare dallo schematico di figura 4.8 il ponte H è nella configurazione che utilizza sia transistor di tipo NPN, sia di tipo PNP, grazie alla quale è possibile avere soltanto 2 ingressi invece di 4. Peculiarità di questo circuito è di consentire di scegliere il verso di rotazione del motore collegato ai terminali DC_1 a DC_2, semplicemente imponendo su uno dei pin di ingresso V_{DD} e sull'altro la massa del circuito. I componenti usati sono, per ogni ponte H:

- 2 transistor di potenza NPN TIP122;
- 2 transistor di potenza PNP TIP127;
- 4 diodi di ricircolo;
- 4 resistori da 47 Ω.

L'alimentazione del ponte H è a 5 V e, a meno delle tensioni di saturazione dei transistori, è la stessa tensione che va ad alimentare il relativo motore. Ciò potrebbe sembrare uno svantaggio, perchè in questo modo non si riesce a sfruttare al massimo la coppia del motore che nominalmente avrebbe bisogno di una tensione di valore compreso tra 12 V e 24 V; in realtà la riduzione di coppia non ci crea problemi, mentre si coglie il vantaggio di abbassare ancora la velocità di rotazione dell'asse del motore e il consumo del motore stesso

4.4 Algoritmo di controllo dei motori

La nostra applicazione necessita di un controllo molto fine della velocità di rotazione dell'asse dei motori per consentire l'aggiustamento dell'orientazione dello specchio in relazione allo spostamento del sole sulla sfera celeste, che è dell'ordine di 1° al minuto. Per ottenere questo risultato l'idea iniziale era di realizzare una sorta di controllo PWM (Pulse Width Modulation), applicando però il segnale modulato sempre sullo stesso pin di ingresso del ponte H, mentre sull'altro si impone alternativamente V_{DD} o massa, a seconda del verso di rotazione che si vuole impostare sul motore. Se ad esempio si imponesse la massa su P_2, l' "interruttore" Q_4 sarebbe chiuso e Q_3 sarebbe aperto, permettendo lo scorrere della corrente soltanto da DC_1 a DC_2; a questo punto entra in gioco il segnale PWM su P_1 che, in base al valore del duty cycle dell'onda rettangolare, regola l'intensità della corrente passante per Q_1 e Q_4 e quindi anche la velocità di rotazione del motore. Successivamente però l'idea iniziale è stata leggermente rivista, poiché questo tipo di pilotaggio dei motori non si comportava bene durante le prove sul campo. Il problema era infatti che la risposta dei motori ad un eventuale variazione del duty cycle del segnale PWM non era immediata, ma era soggetta ad una certa inerzia, che comportava malfunzionamenti soprattutto nel momento in cui il posizionamento dello specchio richiedeva spostamenti molto piccoli.

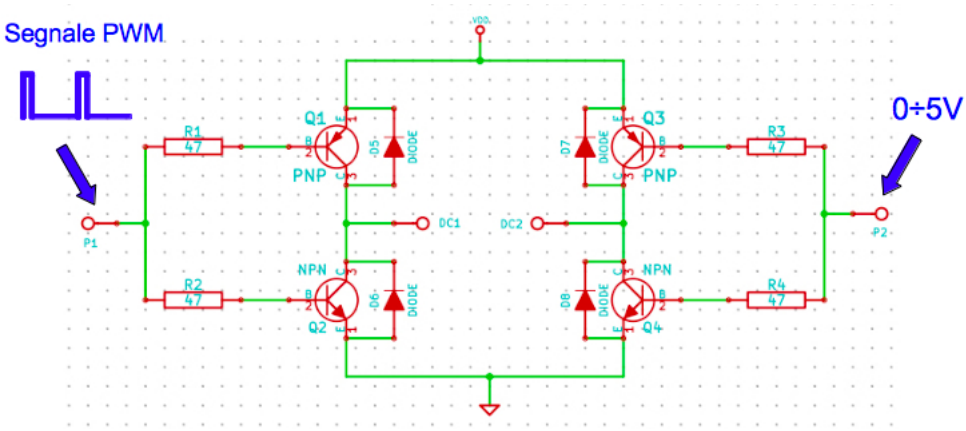

Figura 4.9: Esemplificazione dell'algoritmo PWM implementato

La soluzione è stata adottare un pilotaggio ad impulso, passando da una movimentazione continua dei motori ad una movimentazione a passi. Ciò ha permesso di eliminare l'inerzia dovuta al cambiamento del duty cycle dell'onda PWM, in quanto ad ogni passo i motori vengono fermati per poter analizzare nuovamente i dati provenienti dalla meridiana elettronica e decidere l'entità e il verso del passo. La durata dell'impulso è regolata secondo un controllo di tipo proporzionale, in base all'errore che si manifesta fra gli sbilanciamenti Δ_X e Δ_Y rispetto ai valori obiettivo $\Delta_{X_{OK}}$ e $\Delta_{Y_{OK}}$ ricavabili dagli angoli $\theta_{F_{OK}}$ e $\phi_{F_{OK}}$, che vengono calcolati in modo automatico tramite l'algoritmo operativo precedentemente presentato:

$$\begin{cases} \Delta_{Y_{OK}} = & h \times \tan\left(\phi_{F_{OK}}\right) \times \cos\left(\theta_{F_{OK}}\right) \\ \Delta_{X_{OK}} = & \tan\left(\phi_{F_{OK}}\right) \times \Delta_{Y_{OK}} \end{cases}$$

A questo punto l'algoritmo aziona un motore alla volta scegliendolo in base a quale sia lo sbilanciamento che è affetto dall'errore più grande: per modificare Δ_X l'algoritmo interviene sul motore con asse perpendicolare al terreno (motore 1), mentre per modificare Δ_Y viene azionato il motore con asse parallelo al terreno (motore 2), scegliendo il verso di rotazione opportuno a seconda del segno dell'errore. Questa scelta funziona e semplifica enormemente l'algorit-

mo di controllo, nonostante il movimento di un motore influisca, in maniera minore, anche sullo sbilanciamento associato all'altro motore. Lo scopo dell'algoritmo di controllo è dunque, passo dopo passo, di portare l'errore sugli sbilanciamenti al di sotto di una certa soglia e di mantenercelo, continuando ad analizzare il flusso continuo di dati provenienti dalla meridiana elettronica e riportando in posizione lo specchio qualora per qualche motivo dovesse perdere il corretto orientamento (ad esempio a causa del vento, o di eventuali imperfezioni meccaniche). La soglia di errore è stata introdotta per evitare che lo specchio fosse continuamente in movimento alla ricerca della posizione corretta, condizione ovviamente impossibile a causa dell'inevitabile presenza di errori all'interno del processo di calcolo. Le fonti di errore sono molteplici e, in particolare, dipendono da:

- Limitate precisione e accuratezza del convertitore A/D;
- Approssimazioni sulle variabili di tipo double e float introdotte negli algoritmi;
- Interferenze sul cavo multipolare, non schermato, che collega il sensore al microcontrollore.

Per raggiungere gli sbilanciamenti $\Delta_{X_{OK}}$ e $\Delta_{Y_{OK}}$, con la soglia di errore richiesta, il microcontrollore esegue pertanto i seguenti passi rappresentati nel diagramma di flusso di fig. 4.10.

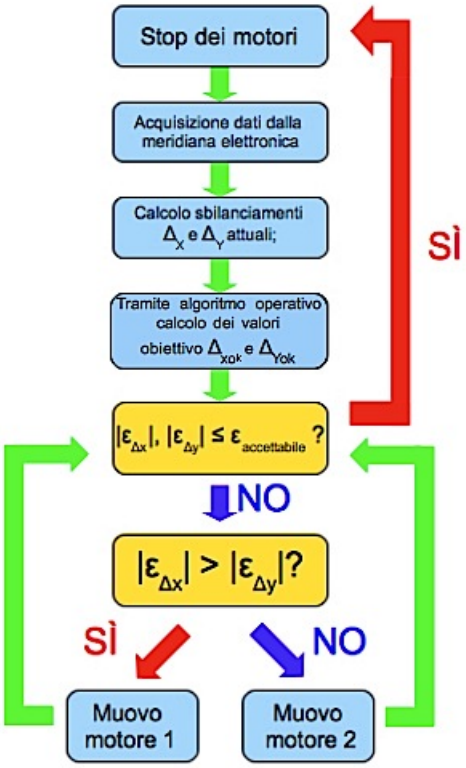

Figura 4.10: Diagramma di flusso dell'algoritmo di controllo dei motori

4.5 Software

Il lato sofware del sistema di concentrazione solare riprende quanto già fatto per la meridiana solare e ne estende le funzionalità; in particolare le possibilità di utilizzo sono:

1. Concentrazione solare;
2. Azionamento manuale dei motori;
3. Azionamento automatico dei motori per il raggiungimento e il mantenimento degli angoli $\theta_{F_{OK}}$ e $\phi_{F_{OK}}$ inseriti da tastiera;

4. Modalità di “verifica”;

5. Acquisizione dei campioni (funzionalità da meridiana elettronica).

É importante far notare che le funzionalità che fanno uso dell'algoritmo di calcolo numerico hanno comportato la necessità di appoggiarsi al PC per il calcolo della posizione del sole, poiché la limitata memoria del microcontrollore (16 Kbyte) non ha consentito l'implementazione diretta sullo stesso. Ciò comporta lo svantaggio sia tecnico, sia economico, di doversi preoccupare del cablaggio in daisy chain degli specchi che compongono il sistema di concentrazione, il cui numero potrebbe raggiungere il centinaio, per poter fornire la stessa informazione a tutti.

4.5.1 Concentrazione solare

La modalità di concentrazione solare è stata quasi completamente implementata su PC, per il motivo precedentemente enfatizzato della insufficiente memoria del microcontrollore da noi utilizzato, ed è una modalità totalmente automatizzata. L'utente è necessario soltanto in una prima fase di setup nel caso in cui si scelga la movimentazione manuale dei motori, poiché lo specchio deve essere posizionato in modo che punti correttamente il bersaglio; a quel punto è sufficiente impostare la funzionalità “concentratore solare” e la coppia microcontrollore-PC si preoccupa di gestire il movimento dei motori. In particolare il microcontrollore fornisce al PC i dati acquisiti dalla meridiana elettronica e successivamente riceve l'informazione sugli sbilanciamenti $\Delta_{X_{OK}}$ e $\Delta_{Y_{OK}}$ che deve far raggiungere alla meridiana tramite la movimentazione dei motori. Il PC invece è il “cervello” del sistema e si occupa di eseguire tutti i calcoli che sono stati illustrati nel presente capitolo.

4.5.2 Azionamento manuale dei motori

La movimentazione dei motori può avvenire nella modalità automatica o manuale. Nella modalità manuale l'utente, dopo aver inserito da tastiera

qual è il motore su cui agire, ha la possibilità di scegliere se il movimento dev'essere di tipo continuo, oppure passo passo: nel primo caso il motore si comporterà da normale motore in continua, mentre nel secondo caso avrà un comportamento simile a quello di un motore stepper che si muove di un passo per volta.

4.5.2.1 Movimentazione continua

Nella modalità classica di utilizzo di un motore DC l'utente può decidere quale sarà il verso e la velocità di rotazione dell'asse. La velocità viene controllata attraverso l'imposizione simultanea di un segnale PWM con duty cycle opportuno su uno dei terminali del ponte H e di una tensione costante pari a 0 V o 5 V sull'altro terminale, mentre l'inversione di marcia viene eseguita imponendo sui terminali di ingresso del ponte H tensioni complementari rispetto a quelle usate per l'altro senso di marcia.

4.5.2.2 Movimentazione passo passo

Nella movimentazione passo passo è possibile impostare il verso di rotazione e la lunghezza del passo. La lunghezza del passo si regola imponendo un impulso di durata opportuna sul terminale del ponte H riservato al segnale PWM, mentre sull'altro terminale si mantengono 0 V o 5 V, in base al verso di rotazione. La durata dell'impulso viene gestita attraverso una funzione di delay presente nella libreria predefinita <delays.h> del C18.

In figura 4.11 è visibile un diagramma di flusso esemplificativo.

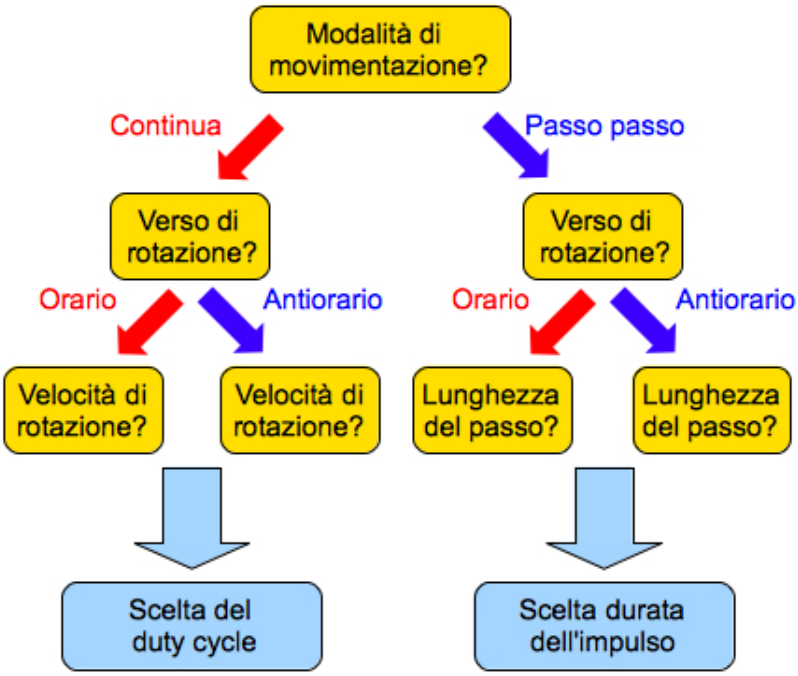

Figura 4.11: Possibilità di scelta nell'azionamento manuale dei motori

4.5.3 Azionamento automatico dei motori

L'azionamento automatico dei motori interviene qualora si voglia che lo specchio assuma una certa orientazione, ovvero raggiunga un posizionamento tale da far sì che la meridiana elettronica misuri gli angoli $\theta_{F_{OK}}$ e $\phi_{F_{OK}}$ inseriti dall'utente e li mantenga nel tempo. I motori vengono azionati automaticamente in base ai valori assunti dagli sbilanciamenti Δ_X e Δ_Y, aggiornati passo dopo passo ad ogni acquisizione della meridiana elettronica, secondo un controllo di tipo proporzionale. Nel dettaglio è il modulo degli errori $\varepsilon_{\Delta_X} = \Delta_X - \Delta_{X_{OK}}$ e $\varepsilon_{\Delta_Y} = \Delta_Y - \Delta_{Y_{OK}}$ a determinare la lunghezza del passo eseguito da ciascun motore, mentre è il segno che ne determina il verso.

Questa modalità è stata molto utile per testare il funzionamento dell'algoritmo di controllo dei motori utilizzato anche nella modalità di concentra-

zione solare e per testare il funzionamento dei sensori. Una semplice, ma non esaustiva, prova per validare i sensori è ad esempio verificare che lo specchio si affacci al sole una volta scelti i valori $\theta_{F_{OK}} = 0°$ e $\phi_{F_{OK}} = 0°$; a tal fine è stata aggiunta una vite perpendicolarmente alla superficie della meridiana come controprova dell'allineamento con i raggi del sole: qualora ciò non fosse verificato sarebbe visibile l'ombra della vite, tanto più lunga quanto peggiore è l'allineamento. Ovviamente la vite è stata rimossa nelle prove sperimentali in modalità "concentrazione solare" per evitare che l'ombra potesse ritrovarsi sopra ai fotodiodi, compromettendo le acquisizioni.

4.5.4 Modalità di verifica

Questa modalità è stata implementata in ultima istanza per avere un'ulteriore conferma, o meno, circa la bontà degli algoritmi di setup e operativo e sull'affidabilità della meridiana elettronica prima di eseguire le prove sperimentali della concentrazione solare. Si tratta di una funzionalità che ricalca quella di concentrazione solare per quanto riguarda la parte di calcolo degli sbilanciamenti, privata, però, della movimentazione automatica dei motori. Lo specchio viene inizialmente posizionato in modo da riflettere i raggi solari sul bersaglio, come nella modalità a concentrazione solare, e a quel punto viene avviata la modalità di verifica:

1. Il microcontrollore effettua l'acquisizione dei segnali prodotti dalla meridiana elettronica e calcola la posizione del bersaglio secondo il procedimento di calcolo dell'algoritmo di setup;

2. L'utente attende un tempo sufficiente (5-10 minuti) per far sì che il bersaglio non sia più puntato correttamente dai raggi solari;

3. A quel punto l'utente riposiziona lo specchio per portarlo a puntare nuovamente il bersaglio, momento in cui acquisisce e salva gli sbilanciamenti della meridiana e, contemporaneamente, salva gli sbilanciamenti

ricavati in modo automatico dall'algoritmo operativo a partire dalla conoscenza della posizione del bersaglio memorizzata precedentemente;

4. Ripetizione dei punti 2 e 3 per un numero di volte tale da garantire un numero soddisfacente di dati per il confronto dei dati ricavati manualmente e quelli ricavati tramite algoritmo.

4.6 Componenti utilizzati e costo del sistema

Uno degli aspetti fondamentali che è sempre stato tenuto in conto durante questo lavoro e che ha influenzato molte scelte è stata la massima attenzione verso il contenimento dei costi per la realizzazione del sistema a concentrazione. In tabella 4.1 è possibile visonare il *bill of materials* dei componenti utilizzati e il costo totale del sistema in versione prototipale.

	Componente	Funzione	Caratteristiche	Quantità	Costo unitario[€]	Costo Totale[€]
1	MICROCHIP - PIC18F4431	µcontrollore	DIP40	1	3.58	3.58
2	918D10241/1	Motoriduttore	1024:1, 12-24V, 0.3A	2	13.14	26.28
3	BP104F	Fotodiodo	-	4	0.444	1.78
4	TL084	OPAMP	-	1	0.345	0.345
5	TIP122	Amplificatore	Darlington NPN	2	0.45	0.9
6	TIP127	Amplificatore	Darlington PNP	2	0.336	0.67
7	Resistore	-	3.3KΩ, 5%, 0.25W	4	0.005	0.02
8	Resistore	-	4.7KΩ, 5%, 0.25W	8	0.005	0.04
9	Resistore	-	100Ω, 5%, 0.25W	2	0.005	0.01
10	Resistore	-	10KΩ, 5%, 0.25W	1	0.005	0.005
11	Condensatore	Filtraggio	Ceramico, 100nF	8	0.16	1.296
12	Condensatore	FIltraggio	Elettrolitico, 100µF	3	0.02	0.06
13	Induttore	Filtraggio	10mH	1	0.5	0.5
14	Diodo	Ricircolo e Blocco	-	9	0.02	0.18
15	Connettore	-	Vertical Pin Header, 40 pin	1	0.06	0.06
16	Cavo 7 poli	-	Non schermato	1	1	1
17	Scatolina nera	Contenitore	7.2×5.0×2.5cm	1	2	2
18	Tasto	Reset	-	1	0.55	0.55
19	Jumper	-	Spaziatura 2.54mm	1	0.25	0.25
20	Zoccolo PIC	-	40 pin	1	0.2	0.2
21	Zoccolo UM232R	-	24 pin	1	0.51	0.51
TOTALE						**40.24**

Tabella 4.1: Bill of materials del sistema in versione prototipale

Con un costo complessivo di circa 40 € sicuramente il prototipo conferma le aspettative di basso costo di realizzazione. Inoltre su alcune voci di costo è possibile intervenire sostituendo componenti con altri più economici: per esempio sarebbe possibile sostituire i motoriduttori in uso con altri (RE140) dal costo di appena 5.49 €, reperibili facilmente da RS, in modo tale da scendere a 24.94 € di costo complessivo del sistema prototipale: ovviamente nell'eventualità di una produzione su larga scala del sistema, il costo di tutti i componenti verrebbe ulteriormente ridotto di almeno un 50% portando il costo finale di produzione al notevole risultato di circa 10 €.

Capitolo 5

Prove sperimentali

La fase finale del lavoro di tesi ha riguardato la prova sul campo del sistema a concentrazione per poterne stabilire l'efficacia.

5.1 Setup sperimentale

Innanzitutto si è dovuto preparare un setup sperimentale adeguato composto da:

1. Uno specchio microrobotico;
2. Un bersaglio su cui puntare i raggi solari;
3. Il PC per il controllo e la verifica dei dati sperimentali.

La distanza scelta fra bersaglio e specchio è di 1.5 m, mentre l'altezza del bersaglio è di circa 1.25 m: è dunque possibile ricavarsi analiticamente l'angolo zenitale del bersaglio come $\phi_B = 90°\text{-}\arctan(\frac{1.25}{1.5}) \simeq 50,2°$ e, con una bussola, l'angolo azimutale (con minore accuratezza), in modo da stabilire subito un confronto con il risultato fornito dall'algoritmo di setup.

Il sito di misura è stato allestito nel giardino di fronte al laboratorio di Nanoelettronica del Dipartimento di Ingegneria dell'Informazione in via Caruso.

Figura 5.1: Immagine del setup sperimentale

Una volta configurato il setup sperimentale si è proceduto avviando la registrazione della videocamera e attivando la modalità di concentrazione solare da PC.

Inizialmente il puntamento del bersaglio è già corretto, perchè la modalità di concentrazione impone la preliminare orientazione, manuale o automatizzata (fig. 5.2), dello specchio in modo che rifletta sul bersaglio. Se, pertanto, gli algoritmi implementati risultano corretti, quello che deve verificarsi è il mantenimento della orientazione imposta inizialmente, ovvero i motori non devono muoversi, fino a quando l'algoritmo numerico per il calcolo delle coordinate altazimutali del sole non ne abbia aggiornato la posizione (1 volta al minuto, che significa circa 0.25° nel movimento del sole). A seguito di questo

aggiornamento non è detto che lo specchio venga riposizionato dai motori: ciò avverrà soltanto se la soglia di errore sugli sbilanciamenti è stata superata.

Figura 5.2: Per il posizionamento automatico dello specchio è necessario inserire 4 fotodiodi sul bersaglio

5.2 Risultati

Le prove effettuate hanno evidenziato luci e ombre del sistema a concentrazione. L'aspetto più evidente è che l'errore di puntamento sul bersaglio risulta maggiore di 4° già dopo circa un'ora di test e crescente nel tempo. Ciò è sicuramente negativo, ma comunque indica che la strada percorsa va nella giusta direzione, poichè la crescita dell'errore di puntamento è in qualche modo rallentata dalla reazione dei motori. Inoltre l'algoritmo di controllo dei

motori ha dato prova di comportarsi egregiamente, infatti se deliberatamente si crea un'ombra sui fotodiodi per falsare le acquisizioni della meridiana elettronica e causare quindi lo spostamento dello specchio dall'orientazione corretta, una volta che l'ombra viene tolta, lo specchio si riporta velocemente nella posizione di partenza.

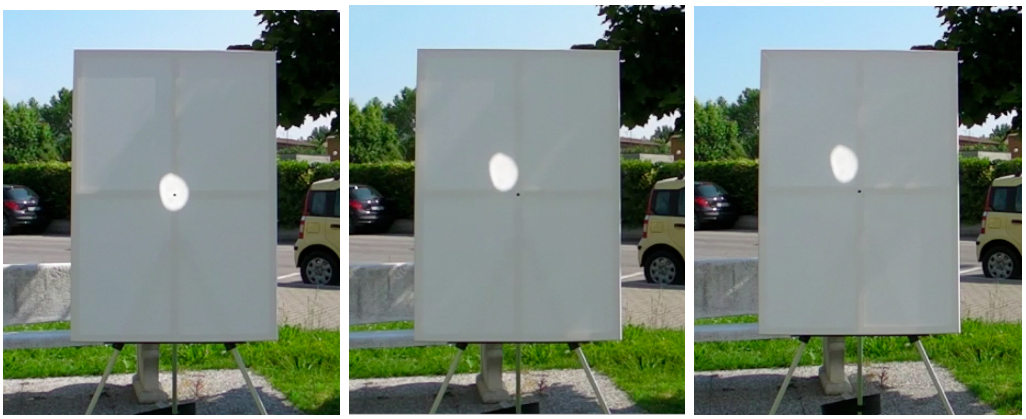

Figura 5.3: Puntamento iniziale, dopo 20 minuti e dopo 40 minuti

5.2.1 Verifica

Al fine di far emergere le possibile cause di errore verificatesi nelle prove sperimentali della concentrazione solare si è sfruttata la modalità di verifica, già descritta nel capitolo precedente.

Il tempo di misura è stato di circa 1 ora e 30 minuti, con intervalli fra un'acquisizione e la successiva di $5 \div 10$ minuti. Dal confronto fra gli sbilanciamenti "ideali" forniti dall'algoritmo operativo e quelli "reali" che, secondo la meridiana elettronica, sarebbero stati necessari per ottenere un puntamento corretto (figure 5.4 e 5.5), risulta che l'andamento reale non si discosta eccessivamente da quello ideale e, anzi, ci viene data la riprova della sostanziale correttezza della nostra implementazione teorica che, però, si scontra con le non idealità del dispositivo reale (imperfezioni geometriche della ma-

schera e nell'allineamento dei fotodiodi, dispersione dei parametri dei vari componenti, imprecisioni della conversione A/D, etc...).

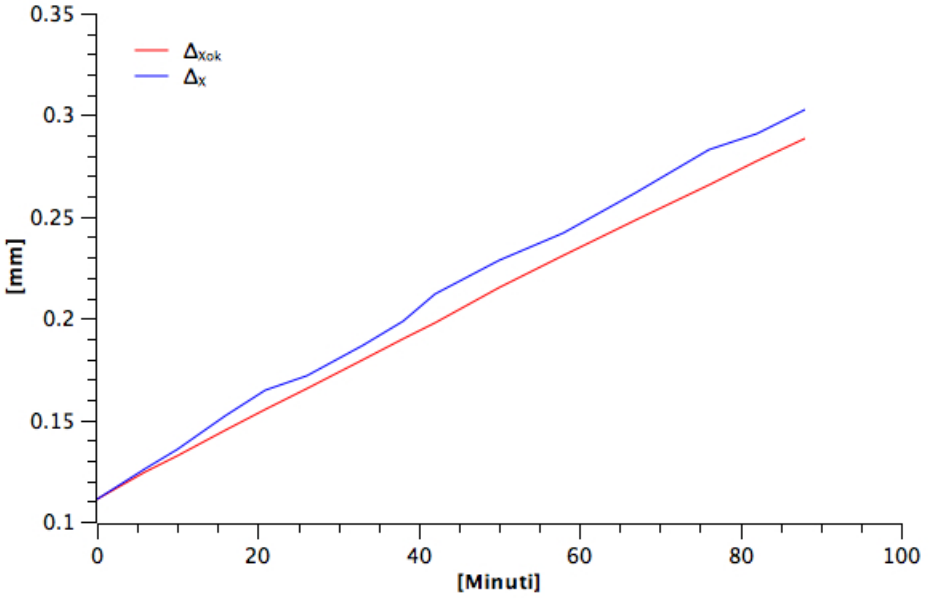

Figura 5.4: Confronto fra $\Delta_{X_{OK}}$ (simulato) e Δ_X (effettivo)

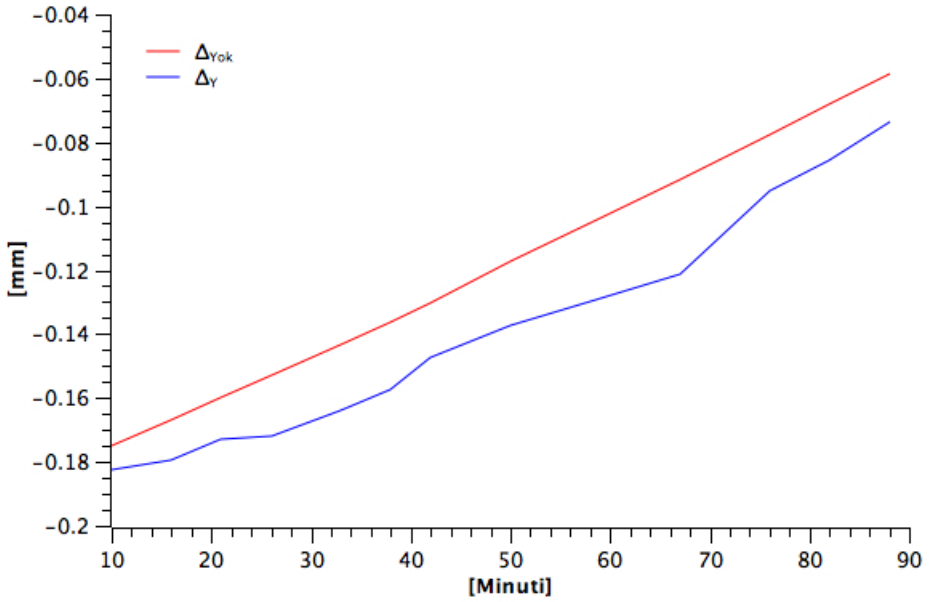

Figura 5.5: Confronto fra $\Delta_{Y_{OK}}$ (simulato) e Δ_Y (effettivo)

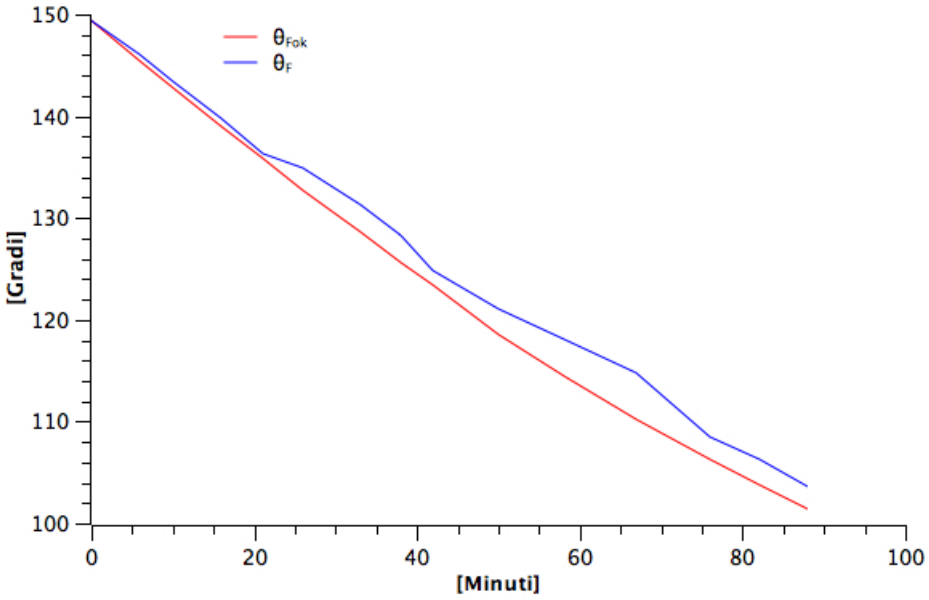

Figura 5.6: Confronto fra $\theta_{F_{OK}}$ e θ_F

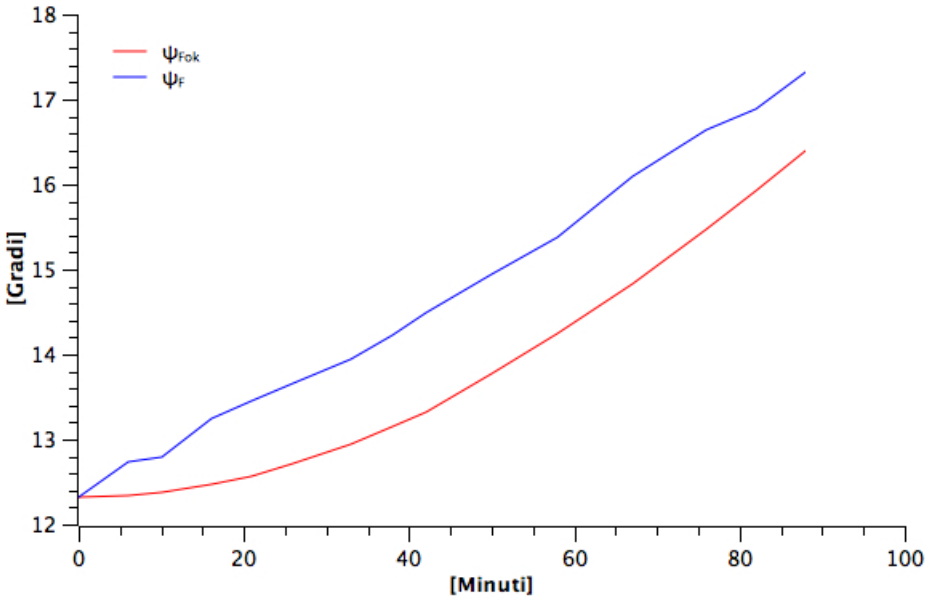

Figura 5.7: Confronto fra $\phi_{F_{OK}}$ e ϕ_F

Capitolo 6

Conclusioni

In conclusione si può affermare che lo specchio microrobotico per la concentrazione solare presentato è sicuramente non ancora ottimizzato e necessita di ulteriori perfezionamenti e indagini per minimizzare l'errore di puntamento. D'altro canto mostra ampi margini di miglioramento, potendo contare su componenti più affidabili (ad es. resistenze con tolleranza inferiore poste nel circuito di amplificazione del segnale dei fotodiodi) ed una più accurata realizzazione delle geometrie del sensore della meridiana elettronica. Inoltre il costo complessivo del sistema prototipale (24.94 € utilizzando i motori più economici a disposizione) risulta molto basso e, nell'ottica di una eventuale produzione su larga scala, è realistica una riduzione del costo al di sotto dei 10 €; in questo modo anche prevedendo l'impiego di un centinaio specchi per ottenere la concentrazione solare necessaria, il costo complessivo non supererebbe i 1000 €, cifra che se raffrontata con il costo di altri sistemi di puntamento e focalizzazione ad oggi in vendita, risulta molto contenuta.

Appendice

Codice C del PIC

```
#include <p18f4331.h>
#include <usart.h>
#include <adc.h>
#include <delays.h>
#include <math.h>


#pragma config WDTEN  = OFF     // Watchdog Timer
#pragma config WINEN  = OFF     // Watchdog Timer Window
#pragma config LVP    = OFF     // Low-Voltage ICSP Enable bit
#pragma config OSC    = ECIO    // Oscillator Selection (EXTERNAL)
#pragma config BOREN  = ON      // Brown-out Reset
#pragma config BORV   = 27      // Brown-out voltage = 2.7 volts
#pragma config PWRTEN = ON      // Power-Up Timer
#pragma config FCMEN  = OFF     // Fail-Safe Clock Monitor OFF
#pragma config IESO   = OFF     // Internal/External Switchover
#pragma config MCLRE  = ON      // MCLR Enable: RE3 Disabled
#pragma config PWM4MX = RB5     // MUXed with RD5
#pragma config PWMPIN = OFF     // PWM output pins RESET state control;
#pragma config HPOL   = HIGH    // PWM1, 3, 5 and 7 are active-HIGH
#pragma config LPOL   = HIGH    // PWM0, 2, 4 and 6 are active-HIGH
#pragma config FLTAMX = RC1     // FLTA su RC1


void PORTSconfig(void){
    // Imposto PORTA: RA0, RA1, RA2, RA3 ingressi (gruppi A,B,C,D)
    LATA  = 0x00;
    PORTA = 0x00;
    TRISA = 0b11111111;
    // Imposto PORTB tutti ingressi
    LATB  = 0x00;
    TRISB = 0x00;
    PORTB = 0b11111111;
    // Imposto PORTC tutte uscite a 0, tranne RX e TX ingressi
    LATC  = 0x00;
    TRISC = 0b11000000;
    PORTC = 0x00;
    // Imposto PORTD tutte uscite a 0
    LATD  = 0x00;
    TRISD = 0x00;
    PORTD = 0x00;
    // Imposto PORTE tutte uscite a 0
    LATE  = 0x00;
    TRISE = 0x00;
    PORTE = 0x00;

}
```

```
void INTconfig(void){   // Disables all interrupts
    RCONbits.IPEN = 0;
    INTCON  = 0x00;     // Disables all interrupts
    INTCON2 = 0x00;     // abilita pull-ups di portb
    INTCON3 = 0x00;
    PIE1 = 0x00;
    PIE2 = 0x00;
    PIE3 = 0x00;
    PIR1 = 0x00;
    PIR2 = 0x00;
    PIR3 = 0x00;
}

void ADCconfig(void){
    ADCON1 = 0b00010000; // Vref+=AVDD, Vref-=AVSS, ENABLE FIFO BUFFER
    ADCON2 = 0b10010001; // TAD = 8 Tosc = 800ns , Tacq = 64TAD = 52us, Fosc/8
    ADCON3 = 0b11000000; // All triggers disabled
    ANSEL0 = 0b11111111; // imposto RA0, RA1, RA2, RA3, RA4, RA5, RA6, RA7 come ingressi ANALOG
    ANSEL1 = 0b00000000; // e i restanti pin come DIGITAL I/O
}

void ADCch_sel(int canale){       //Funzione che seleziona modalita single shot e canale da convertire
    switch (canale){
        case 0:
            ADCHS  = 0b11111100; //selez AN0
            ADCON0 = 0b00000001; //Single-Shot and single channel enabled, GO/DONE = 0, A/D ON, group A
            break;
        case 1:
            ADCHS  = 0b11001111; //selez AN1
            ADCON0 = 0b00000101; //group B
            break;
        case 2:
            ADCHS  = 0b11110011; //selez AN2
            ADCON0 = 0b00001001; //group C
            break;
        case 3:
            ADCHS  = 0b00111111; //selez AN3
            ADCON0 = 0b00001101; //group D
            break;
    }
}

void main (void){
    unsigned char s = 'w', a = 1, w = 0, u, x, y;
    unsigned char i, j, t, k = 'a', motore, passi, dato, modalita, PDCL, PDCH, g = 0;
    int canale[4] = {0,0,0,0}, somma, somma1;
    float THETAf_OK, PHIf_OK, erroreaccettabileDx, erroreaccettabileDy;
    float DeltaX, DeltaX1, DeltaY, DeltaY1, Dx, Dy;
    float erroreDeltaX, erroreDeltaY, erroreDeltaY1, erroreDeltaX1;
    const float PI = 3.141592654;

    PORTSconfig();
    INTconfig();
```

```
ADCconfig();
baudUSART (BAUD_IDLE_CLK_HIGH &
           BAUD_16_BIT_RATE &
           BAUD_WAKEUP_OFF &
           BAUD_AUTO_OFF);

OpenUSART (USART_TX_INT_OFF &
           USART_RX_INT_OFF &
           USART_ASYNCH_MODE &
           USART_EIGHT_BIT &
           USART_CONT_RX &
           USART_BRGH_HIGH, 1249); // baud rate = 12000000/(4*(1249+1))= 2400, error = 0%


PTCON0 = 0b00000000;   //  free running mode, 1:1 prescale, 1:1 Postscale
PTCON1 = 0b00000000;   //  PWM time base is OFF

PTPERH = 0b00000001;   //  definisce il periodo del PWM: TPWM=(PTPER+1)xPTMR PostScaler/FOSC/4
PTPERL = 0b00000011;

PWMCON0 = 0b01011111;  //  ALL PWM I/O pins are enabled for PWM output; Independent mode
PWMCON1 = 0b00000001;  //  enables PWM update through PDCx registers.

OVDCOND = 0;           // override totale
OVDCONS = 255;         // tutti i PWM a 1 per stoppare i motori

PTCON1 = 0b10000000;        // PWM time base is ON
FLTCONFIG = 0b00000000;     // DISABILITO  fault  del pwm


/************************ Modalità MOTORI ************************/

while(1){    // CICLO INFINITO

    modalita = 'j';
    g = 0;

    while(!DataRdyUSART());
    motore = ReadUSART();

    while (BusyUSART());
    WriteUSART(motore);


    switch (motore){
        case 'x':
            /****************** THETAf_OK ************************************/
            while(!DataRdyUSART());                 // ATTENDO i primi 8 bit di THETAf_OK
            THETAf_OK = (unsigned char) ReadUSART();
            while (BusyUSART());                    // attendo che l'usart si liberi
            WriteUSART((int)THETAf_OK);

            while(!DataRdyUSART());                 // ATTENDO i secondi 8 bit di THETAf_OK
            dato = ReadUSART();
            THETAf_OK += (dato * 256);
```

```
            THETAf_OK = THETAf_OK/180.0*PI;          // THETA obiettivo del baricentro in rad
            while (BusyUSART());                     // attendo che l'usart si liberi
            WriteUSART(dato);

            /*****************************PHIf_OK*****************************/

            while(!DataRdyUSART());                    // ATTENDO PHIf_OK DA PC
            dato = ReadUSART();
            PHIf_OK = (float) dato/180.0*PI;

            while (BusyUSART());
            WriteUSART(dato);

            if (PHIf_OK == 0.0)
                THETAf_OK = 0.0;

            break;

        case 'v':
        case 'z':

            while(!DataRdyUSART());
            modalita = ReadUSART();
            WriteUSART(modalita);


            if ((modalita == 'o') || (modalita == 'a')){

                while(!DataRdyUSART());
                PDCL = ReadUSART();          // ricevo 8 bit meno significativi

                while(!DataRdyUSART());
                PDCH = ReadUSART();          // ricevo 8 bit più significativi
            }

            else if ((modalita == 'p') || (modalita == 'q')){

                while(!DataRdyUSART());                // ATTENDO num passi DA PC
                passi = (unsigned char) ReadUSART();   // ricevo numero passi
                WriteUSART(passi);
            }
    }

    // CALCOLO THETAf

    if (motore == 'x' || motore == 'c' || motore == 'm' || modalita == 's' || modalita == 'o'
        || modalita == 'a' || motore == 'k'){

    calcolo:

        // invio info su quale motore sto muovendo
        while(!DataRdyUSART());                    // ATTENDO start DA PC
        dato = ReadUSART();                        // libero buffer in rx
        while (BusyUSART());                       // attendo che il dato sia stato trasmesso
        WriteUSART(s);
```

```
for(i=0; i<4; i++)    // ad ogni richiesta di acquisizione reinizializzo l'array
    canale[i] = 0;


for (j=0; j<30; j++)              // faccio 30 acquisizioni per canale e poi ne faccio la media
    for(i=0; i<4; i++){
        ADCch_sel(i);             // Seleziona single shot mode, gruppo e canale e attiva l'ADC
        ConvertADC();             // CONVERTO IN DIGITALE IL CAMPIONE
        while(BusyADC());         // Wait for completion
        canale[i] += ReadADC();   // faccio la somma dei 30 campioni e poi divido per 30
    }


for(i=0; i<4; i++){
    canale[i] = (canale[i]/30.0 + 0.5);   //calcolo la media e approx all'intero + vicino

    while(!DataRdyUSART());
    dato = ReadUSART();                     // leggo RCREG così azzero il flag PIR1bits.RCIF
    while (BusyUSART());                    // attendo che il dato sia stato trasmesso

    WriteUSART(canale[i] % 256);            // la write invia 8 bit per volta

    while(!DataRdyUSART());
    dato = ReadUSART();
    while (BusyUSART());                    // attendo che il dato sia stato trasmesso

    WriteUSART(canale[i] >> 8);
}


if (motore == 'm')  // se sono in modalità meridiana solare ritorno ad acquisire dai fotodiodi
    goto calcolo;

if (modalita == 'p' && motore != 'x' && motore != 'c'){

    if (motore == 'v')
        goto motoreVO;
    else
        goto motoreZO;
}

if (modalita == 'q' && motore != 'x' && motore != 'c'){

    if (motore == 'v')
        goto motoreVA;
    else
        goto motoreZA;
}


/*************************ALGORITMO*****************************/
DeltaX1 = DeltaX;
DeltaY1 = DeltaY;
somma1 = somma;

somma = (canale[0] + canale[1] + canale[2] + canale[3]);
```

```
        // 1.5 = metà lato fotodiodo, NORD è asse Y positivo, EST è asse X positivo
        DeltaY = (float)(1.5) * (canale[0] + canale[1] - canale[2] - canale[3]) / somma;
        DeltaX = (float)(1.5) * (canale[1] + canale[3] - canale[0] - canale[2]) / somma;

        if (motore == 'c' || motore == 'x'){

            while(!DataRdyUSART());          // ATTENDO start DA PC
            dato = ReadUSART();              // 8 bit meno significativi del val ass di Dx x 14000
            WriteUSART (dato);               // lo reinvio per controllo
            Dx = (float) (dato / 14000.0);   // leggo primi 8 bit di THETAf_OK

            while(!DataRdyUSART());          // ATTENDO start DA PC
            dato = ReadUSART();              // salvo gli 8 bit più significativi
            WriteUSART (dato);
            Dx += (float) (dato / 14000.0 * 256);

            while(!DataRdyUSART());          // ATTENDO start DA PC
            dato = ReadUSART();              // salvo il segno di Dx (se dato = 1->segno negativo)
            WriteUSART (dato);
            if ((int)dato == 1)
                Dx = -Dx;

            while(!DataRdyUSART());          // ATTENDO start DA PC
            dato = ReadUSART();              // 8 bit meno significativi
            Dy = (float) (dato / 16000.0);   // leggo primi 8 bit di THETAf_OK

            while(!DataRdyUSART());          // ATTENDO start DA PC
            dato = ReadUSART();              // salvo gli 8 bit più significativi
            Dy += (float) (dato / 16000.0 * 256);

            while(!DataRdyUSART());          // ATTENDO start DA PC
            dato = ReadUSART();              // salvo il segno di Dx (se dato = 1->segno negativo)
            if ((int)dato == 1)
                Dy = -Dy;


            if (fabs (Dx) < 1.0 )
                erroreaccettabileDx = 0.005;
            else if (fabs (Dx) < 2.5)
                erroreaccettabileDx = 0.005 * fabs(Dx);
            else
                erroreaccettabileDx = 0.01 * fabs(Dx);

            if (fabs (Dy) < 1.0 )
                erroreaccettabileDy = 0.005;
            else if (fabs (Dy) < 2.5)
                erroreaccettabileDy = 0.005 * fabs(Dy);
            else
                erroreaccettabileDy = 0.01 * fabs(Dy);

            erroreDeltaX  = fabs(DeltaX - Dx);  //mi interessa il modulo
            erroreDeltaY  = fabs(DeltaY - Dy);
            erroreDeltaX1 = fabs(DeltaX1 - Dx);
            erroreDeltaY1 = fabs(DeltaY1 - Dy);
        }
    }
```

```
switch (motore) {
    case 'x':case'c':

        while (somma < 100.0){   // se la luce non è abb forte
            if (somma >= somma1){   // mi sono avvicinato dall'obiettivo
                if ((int)g == 0)              // se la causa è stata avere messo POUT2 a 0
                    OVDCONSbits.POUT2 = 0;    // rimetto POUT2 a 0 altrim POUT3 a 0
                else
                    OVDCONSbits.POUT3 = 0;
            }
            else{                             // mi sono allontanato dall'obiettivo:
                if ((int)g == 0){             // se la causa è POUT2 = 0 (g=0)
                    OVDCONSbits.POUT3 = 0;  // allora metto POUT3 = 0 (g=1)
                    g = 1;
                }
                else{
                    OVDCONSbits.POUT2 = 0;
                    g = 0;
                }
            }
            Delay10KTCYx(0);   // uso un passo bello lungo (con 0 è il max possibile)
            OVDCONS = 255;
            s = 'a';           // a = condizione di luce scarsa
            goto calcolo;
        }

        if (erroreDeltaY > erroreDeltaX)
            goto controlloDeltaY;
        else
            goto controlloDeltaX;


    controlloDeltaY:

        if (erroreDeltaY > a * erroreaccettabileDy ){

            s = 'c';  //c = sono oltre il limite di errore accettabile per deltaY
            if (DeltaY > Dy)             // se sono a destra di Dy mi dovrò muovere a sinistra
                OVDCONSbits.POUT4 = 0;
            else
                OVDCONSbits.POUT5 = 0;


            if (erroreDeltaY > 1.5)
                Delay10KTCYx(0);
            else if (erroreDeltaY > 0.5)
                Delay10KTCYx(150);
            else if (erroreDeltaY > 0.03)
                Delay10KTCYx(100);
            else
                Delay10KTCYx((int)(1150*erroreDeltaY));

            OVDCONS = 255;
            goto calcolo;
        }
```

```
controlloDeltaX:

    if (erroreDeltaX > a * erroreaccettabileDx){

        s = 'e'; //e = sono oltre il limite di errore accettabile per deltaX

        if (DeltaX < Dx)   // mi sono avvicinato dall'obiettivo
            OVDCONSbits.POUT3 = 0;
        else
            OVDCONSbits.POUT2 = 0;

        if (erroreDeltaX > 1.5)
        ista    Delay10KTCYx(0);
        else if (erroreDeltaX > 0.5)
            Delay10KTCYx(150);
        else if (erroreDeltaX > 0.03)
            Delay10KTCYx(100);
        else
            Delay10KTCYx((int)(1150*erroreDeltaX));

        OVDCONS = 255;
        goto calcolo;
    }

    if (erroreDeltaX < a * erroreaccettabileDx &&
        erroreDeltaY < a * erroreaccettabileDy && somma > 100.0)
        s = 'f';  // condizione di obiettivo raggiunto

    goto calcolo;
    break;


case 'v':

    OVDCONDbits.POVD2 = 0;  // PWM2 override
    switch (modalita) {
        case 'o':
            OVDCONSbits.POUT2 = 1;  // PWM2 = 1
            OVDCONDbits.POVD3 = 1;  // PWM3 segnale PWM con duty cycle specificato da PC
            PDC1L = PDCL;
            PDC1H = PDCH;
            break;

        case 'a':
            OVDCONSbits.POUT2 = 0;
            OVDCONDbits.POVD3 = 1;
            PDC1L = PDCL;
            PDC1H = PDCH;
            break;

        case 's':
            OVDCONDbits.POVD3 = 0;  // PWM3 override
            OVDCONSbits.POUT2 = 1;  // PWM2 = 1
            OVDCONSbits.POUT3 = 1;  // PWM3 = 1
            break;
```

```
            case 'p':
                for (t = 0; t < passi; t++){

                    OVDCONDbits.POVD3 = 0;
                    OVDCONSbits.POUT2 = 1;  // PWM2 = 1
                    OVDCONSbits.POUT3 = 0;  // PWM3 = 0

                    Delay10KTCYx(5);        // passo minimo

                    OVDCONSbits.POUT3 = 1;  // PWM3 = 1

                    goto calcolo;

                motoreVO:
                    ;
                }
                break;

            case 'q':
                for (t = 0; t < passi; t++){

                    OVDCONDbits.POVD3 = 0;
                    OVDCONSbits.POUT2 = 0;  // PWM2 = 0
                    OVDCONSbits.POUT3 = 1;  // PWM3 = 1

                    Delay10KTCYx(5);

                    OVDCONSbits.POUT2 = 1;  // PWM2 = 1

                    goto calcolo;

                motoreVA:
                    ;

                }
                break;
        }
        break;

    case 'z':

        OVDCONDbits.POVD4 = 0;
        switch (modalita) {
            case 'o':
                OVDCONSbits.POUT4 = 1;
                OVDCONDbits.POVD5 = 1;
                PDC2L = PDCL;
                PDC2H = PDCH;
                break;

            case 'a':
                OVDCONSbits.POUT4 = 0;
                OVDCONDbits.POVD5 = 1;
                PDC2L = PDCL;
                PDC2H = PDCH;
                break;
```

```
                    case 's':
                        OVDCONDbits.POVD5 = 0;  // PWM5 override
                        OVDCONSbits.POUT4 = 1;  // PWM4 = 1, PWM5 = 1;
                        OVDCONSbits.POUT5 = 1;
                        break;

                    case 'p':
                        for (t = 0; t < passi; t++){

                            OVDCONDbits.POVD5 = 0;
                            OVDCONSbits.POUT4 = 1;  // PWM4 = 1
                            OVDCONSbits.POUT5 = 0;  // PWM5 = 0

                            Delay10KTCYx(5);

                            OVDCONSbits.POUT5 = 1;  // PWM5 = 1

                            goto calcolo;

                        motoreZO:
                            ;
                        }
                        break;

                    case 'q':
                        for (t = 0; t < passi; t++){

                            OVDCONDbits.POVD5 = 0;  // PWM5 override
                            OVDCONSbits.POUT4 = 0;  // PWM4 = 0
                            OVDCONSbits.POUT5 = 1;  // PWM5 = 1

                            Delay10KTCYx(5);

                            OVDCONSbits.POUT4 = 1;  // PWM4 = 1

                            goto calcolo;

                        motoreZA:
                            ;

                        }
                        break;
                }
            }
        }
        return;
    }
```

Codice lato PC

```
#include <iostream>
#include <ftd2xx.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <string.h>
# define PI 3.141592654
using namespace std;
void binario (char *bin, int decimale){
```

```
        int k = 0;
        while (decimale != 0){
                bin[k] = decimale % 2;
                k++;
                decimale = decimale / 2;
        }
}
double latitude=44.30;
double longitude=10.0; // LONGITUDINE RISPETTO A Monte Mario double DELTA=54.0;
void calcola_azimut_zenit(double anno, double mese, double giorno, double ore, double minuti, double *PHIs,
double *THETAs)
// CALCOLA LA POSIZIONE DEL SOLE (PHIs angolo zenitale E THETAs angolo azimutale)
// SECONDO L'ALGORITMO DI Roberto Grena
// PUBBLICATO IN Solar Energy 82 (2008) 462470
// anno, mese, giorno: OVVIA INTERPRETAZIONE.
// ora, minuti: ATTENZIONE!!! TEMPO UNIVERSALE (DI GREENWICH)
// UN'ORA INDIETRO RISPETTO AL TMEC (Tempo Medio Europa Centrale)
// MENO UNA EVENTUALE ORA DI TEMPO LEGALE
// RICORDARE!!!!! RESTITUISCE PHIs DA 0 A 360 GRADI, 0 E' LA DIREZIONE DEL NORD.
{
double tg;
double tt;
double ut;
double sum;
double ly,lm,lh,lp,ll;
double t2;
double deltagamma, epsilon, gamma, delta_THETAs;
double THETAs_solar, delta_solar;
double hh;
double THETAs_topocentric,delta_topocentric;
double hh_topocentric, ch_topocentric, sh_topocentric;
double e_zero;
double PHIs_finale, THETAs_finale;

    ut=ore+minuti/60.0;      // TEMPO UNIVERSALE ESPRESSO IN ORA E FRAZIONE DI ORA.
    if(mese==1 || mese ==2){
                mese=mese+12.0;
                anno=anno-1;
    }
    tg= floorf(365.25*(anno-2000.0))+floorf(30.6001*(mese+1))+giorno+ut/24.0-1158.5;
    tt=tg+DELTA/86400.0; // DA CHIARIRE IL DELTA
    sum=1.72019e-2*tt-0.0563;
    ly=1.74094+1.7202768683e-2*tt+3.34118e-2*sin(sum)+3.448e-4*sin(2.0*sum);
    lm=3.13e-5*sin(0.2127730*tt-0.585);// MOON PERTURBATION
    lh=1.26e-5*sin(4.243e-3*tt+1.46)+2.35e-5*sin(1.0727e-2*tt+0.72)+2.76e-5*sin(1.5799e-2*tt+2.35)
       +2.75e-5*sin(2.1551e-2*tt-1.98)+1.26e-5*sin(3.1490e-2*tt-0.8);    // HARMONIC CORRECTION
    t2=0.001*tt;
    lp=(((-2.30796e-7*t2+3.7976e-6)*t2-2.0458e-5)*t2+3.976e-5)*t2*t2;    // POLYNOMIAL CORRECTION
    ll=ly+lm+lh+lp;
        deltagamma=8.33e-5*sin(9.252e-4*tt-1.173);
    epsilon=-6.21e-9*tt+0.409086+4.46e-5*sin(9.252e-4*tt+0.397);
    gamma=ll+PI+deltagamma-9.932e-5;  // GEOCENTRIC SOLAR LONGITUDE
    THETAs_solar=atan2(sin(gamma)*cos(epsilon),cos(gamma));
    delta_solar=asin(sin(epsilon)*sin(gamma));
    hh=6.30038809903*tg+4.8824623+0.9174*deltagamma+longitude/180*PI-THETAs_solar;
    delta_THETAs=-4.26e-5*cos(latitude/180.0*PI)*sin(hh);
        // TOPOCENTRIC SUN COORDINATES
    // RIGHT ASCENSION
    THETAs_topocentric=THETAs_solar+delta_THETAs;
    // DECLINATION
    delta_topocentric=delta_solar-4.26e-5*(sin(latitude/180.0*PI)-delta_solar*cos(latitude/180.0*PI));
    // TOPOCENTRIC HOUR ANGLE
```

```
    hh_topocentric=hh-delta_THETAs;
    ch_topocentric=cos(hh)+delta_THETAs*sin(hh);
    sh_topocentric=sin(hh)-delta_THETAs*cos(hh);
    // SOLAR ELEVATION ANGLE nota!!!! NO REFRACTION CORRECTIONS!!!!
    e_zero=asin(sin(latitude/180.0*PI)*sin(delta_topocentric)+
                        cos(latitude/180.0*PI)*cos(delta_topocentric)*ch_topocentric);
    // REMEMBER!!!!!! NON REFRACTION CORRECTIONS !!!!!!!!!
    PHIs_finale=PI/2.0-e_zero;
    THETAs_finale=atan2(sh_topocentric,ch_topocentric*sin(latitude/180.0*PI)-
                                        tan(delta_topocentric)*cos(latitude/180.0*PI));
    *PHIs=PHIs_finale/PI*180.0; // RISULTATO IN GRADI SESSADECIMALI
    *THETAs= 180.0 + THETAs_finale/PI*180.0;
        // RICORDARE (VEDERE L'ARTICOLO): PHIs_finale=0 SE DIREZIONE SUD
        // FINE!!!
        return;
}


void algoritmo_operativo(double phiB, double thetaB, double phiS, double thetaS, double *PHIf_OK,
double *THETAf_OK, double *PhiFV){
        double aux1, aux2, PhiFVa, PhiFVb, sinB, sinS, B, S, phiFV, phiF_OK, thetaF_OK;
        aux1 = cos(phiS) * cos(phiB);
        if (thetaB < thetaS)
                aux2=sin(phiS)*sin(phiB)*cos(thetaS-thetaB);
        else
           aux2 = sin(phiS) * sin(phiB) * cos(thetaB-thetaS);
        phiF_OK = (acos (aux1 + aux2))/2.0;  // "PHIf obiettivo" che lo specchio deve raggiungere
        *PHIf_OK = phiF_OK;
        if (thetaB < thetaS)
                sinB = sin(phiS) * sin(thetaS - thetaB) / sin(2.0 * phiF_OK);
        else
                sinB = sin(phiS) * sin(thetaB - thetaS) / sin(2.0 * phiF_OK);
        if (phiF_OK == 0)
                sinB = 0;

        B = asin( sinB );
        if (thetaB < thetaS)
                sinS = sin(phiB) * sin(thetaS - thetaB) / sin(2.0 * phiF_OK);
        else
                sinS = sin(phiB) * sin(thetaB - thetaS) / sin(2.0 * phiF_OK);
        S = asin(sinS);
        PhiFVa = acos( cos(phiF_OK) * cos(phiB) + sin(phiF_OK) * sin(phiB) * cos(B) );
        PhiFVb = acos( cos(phiF_OK) * cos(phiS) + sin(phiF_OK) * sin(phiS) * cos(S) );
        if (sinB>sinS)
                phiFV = PhiFVb;
        else
                phiFV = PhiFVa;
        *PhiFV = phiFV;
        if (phiS < phiB)
                thetaF_OK = PI - asin( sin(phiB) * sinB / sin(phiFV) );//"THETAf obiettivo". (baricentro)
        else
                thetaF_OK = asin( sin(phiB) * sinB / sin(phiFV) );
        *THETAf_OK = thetaF_OK;
 }


void algoritmo_di_setup (double phiF, double thetaF, double phiS, double thetaS, double *PHIb,
double *THETAb, double *PhiFV){
        double modulo, sinCHI, cosCHI, CHI, PHIfv, phiB, thetaB;
        modulo = sqrt( pow(cos(phiF), 2) + pow(sin(phiF), 2) * pow(cos(thetaF), 2) );
        sinCHI = sin(phiF)*cos(thetaF)/modulo;
        cosCHI = cos(phiF)/modulo;
```

```
        if (sinCHI > 0 && cosCHI > 0)         //primo quadrante
                CHI = acos( cos(phiF)/modulo );
    else if (sinCHI < 0 && cosCHI > 0)
                CHI = -acos( cos(phiF)/modulo ); //quarto quadrante
        else
                cout << "errore sul CHI \n\n\n";
        PHIfv =  acos( cos(phiS) / modulo ) - CHI;
        *PhiFV = PHIfv;
        phiB = acos( cos(phiF) * cos(PHIfv) + sin(phiF) * sin(PHIfv) * cos(thetaF) );
        *PHIb = phiB;
        thetaB = thetaS - acos(  ( cos(2*phiF) - cos(phiS) * cos(phiB) ) / ( sin(phiS) * sin(phiB) )
);
        *THETAb = thetaB;
}


/*************************MAIN*****************************************************************/
int main() {

        /*****************************DICHIARAZIONE VARIABILI*************************************/
        FT_STATUS ftstatus1 = 0, ftstatus2 = 0;
    FT_HANDLE fthandle, fthandle1, fthandle2;
    DWORD byteinviati; //, devindex = 0;        DWORD codice = 210; // --> codice di start qualsiasi
        DWORD dutyL, dutyH, bytericevuti;  //duty cyle PWM e passi in modalita passo-passo
        DWORD numDevs = 0;  // num dispositivi connessi
        char *descrPointer[3];
        int duty_cycle;
    unsigned int passi = 1;
    char descrittore1[64], descrittore2[64];     //descrittore dispositivi connessi
    unsigned char modalita = 'k';
        unsigned char motore, rxchar, txchar, info, datoTx1, datoTx2;
        char rapBin[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //rappresentazione binaria del codice ricevuto
        unsigned int num_acquisizioni = 0;
        int angolo, angoloRX, ValInteroDx, ValInteroDy;
        double  modulo, CHI, sinB, DxPIC;
        double THETAf_OK = 0.0, PHIf_OK=0.0, aux1, aux2, B;
        double THETAb, PHIb, THETAs, PHIs;
    double PhiFV; //PhiFV è l'angolo fra la verticale passante per lo zenit e l'asse Y dei FD.
        int rxbuff[4] = {0,0,0,0}; //buffer per ricezione 4 campioni di 16 bit ciascuno dal pic
    double tensione[4], DeltaX, DeltaY, D, THETAf1, PHIf1, THETAf2, PHIf2;
        double erroreTHETAf1, errorePHIf1, somma, deltaTheta, deltaPhi, erroreTHETAf2, errorePHIf2;
    double erroreDeltaX, erroreDeltaY, Dx, Dy, erroreaccettabileDx, erroreaccettabileDy;
        const double h = 1.0; // distanza dei fotod dal foro esterno
        const double valoreMAX = 1023.0; //1111111111 è il max valore che può assumere il campione a 10 bit
    const double Vref = 5.0; //tensione di riferimento del convertitore A/D
        double THETAf3;
        double sinCHI, cosCHI, PhiFVa, PhiFVb, S, sinS;
        int num_acquisizioniK = 0;
        FILE *stream;
    time_t rawtime;
    struct tm * timeinfo;
        double anno, mese, giorno, ore, minuti, minuti1;        //double ore, minutes;
        /************************SETUP UM232*************************************/
        descrPointer[0] = descrittore1;
        descrPointer[1] = descrittore2;
        descrPointer[2] = NULL;
    do  {        // verifico se il dispositivo è collegato

                ftstatus1 = FT_ListDevices(descrPointer, &numDevs, FT_LIST_ALL|FT_OPEN_BY_SERIAL_NUMBER);
                if (ftstatus1 == FT_OK && numDevs != 0){
                        cout << "DISPOSITIVI COLLEGATI " << numDevs << '\n';
                        cout << "NUM seriale1 = " << descrPointer[0]<<'\n';
```

```
                        if (numDevs == 2)
                                cout << "NUM seriale2 = " << descrPointer[1]<<'\n';
                }
                else{
                        cout<< "Dispositivi non trovati!"<<'\n'
                                <<"Premere un tasto e invia per continuare..."<<'\n';
                  char tasto_wait;
                        cin >> tasto_wait;
                }
        }while (numDevs == 0);


ftstatus1 = FT_OpenEx(descrittore1, FT_OPEN_BY_SERIAL_NUMBER, &fthandle1); // dispositivo1
if(ftstatus1 == FT_OK)
                cout << '\n' << "TUTTO BENE, DISPOSITIVO 1 APERTO\n";
        else
                cout << "PROBLEMA CON L'APERTURA DEL DISPOSITIVO 1\n\n\n";

ftstatus1 = FT_SetBaudRate(fthandle1, 2400); // Set baud rate to 2400
if (ftstatus1 == FT_OK)
                cout << "DISP1: BAUD RATE SETTATO\n";
else
                cout << "DISP1: Errore di settaggio baud rate\n\n";

ftstatus1 = FT_SetDataCharacteristics (fthandle1, FT_BITS_8, FT_STOP_BITS_1, FT_PARITY_NONE);
if (ftstatus1 == FT_OK)
                cout << "DISP1: FORMATO DATI SETTATO A 8 BIT\n" << flush;
else
                cout << "DISP1: Errore di settaggio formato dati\n\n";

        ftstatus1 = FT_SetTimeouts(fthandle1, 5000, 1000);
        if (ftstatus1 == FT_OK)
                cout << "DISP1: TIMEOUTS: FT_Read = 5s, FT_Write = 1s\n\n";
        else
                cout << "DISP1: ERRORE SETTAGGIO TIMEOUTS FT_Read e FT_Write\n\n";


        if (numDevs == 2) {
                ftstatus2 = FT_OpenEx(descrittore2, FT_OPEN_BY_SERIAL_NUMBER, &fthandle2); // dispositivo2
                if(ftstatus2 == FT_OK)
                        cout << "TUTTO BENE, DISPOSITIVO 2 APERTO\n";
                else
                        cout << "PROBLEMA CON L'APERTURA DEL DISPOSITIVO 2\n\n\n";

                ftstatus2 = FT_SetBaudRate(fthandle2, 2400); // Set baud rate to 2400
                if (ftstatus2 == FT_OK)
                        cout << "DISP2: BAUD RATE SETTATO\n";
                else
                        cout << "DISP2: Errore di settaggio baud rate\n\n";

                ftstatus2 = FT_SetDataCharacteristics (fthandle2, FT_BITS_8, FT_STOP_BITS_1, FT_PARITY_NONE);
                if (ftstatus2 == FT_OK)
                        cout << "DISP2: FORMATO DATI SETTATO A 8 BIT\n" << flush;
                else
                        cout << "DISP2: Errore di settaggio formato dati\n\n";

                ftstatus2 = FT_SetTimeouts(fthandle2, 5000, 1000);
                if (ftstatus2 == FT_OK)
                        cout << "DISP2: TIMEOUTS: FT_Read = 5s, FT_Write = 1s\n\n";
                else
                        cout << "DISP2: ERRORE SETTAGGIO TIMEOUTS FT_Read e FT_Write\n\n";
        }
```

```
        if (strcmp(descrPointer[0], "FTV8BBN2") == 0){   //num seriale dell'um232 della meridiana
                if (numDevs == 2){
                        fthandle = fthandle1;
                        fthandle1 = fthandle2;
                        fthandle2 = fthandle;
                        cout << "ho invertito gli fthandle\n\n";
                }
                else
                        cout << "HAI INSERITO LA SCHEDA PER IL BERSAGLIO!
                                        INSERIRE LA SCHEDA PER LO SPECCHIO\n\n";
        }

                /************************CICLO INFINITO*****************************/
inizio:
        while (1) {
                do {
                        cout << "Seleziona QUALE MOTORE comandare ('v' o 'z'),
                                        RAGGIUNGI E MANTIENI THETA e PHI ('x'),
                                        CONCENTRAZIONE SOLARE ('c'), o MERIDIANA SOLARE ('m'),
                                        o VERIFICA ('k') ==> ";
                        cin >> motore;
                        cout << "\n";
                        if (motore!='v' && motore!='z' && motore!='x' && motore!='c'
                                && motore!='m' && motore!='k')
                                cout << "Scelta non possibile\n\n";
                } while (motore!='v' && motore!='z' && motore!='x' && motore!='c'
                             && motore!='m' && motore!='k');

                ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
                ftstatus1 = FT_Write(fthandle1, &motore, 1, &byteinviati);   // invio scelta motore
                if (!FT_SUCCESS(ftstatus1))                         cout << "\nErrore di scrittura\n";
                ftstatus1 = FT_Read(fthandle1, &rxchar, 1, &bytericevuti); // ricevo OK dal pic
                if (FT_SUCCESS(ftstatus1))
                        cout << rxchar << "\n\n";  //VERIFICO CHE MI SIA RITORNATO IL DATO INVIATO
                switch(motore){
                        case 'x':
                                /************************ANGOLO THETA************************/
                                do {
                                        cout << "Seleziona l'angolo THETA intero del baricentro
                                                rispetto al NORD, ovvero l'asse x del piano dei FD
                                                (EST=90 SUD=180 OVEST=270) [0, 360] ==> ";
                                //90=proiezione dei raggi sull'OVEST-->sole a EST
                                        cin >> angolo;
                                        cout << "\n";
                                        if (angolo < 0 || angolo > 360)
                                                cout << "Scelta non possibile\n\n";
                                } while (angolo < 0 || angolo > 360);

                                txchar = angolo % 256;     // 8 bit meno significativi
                                ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
                                ftstatus1 = FT_Write(fthandle1, &txchar, 1, &byteinviati);
                                if (!FT_SUCCESS(ftstatus1))
                                        cout << "\nErrore di scrittura\n";
                                ftstatus1 = FT_Read(fthandle1, &rxchar, 1, &bytericevuti);
                                angoloRX = rxchar;
                                txchar = angolo >> 8;
                                ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
                                ftstatus1 = FT_Write(fthandle1, &txchar, 1, &byteinviati);
                                if (!FT_SUCCESS(ftstatus1))
                                        cout << "\nErrore di scrittura\n";
                                ftstatus1 = FT_Read(fthandle1, &rxchar, 1, &bytericevuti);
                                angoloRX += rxchar * 256;
```

```
                        if (FT_SUCCESS(ftstatus1))
                                cout << angoloRX << "\n\n";

                        THETAf_OK = (double)angolo / 180.0*PI;  //in radianti.
                        //rappresenta l'angolo fra la proiez dei raggi del sole
                        // sul piano dei FD e l'asse x nel piano dei FD
                        /**********************ANGOLO PHI**********************/
                        do {
                                cout << "Seleziona l'angolo PHI intero [-90, 90] ==> ";
// angolo calcolato a partire dalla perpendicolare al piano dei FD
                                cin >> angolo;
                                cout << "\n";
                                if (angolo < 0 || angolo > 90)
                                        cout << "Scelta non possibile\n\n";
                        } while (angolo < 0 || angolo > 90);
                        PHIf_OK = double(angolo)/180.0*PI; // in radianti
                        if (PHIf_OK == 0.0 && THETAf_OK != 0.0){
                                cout << "Hai inserito PHI = 0, questa scelta ammette
                                        soltanto Theta = 0. Theta sarà imposto a 0";
                                THETAf_OK = 0.0;
                        }
                        ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
                        ftstatus1 = FT_Write(fthandle1, &angolo, 1, &byteinviati);// invio PHIf_OK
                        if (!FT_SUCCESS(ftstatus1))
                                cout << "\nErrore di scrittura\n";
                        ftstatus1 = FT_Read(fthandle1, &rxchar, 1, &bytericevuti);// ricevo OK dal pic
                        if (FT_SUCCESS(ftstatus1))
                                cout << (int)rxchar << "\n\n";

                        break;

                case 'c':
                        cout << "OCCHIO CHE L'ULTIMA DOMENICA DI MARZO E OTTOBRE C'è IL CAMBIO ORA
                                        SOLARE-LEGALE E VICEVERSA!\n";
                        cout << "CON L'ORA LEGALE BISOGNA TOGLIERE UNA UNITà ALL'ORA CHE
                                        SI PASSA ALL'ALGORITMO COMPLESSO\n\n ";
                        break;

                case 'v':
                case 'z':
                        do {
                                cout << "Seleziona il comando fra i seguenti:\n\n";
                                cout << "- o: Rotazione CONTINUA in senso ORARIO\n";
                                cout << "- a: Rotazione CONTINUA in senso ANTIORARIO\n";
                                cout << "- p: Rotazione PASSO PASSO in senso ORARIO\n";
                                cout << "- q: Rotazione PASSO PASSO in senso ANTIORARIO\n";
                                cout << "- s: stop\n\n";


                                cin >> modalita;
                                cout << "\n";
                                if (modalita!='o' && modalita!='a' && modalita!='p' &&
                                                modalita!='q'&& modalita!='s')
                                        cout << "Comando non riconosciuto\n\n";
                        } while (modalita!='o' && modalita!='a' && modalita!='p' &&
                                                modalita!='q'&& modalita!='s');


                        ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
                        //invio modalita di rotazione o stop
                        ftstatus1 = FT_Write(fthandle1, &modalita, 1, &byteinviati);
                        if (!FT_SUCCESS(ftstatus1))
```

```
                                cout << "\nErrore di scrittura\n";
                        ftstatus1 = FT_Read(fthandle1, &rxchar, 1, &bytericevuti); // ricevo OK
                        if (FT_SUCCESS(ftstatus1))
                                cout << rxchar << "\n\n";


                        if (modalita == 'p' || modalita == 'q') {   // passo passo
                                cout << "Numero passi (min 0, max 255): ";
                                cin >> passi;
                                cout << endl;
                                ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
                                ftstatus1 = FT_Write(fthandle1, &passi, 1, &byteinviati); //invio passi
                                if (!FT_SUCCESS(ftstatus1))
                                        cout << "\nErrore di scrittura\n";
                                ftstatus1 = FT_Read(fthandle1, &rxchar, 1, &bytericevuti); // ricevo OK
                                if (FT_SUCCESS(ftstatus1))
                                        cout << (int)rxchar << "\n\n";
                        }


                        else if (modalita == 'o' || modalita == 'a') {   // rotaz continua
                                cout << "Inserire valore duty cycle
                        [0-640 per senso orario, 400-1280 per antiorario]: ";
                                cin >> duty_cycle;
                                cout << "\n\n";
                                dutyL = duty_cycle % 256;
                                dutyH = duty_cycle >> 8;
                                ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
                                ftstatus1 = FT_Write(fthandle1, &dutyL, 1, &byteinviati); //invio dutyL
                                if (!FT_SUCCESS(ftstatus1))
                                        cout << "\nErrore di scrittura\n";
                                ftstatus1 = FT_Write(fthandle1, &dutyH, 1, &byteinviati); //invio dutyH
                                if (!FT_SUCCESS(ftstatus1))
                                        cout << "\nErrore di scrittura\n";
                        }
                } // fine switch(motore)
                /******************ACQUISIZIONE DATI******************/
                for (int num_passi = 0; num_passi < passi; num_passi++) {
                acquisizioni:
                        // ricevo info su quale motore sto muovendo
                        ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
                        ftstatus1 = FT_Write(fthandle1, &codice, 1, &byteinviati);
                        ftstatus1 = FT_Read(fthandle1, &info, 1, &bytericevuti);
                        //al primo ciclo non viene stampato nulla a schermo perchè
//'info' contiene il valore 'w'
// v = motore deltaX
// z = motore deltay
                        if (info == 'a')
                                printf("LUCE SCARSA --> MUOVO 'V'");
                        else if (info == 'b')
                                printf("oltre ERR GROSSOLANO di deltaY --> MUOVO 'Z'");
                        else if (info == 'c')
                                printf("oltre ERR OK per deltaX --> MUOVO 'V'");
                        else if (info == 'd')
                                printf("oltre ERR GROSSOLANO per deltaX --> MUOVO 'V'");
                        else if (info == 'e')
                                printf("oltre ERR OK per deltaY  --> MUOVO 'Z'");
                        else if (info == 'f')
                                printf("OBIETTIVO RAGGIUNTO");

                        // ricevo campioni dai fotodiodi
                        for (int i=0; i<4; i++){
```

```
                        ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
                        //start acquisizione i-esimo campione
                        ftstatus1 = FT_Write(fthandle1, &codice, 1, &byteinviati);
                        // ricevo gli 8 bit meno significativi
                        ftstatus1 = FT_Read(fthandle1, &rxchar, 1, &bytericevuti);
                        rxbuff[i] = rxchar;
                        ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
                        // invio segnale di start
                        ftstatus1 = FT_Write(fthandle1, &codice, 1, &byteinviati);
                        // ricevo gli 8 bit più significativi
                        ftstatus1 = FT_Read(fthandle1, &rxchar, 1, &bytericevuti);
                        rxbuff[i] += rxchar*256;
                } //fine ciclo for

                /****************ALGORITMO MERIDIANA NUOVA*******************/
                // A QUESTO PUNTO LO SPECCHIO DEVE ESSERE POSIZIONATO
        //IN MODO DA RIFLETTERE SUL BERSAGLIO
                THETAf1 = THETAf2;   // al primo ciclo assumeranno dei valori casuali.
                PHIf1 = PHIf2;
                erroreTHETAf1 = erroreTHETAf2;
                errorePHIf1 = errorePHIf2;

                somma = (double) (rxbuff[0] + rxbuff[1] + rxbuff[2] + rxbuff[3]);
                DeltaY = (double) 1.5 * (rxbuff[0] + rxbuff[1] - rxbuff[2] - rxbuff[3]) / somma;
                DeltaX = (double) 1.5 * (rxbuff[1] + rxbuff[3] - rxbuff[0] - rxbuff[2]) / somma;
                D = sqrt(DeltaX * DeltaX + DeltaY * DeltaY);

                THETAf2 = atan2(DeltaX,DeltaY);
                // anche se deltaY non è perfettamente uguale a 0 pongo THETAf2 = 0
          if (fabs(DeltaY) < 0.1  && fabs(DeltaX) < 0.1)
                        THETAf2 = 0.0;
                PHIf2 = atan2(D,h);      // in radianti! RISPETTO ALLA PERPENDICOLARE AL FORO
                THETAf3 = THETAf2; // var di comodo per la funzione di meridiana
                if (THETAf2 < 0.0)
                        THETAf2 = 2*PI + THETAf2;  // così THETAf è sempre positivo
                deltaTheta = (THETAf2 - THETAf1);
                deltaPhi = (PHIf2 - PHIf1);

                // avendo calcolato THETAf2 e PHIf2 nella condizione di riflessione sul bersaglio
                // devo calcolare THETAs e PHIs mediante l'algoritmo computazionale complesso
                //e sfruttare le formule dei triangoli sferici
                // per ricavare THETAb e PHIb
                minuti1 = minuti; // salvo info sui minuti prima di aggiornarli
                time (&rawtime);
        timeinfo = localtime (&rawtime);     // salvo data e ora dell'acquisizione
                minuti = timeinfo->tm_min;
                ore    = timeinfo->tm_hour;
                giorno = timeinfo->tm_mday;
                mese   = timeinfo->tm_mon + 1.0; // in questo modo gennaio = 1
                anno   = timeinfo->tm_year + 1900.0; // tm_year dà gli anni che sono passati dal 1900
                if (motore == 'm'){
                        calcola_azimut_zenit(anno, mese, giorno, ore-2.0, minuti, &PHIs, &THETAs);
                 // ora solare: ore-1.0; ora legale: ore-2.0
                        THETAs = THETAs /180.0*PI; //in radianti. ANGOLO AZIMUTALE rispetto al NORD
                        PHIs   = PHIs /180.0*PI; //in radianti. ANGOLO ZENITALE
                }
                if (motore == 'k'){
/****************** ALGORITMO COMPLESSO *********/
                        calcola_azimut_zenit(anno, mese, giorno, ore-2.0, minuti, &PHIs, &THETAs);
                 // ora solare: ore-1.0; ora legale: ore-2.0
                        THETAs = THETAs /180.0*PI; //in radianti. ANGOLO AZIMUTALE rispetto al NORD
                        PHIs   = PHIs /180.0*PI; //in radianti. ANGOLO ZENITALE
```

```
/*************** fino a qui ho THETAf2 PHIf2 THETAs PHIs ******************************/
                                if (num_acquisizioniK == 0){
  // calcolo THETAb e PHIb soltanto al primo ciclo perchè
  // la posizione del bersaglio rimane costante
                                        algoritmo_di_setup (PHIf2, THETAf2, PHIs, THETAs, &PHIb ,&THETAb, &PhiFV);
                                        cout << "THETAs = "       << THETAs  *180/PI  << "\n";
                                        cout << "PHIs = "         << PHIs    *180/PI  << "\n";
                                        cout << "THETAf2 = "      << THETAf2 *180/PI  << "\n";
                                        cout << "PHIf2 = "        << PHIf2   *180/PI  << "\n";
                                        cout << "PhiFV = "              << PhiFV   *180/PI
<< "\n";
                                        cout << "PHIb = "         << PHIb    *180/PI  << "\n";
                                        cout << "THETAb = "       << THETAb  *180/PI  << "\n\n";
                                } // fine (if num_acquisizioni == 0)

/******* QUESTA PARTE VIENE SVOLTA SEMPRE (ALGORITMO OPERATIVO) *******/
                                //SFRUTTO LE FORMULE DEI TRIANGOLI SFERICI per ricavare
                // gli angoli da raggiungere con lo specchio
                                algoritmo_operativo(PHIb, THETAb, PHIs, THETAs, &PHIf_OK ,&THETAf_OK, &PhiFV);
                                if (num_acquisizioniK == 0){
 // dopo il primo ciclo thetaf_ok e phif_ok  vengono cmq stampati a schermo
// (vedi codice più sotto)
                                        cout << "THETAf_OK = " << THETAf_OK  * 180/PI << "\n";
                                        cout << "PHIf_OK = "   << PHIf_OK    * 180/PI << "\n\n";
                                }
                                Dy =  h * tan(PHIf_OK) * cos(THETAf_OK);
//il meno non ci va perchè THETAf_OK è quello del baricentro delle tensioni!
                                Dx =  tan(THETAf_OK) * Dy;
                        }       // fine if (motore == 'k')
                        if (motore == 'c'){
                    /********** ALGORITMO COMPLESSO -> ricavo THETAs e PHIs ***********/
                                calcola_azimut_zenit(anno, mese, giorno, ore-2.0, minuti, &PHIs, &THETAs);
 // ora solare: ore-1.0; ora legale: ore-2.0
                                THETAs = THETAs /180.0*PI; //in radianti. ANGOLO AZIMUTALE rispetto al NORD
                                PHIs   = PHIs /180.0*PI; //in radianti. ANGOLO ZENITALE
                        /***************fino a qui ho THETAf2 PHIf2 THETAs PHIs***************/
                                if (num_acquisizioni == 0){
  // calcolo THETAb e PHIb soltanto al primo ciclo perchè
// la posizione del bersaglio rimane costante

                                        algoritmo_di_setup (PHIf2, THETAf2, PHIs, THETAs, &PHIb ,&THETAb, &PhiFV);

                                        cout << "THETAs = "       << THETAs  *180/PI  << "\n";
                                        cout << "PHIs = "         << PHIs    *180/PI  << "\n";
                                        cout << "THETAf2 = "      << THETAf2 *180/PI  << "\n";
                                        cout << "PHIf2 = "        << PHIf2   *180/PI  << "\n";
                                        cout << "PhiFV = "      << PhiFV   *180/PI  << "\n";
                                        cout << "PHIb = "         << PHIb    *180/PI  << "\n";
                                        cout << "THETAb = "       << THETAb  *180/PI  << "\n\n";

                                } // fine (if num_acquisizioni == 0)
                                /******* QUESTA PARTE VIENE SVOLTA SEMPRE (ALGORITMO OPERATIVO) *******/
                                //SFRUTTO LE FORMULE DEI TRIANGOLI SFERICI per ricavare
                //gli angoli da raggiungere con lo specchio
                                algoritmo_operativo(PHIb, THETAb, PHIs, THETAs, &PHIf_OK ,&THETAf_OK, &PhiFV);

                                if (num_acquisizioni == 0){
  // dopo il primo ciclo thetaf_ok e phif_ok  vengono cmq stampati
// a schermo (vedi codice più sotto)
                                        cout << "THETAf_OK = " << THETAf_OK  * 180/PI << "\n";
                                        cout << "PHIf_OK = "   << PHIf_OK    * 180/PI << "\n\n";
                                }
```

```
    }       // fine if (motore == 'c')
    /**************** VALORI OBIETTIVO DI DELTAX E DELTAY ****************/
if (motore == 'x' || motore == 'c') {

    // h * tanPHI / sqrt(tanTHETA*tanTHETA+1.0));
            Dy =  h * tan(PHIf_OK) * cos(THETAf_OK);
 Dx =  tan(THETAf_OK) * Dy;

            ValInteroDx = (int) (fabs(Dx) * 14000.0 + 0.5);
     ValInteroDy = (int) (fabs(Dy) * 16000.0 + 0.5);
            /*************** INVIO FABS(DX) E IL SUO SEGNO ***************/
            datoTx1 = ValInteroDx % 256;   //il deltaTHETA se lo calcola il PIC da solo
            datoTx2 = ValInteroDx >> 8;
       if (Dx >= 0.0)
                txchar = 0; // invio info sul segno di Dx
            else
                txchar = 1;// Dx negativo
            /**primi 8 bit**/
            ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
            ftstatus1 = FT_Write(fthandle1, &datoTx1, 1, &byteinviati);
            if (!FT_SUCCESS(ftstatus1))
                cout << "\nErrore di scrittura\n";
            ftstatus1 = FT_Read(fthandle1, &rxchar, 1, &bytericevuti);
            DxPIC = (float) (rxchar/14000.0);
            /**secondi 8 bit**/
            ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
            ftstatus1 = FT_Write(fthandle1, &datoTx2, 1, &byteinviati);
            if (!FT_SUCCESS(ftstatus1))
                cout << "\nErrore di scrittura\n";
            ftstatus1 = FT_Read(fthandle1, &rxchar, 1, &bytericevuti);
            DxPIC += (float) (rxchar/14000.0*256);
            /**segno**/
            ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
            ftstatus1 = FT_Write(fthandle1, &txchar, 1, &byteinviati);
            if (!FT_SUCCESS(ftstatus1))
                cout << "\nErrore di scrittura\n";
            ftstatus1 = FT_Read(fthandle1, &rxchar, 1, &bytericevuti);
            cout << "\nsegno = " << (int)rxchar << "\n";
            if ((int)rxchar == 1)
                DxPIC = -DxPIC;
            cout << "DxPIC = "<< DxPIC << "("<< Dx << ")\n";
            /*************** INVIO Dy ***************/
            datoTx1 = ValInteroDy % 256;   //il deltaTHETA se lo calcola il PIC da solo
            datoTx2 = ValInteroDy >> 8;
            if (Dy >= 0.0)
                txchar = 0; // invio info sul segno di Dx
            else
                txchar = 1;// Dx negativo

            /**primi 8 bit**/
            ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
            ftstatus1 = FT_Write(fthandle1, &datoTx1, 1, &byteinviati);
            if (!FT_SUCCESS(ftstatus1))
                cout << "\nErrore di scrittura\n";
            /**secondi 8 bit**/
            ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
            ftstatus1 = FT_Write(fthandle1, &datoTx2, 1, &byteinviati);
            if (!FT_SUCCESS(ftstatus1))
                cout << "\nErrore di scrittura\n";
            /**segno**/
            ftstatus1 = FT_Purge(fthandle1, FT_PURGE_RX | FT_PURGE_TX);
            ftstatus1 = FT_Write(fthandle1, &txchar, 1, &byteinviati);
```

```
                        if (!FT_SUCCESS(ftstatus1))
                                cout << "\nErrore di scrittura\n";

                }  // fine if (motore == 'x' || motore == 'c')
                /*********************************************************************/

                if (fabs (Dx) < 1.0 )
        erroreaccettabileDx = 0.005;
                else if (fabs (Dx) < 2.5)
                        erroreaccettabileDx = 0.005 * fabs(Dx);
                else
                        erroreaccettabileDx = 0.01 * fabs(Dx);


                if (fabs (Dy) < 1.0 )
                        erroreaccettabileDy = 0.005;
                else if (fabs (Dy) < 2.5)
                        erroreaccettabileDy = 0.005 * fabs(Dy);
                else
                        erroreaccettabileDy = 0.01 * fabs(Dy);

                erroreDeltaX = fabs(DeltaX - Dx);  //mi interessa il modulo
                erroreDeltaY = fabs(DeltaY - Dy);  //mi interessa il modulo
                /******************STAMPA RISULTATI A SCHERMO******************/

                cout << "\n\n\n\n";
                cout << "DATA: " << asctime (timeinfo)  << "\n";
                printf("ACQUISIZIONE %d    \n", num_acquisizioni);

                for (int i=0; i<4; i++){
                        for(int s=0; s<16; s++)
                                rapBin[s] = 0;          // inizializzo a 0 la rappr binaria del campione
                        binario(rapBin, rxbuff[i]);  //conversione da decimale(rxbuff) a binario(rapBin)
                        tensione[i] = (double)rxbuff[i]/valoreMAX*Vref;
                        printf("Campione %d: %4d   Binario: ", (i), rxbuff[i]);
                        for(int s=15; s>=0; s--)
                                cout << (int)rapBin[s];
                        printf("   Tensione[V]: %10f\n", tensione[i]);
                }
                printf("\nSomma: %8f\n", somma);
                printf("\nDeltaY[mm]: %8f  (-->%8f)  DeltaX[mm]: %8f  (-->%8f)
                           D[mm]: %8f\n", DeltaY, Dy, DeltaX, Dx, D);
                if (motore == 'c' || motore == 'x')
                        printf("\nerroreDeltaY[mm]:%8f(-->%8f) erroreDeltaX[mm]:%8f(-->%8f)\n",
erroreDeltaY, erroreaccettabileDy, erroreDeltaX, erroreaccettabileDx);
                if (motore == 'm')
                        printf("\nTHETAf+180: %8f (-->(THETAs) = %8f)   PHIf: %8f (-->%8f) \n",
THETAf3*180.0/PI+180.0, THETAs*180.0/PI, PHIf2*180.0/PI, PHIs*180.0/PI);
                else if (motore == 'v' || motore == 'z' || motore == 'x')
                        printf("\nTHETAf: %8f (-->%8f)  PHIf: %8f (-->%8f)  \n",
THETAf2*180.0/PI, THETAf_OK*180.0/PI, PHIf2*180.0/PI, PHIf_OK*180.0/PI);
                else {  // motore = c e motore = k
                        printf("\nTHETAf: %8f (-->(THETAf_OK) = %8f) PHIf:%8f(-->%8f)\n",
THETAf2*180.0/PI,THETAf_OK*180.0/PI, PHIf2*180.0/PI, PHIf_OK*180.0/PI);
                        printf("\nTHETAs: %8f  PHIs: %8f\n", THETAs*180.0/PI, PHIs*180.0/PI);
                        printf("\nTHETAb: %8f PHIb: %8f\n", THETAb*180.0/PI, PHIb*180.0/PI);
                }

                if (motore == 'x' || motore == 'c') {
                        erroreTHETAf2 = fabs (THETAf2 - THETAf_OK) *180.0/PI;
                        errorePHIf2 = fabs (PHIf2 - PHIf_OK) *180.0/PI;
                        printf("\nerroreTHETAf2: %8f    errorePHIf2: %8f\n",
```

```
erroreTHETAf2, errorePHIf2);
                        }
                        if (motore == 'c' && numDevs == 2){
                                // invio richiesta di invio campioni dai FD sul bersaglio
                        richiesta:
                                ftstatus2 = FT_Purge(fthandle2, FT_PURGE_RX | FT_PURGE_TX);
                                ftstatus2 = FT_Write(fthandle2, &codice, 1, &byteinviati);
                                ftstatus2 = FT_Read(fthandle2, &rxchar, 1, &bytericevuti);
                                if (FT_SUCCESS(ftstatus2))
                                        if (bytericevuti != 1) goto richiesta;
                                if (!FT_SUCCESS(ftstatus2)) goto richiesta;

                                //ricevo dati dai fotodiodi sul bersaglio
                                for (int i=0; i<4; i++){
                                        ftstatus2 = FT_Purge(fthandle2, FT_PURGE_RX | FT_PURGE_TX);
                                        ftstatus2 = FT_Write(fthandle2, &codice, 1, &byteinviati);
                                        ftstatus2 = FT_Read(fthandle2, &rxchar, 1, &bytericevuti);
                                        rxbuff[i] = rxchar;
                                        ftstatus2 = FT_Purge(fthandle2, FT_PURGE_RX | FT_PURGE_TX);
                                        ftstatus2 = FT_Write(fthandle2, &codice, 1, &byteinviati);
                                        ftstatus2 = FT_Read(fthandle2, &rxchar, 1, &bytericevuti);
                                        rxbuff[i] += rxchar*256;
                                } //fine ciclo for
                                printf("FD1: %-10d FD2: %-10d FD3: %-10d FD4: %-10d\n",
                                                rxbuff[0], rxbuff[1], rxbuff[2], rxbuff[3]);
                        } // fine if (motore == 'c' && numDevs == 2)
                        cout << "\n";
                        num_acquisizioni++;     //incremento l'indice

                        /*****************SALVATAGGIO DATI SU FILE TXT*****************/
                        switch (motore) {
                                case'k':
                                        if((stream = fopen("VERIFICA.TXT", "a")) != NULL){
                                                if(num_acquisizioniK == 0){
                                                        fprintf (stream, "DATA:   %s \n\n\n",
asctime(timeinfo));
                                                        fprintf (stream, "Minuti
THETAs
        PHIs          THETAb          PHIb            THETAf_OK
THETAf          PHIf_OK         PHIf
 Dx              DeltaX          Dy
DeltaY          FD0         FD1             FD2         FD3\n");
                                                }
                                                fprintf (stream, "%-10f      %-10f      %-10f
%-10f
%-10f      %-10f      %-10f      %-10f      %-10f
%-10f      %-10f      %-10f      %-10f      %-5d
%-5d      %-5d      %-5d\n", minuti, THETAs*180.0/PI, PHIs*180.0/PI,
THETAb*180.0/PI, PHIb*180.0/PI, THETAf_OK*180.0/PI,
PHIf_OK*180.0/PI, THETAf2*180.0/PI, PHIf2*180.0/PI,
Dx, DeltaX, Dy, DeltaY, rxbuff[0], rxbuff[1],
rxbuff[2], rxbuff[3]);
                                                fclose (stream);
                                        }
                                        else
                                                printf( "Problema nell'apertura del file\n" );
                                        break;
                                case 'm':
                                        if((stream = fopen("MERIDIANA.TXT", "a")) != NULL){
                                                if(num_acquisizioni == 1){
                                                        fprintf (stream, "DATA:   %s \n\n\n",
                asctime(timeinfo));
```

```
                                                fprintf (stream, "Minuti    THETAs
                                        PHIs                THETAf          PHIf
                                    Dx                  Dy              DeltaX
                                        DeltaY              FD0         FD1
                                                FD2         FD3\n");
                                        }
                                        fprintf (stream, "%-10f      %-10f      %-10f
                            %-10f       %-10f       %-10f       %-10f       %-10f
                            %-10f       %-5d        %-5d        %-5d        %-5d\n",
                                minuti, THETAs*180.0/PI, PHIs*180.0/PI,
                                (THETAf3*180.0/PI)+180.0, PHIf2*180.0/PI, Dx, Dy,
                                DeltaX, DeltaY, rxbuff[0], rxbuff[1],
                                rxbuff[2], rxbuff[3]);
                                        fclose (stream);
                                }
                                else
                                        printf( "Problema nell'apertura del file\n" );
                                break;
                            case 'c':
                                if((stream = fopen("CONCENTRAZIONE.TXT", "a")) != NULL){
                            if(num_acquisizioni == 1){
                                        fprintf (stream, "DATA:    %s \n\n\n",
                            asctime(timeinfo));
                                        fprintf (stream, "Minuti       THETAs
                        PHIs            THETAb          PHIb            THETAf_OK
                                PHIf_OK         Dx              DeltaX              Dy
                    DeltaY          FD0             FD1         FD2         FD3\n");
                                        }
                                        else if (minuti1 != minuti){
  // salvo su file soltanto una volta al minuto
//(tanto THETAs e PHIs variano una volta al minuto)
                                        fprintf (stream, "%-10f      %-10f
                            %-10f       %-10f       %-10f       %-10f       %-10f
                            %-10f       %-10f       %-10f       %-10f       %-5d
                                %-5d        %-5d        %-5d\n",
                                minuti, THETAs*180.0/PI, PHIs*180.0/PI,
                                THETAb*180.0/PI, PHIb*180.0/PI, THETAf_OK*180.0/PI,
                                PHIf_OK*180.0/PI, Dx, DeltaX, Dy, DeltaY,
                                rxbuff[0], rxbuff[1], rxbuff[2], rxbuff[3]);
                                        fclose (stream);
                                }
                                }
                                else
                                        printf( "Problema nell'apertura del file\n" );
                                break;
                            case 'x':
                                if((stream = fopen("CONTROLLO.TXT", "a")) != NULL){
                            if(num_acquisizioni == 1){
                                        fprintf (stream, "DATA:    %s \n\n\n",
                        asctime(timeinfo));
                                        fprintf (stream, "Minuti
                    THETAf_OK       PHIf_OK             THETAf          PHIf
                ErroreTheta    ErrorePhi       ErroreDeltaX    ErroreDeltaY\n");
                                        }
                                        fprintf (stream, "%-10f      %-10f      %-10f
                        %-10f       %-10f       %-10f       %-10f       %-10f       %-10f\n",
                                minuti, THETAf_OK*180.0/PI,PHIf_OK*180.0/PI,
                                THETAf2*180.0/PI,PHIf2*180.0/PI, erroreTHETAf2,
                                errorePHIf2, erroreDeltaX, erroreDeltaY);
                                        fclose (stream);
                                }
                                else
```

```
                                        printf( "Problema nell'apertura del file\n" );
                        }
                        if (motore == 'c' || motore == 'x')
                                goto acquisizioni;
                        else if (motore == 'm'){
                                for (int z = 1; z>0; z--){ // ritardo di 1 minuto
                                        cout << z << "  " << flush;
                                        sleep(1);
                                }
                                goto acquisizioni;
                        }

                } // fine ciclo for (int num_passi = 0; num_passi < passi; num_passi++)
                if (motore == 'k')
                        num_acquisizioniK++;
                num_acquisizioni = 0;
                passi = 1;
        } // fine while (1)
return 0;
```

Bibliografia

[1] Roberto Grena, *An algorithm for the computation of the solar position.* Elsevier, 2007

[2] CPV Consortium, *CONVEGNO INTERNAZIONALE - FOTOVOLTAICO A CONCENTRAZIONE - il dispiegamento sul mercato di una tecnologia avanzata a elevata producibilità.* Solar Expo, 4 Maggio 2011, Verona

[3] A. Prest, H Grassmann, *The linear mirror for solar energy exploitation.* Il Nuovo Cimento, 2009

[4] M. Falchetta, *Il programma ENEA sull'energia solare a concentrazione ad alta temperatura.* ENEA Centro Ricerche, 2006

[5] Nancy Hartsoch, *CPV Industry Overview.* CPV Consortium, 2011

[6] A. Rifatto, *Lezioni ed esercizi di astronomia,* www.na.astro.it/~rifatto/personale/dispense_di_astronomia.pdf. 2003

[7] Sid Katzen, *The Essential PIC18® Microcontroller.* Springer, 2010

[8] Mauro Laurenti, *C18 Step By Step.* 2009

[9] Microchip, *MPLAB® C18 C COMPILER LIBRARIES.* 2005

[10] Microchip, *PIC18F2331/2431/4331/4431 Data Sheet.* 2010

[11] Microchip, *PIC18 CONFIGURATION SETTINGS ADDENDUM.* 2006

[12] FTDI, *UM232R USB - Serial UART Development Module Datasheet*, 2011

[13] Como Drills, *Low Voltage D.C. Motors & Gearbox Units. 2012*

[14] *Impianto Gemasolar*, http://www.focus.it/scienza/energia/03112011-1023-478-il-sole-anche-di-notte_C38.aspx, 2012

[15] *Efficienza di conversione*, *http://www.comisaenergy.it/ITA/Efficienza.asp?area=1&percorso=11*, 2012

[16] Franco Martinelli, *La sfera celeste,* http://www.nauticoartiglio.lu.it/almanacco/quaderni/www/sfera/WITN_sfera_2.htm, 2012

[17] Mauro Bertolini, *Mini corso di trigonometria sferica*, http://www.nauticoartiglio.lu.it/didattica/trigsfer/trigsferica.htm, 2012

[18] *La Terra e la luna.* http://www.icsbaunei.nu.it/scienze%20ipertesto/nuova_pagina_11.htm, 2012