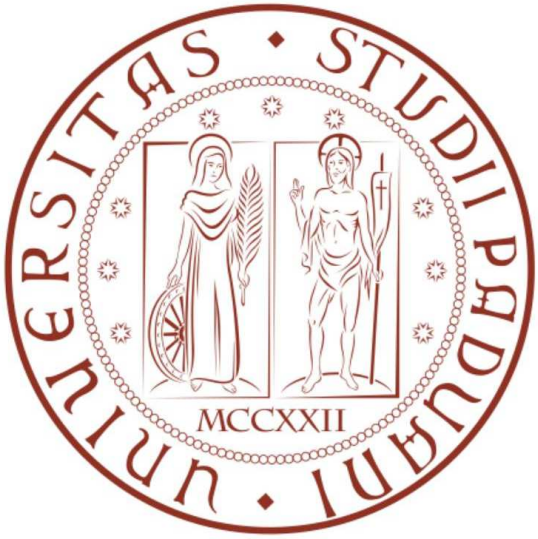



Università degli studi di Padova
Facoltà di Ingegneria
Dipartimento di Tecnica e Gestione dei Sistemi Industriali

Corso di Laurea Triennale in Ingegneria Meccanica e Meccatronica
Curriculum Meccatronico

Tesi di Laurea

# REALIZZAZIONE DI UN SISTEMA 3D DI POSIZIONAMENTO DI UN SENSORE PER LA MAPPATURA DI CAMPI MAGNETICI

Relatore: Prof. Giuseppe Chitarin

Laureando: Gabriele Volpato
Matricola: 1075443

Anno Accademico 2016/2017

# INDICE

# INTRODUZIONE

L' obiettivo di questa tesi è quello di automatizzare una struttura a tre assi, già esistente, per la misurazione del campo magnetico lungo una griglia composta da magneti permanenti.
La struttura inizialmente veniva messa in movimento tramite manovelle, quindi manualmente.
La seguente tesi descrive la stessa struttura con l'inserimento di tre motori, uno per ogni asse.
I tre motori pilotati da un microcontrollore, Arduino Due, muovono la struttura in due modalità:
1) manuale – tramite un joystick e una manopolina che comandano singolarmente ogni motore;
2) automatica – tramite una sequenza di movimenti prestabiliti, i motori si mettono in movimento.

I motori utilizzati sono di tipo passo passo, poiché si prestano molto bene al controllo della posizione e sono molto accurati negli spostamenti programmati.

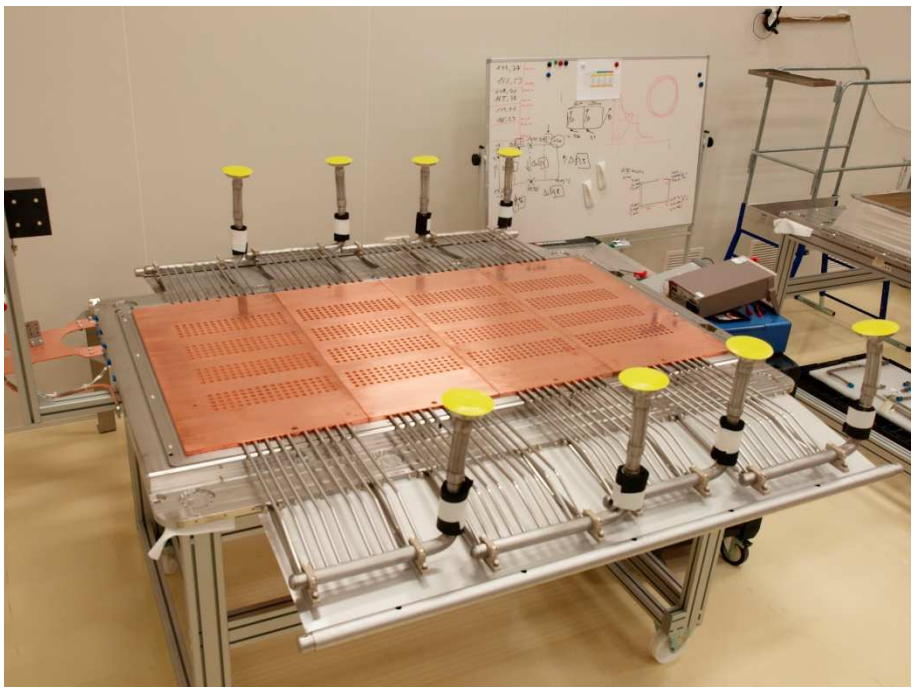
Per le misure magnetiche viene utilizzato un Gaussmetro esterno dotato di una sonda che viene messa in movimento appunto dalla struttura.

Per la determinazione del percorso da far effettuare al sensore è stata creata un'interfaccia apposita che schematizza la griglia in due dimensioni. In questo modo si semplifica la selezione dei punti in cui si è interessati ad effettuare la misurazione, cliccando un punto della griglia con il mouse.

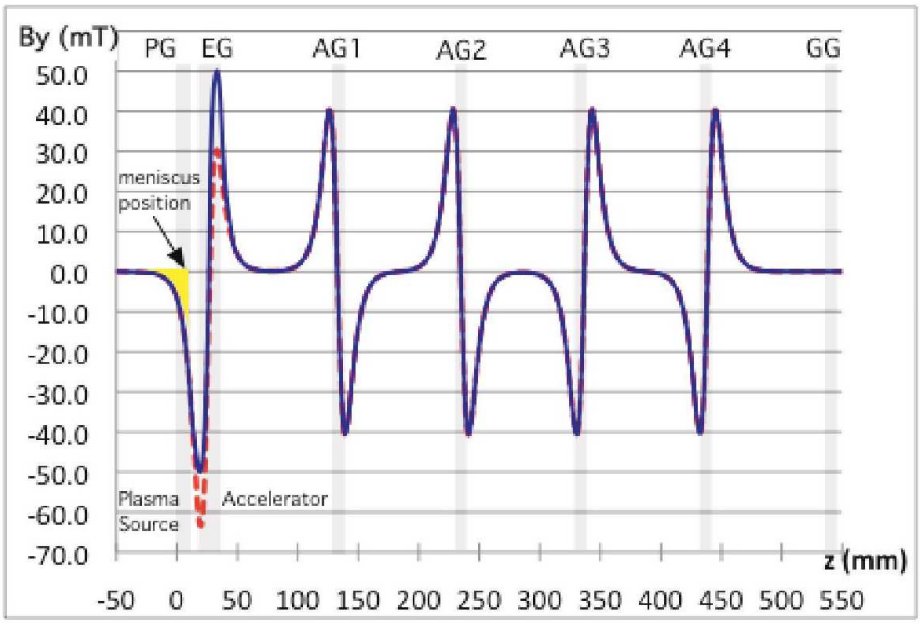
In conclusione, al fine di migliorare il sistema progettato, vengono avanzate delle proposte pensate durante la realizzazione di quest'ultimo.

# CAPITOLO 1: Misure magnetiche

La struttura è stata progettata e realizzata con lo scopo di effettuare delle misurazioni magnetiche accurate in vari punti di una griglia forata, contenente magneti permanenti.



La griglia è utilizzata in un centro di ricerca ed è un componente di un acceleratore di ioni negativi. Tralasciando l'utilizzo specifico delle griglie all'interno dell'acceleratore, il campo magnetico generato da esse deve soddisfare un certo andamento che è possibile notare in figura[1].



Per il corretto funzionamento dell'acceleratore di ioni negativi, dunque, è richiesta una grande attenzione sia in fase di costruzione delle griglie, sia in fase di misurazione del campo magnetico. Quest'ultima fase funge da verifica per l'intero sistema.

Altro aspetto molto importante per la verifica è la scelta dello strumento di misurazione. Infatti, in fase preliminare, onde evitare un mal funzionamento nell'acquisizione dei dati, si sono tenuti separati i sistemi di movimentazione del sensore e delle misurazioni. La scelta dello strumento esterno, quindi, ha favorito la semplificazione nella progettazione del sistema di posizionamento del sensore. Lo strumento utilizzato per le misurazioni magnetiche è un Gaussmetro dotato di una sonda agganciata all'asse Z della struttura. Il Gaussmetro nello specifico è il Brockhaus/Lake Shore 425. Essendo esterno alla struttura è dunque dotato di un proprio programma per l'acquisizione dati e non sarà oggetto della seguente tesi.

# CAPITOLO 2: Descrizione della struttura a tre assi per il posizionamento del sensore e requisiti del sistema di controllo dei motori

La struttura utilizzata per questa applicazione è stata acquistata dalla RK- Rose+Krieger[2].

È composta da tre guide, una per ogni asse direzionale.

Ogni guida ha in uno degli estremi un motore a passo collegato tramite un giunto elastico alla vite senza fine interna alla struttura.

Quest'ultima permette il movimento lineare del carrellino posto sopra la guida. La vite senza fine presenta un passo di 1 mm, ciò significa che ad ogni giro si effettua uno spostamento lineare di un mm.

Le guide sono costituite da una lega di alluminio con sezione quadrata 20x20 mm per l'asse Z e sezione rettangolare 40x20 mm per gli altri due assi. Tutte e tre presentano lunghezze differenti, infatti:

- X: 500 mm
- Y: 400 mm
- Z: 300 mm

Solamente la guida lungo l'asse X è fissata alla struttura della griglia su cui effettuare le misure magnetiche, mentre la guida direzionata lungo l'asse Y è posta sopra il carrellino della guida precedente in modo tale da permettere uno spostamento nelle due direzioni. L' ultima guida naturalmente è fissata al carrellino dell'asse Y.

Ad ogni estremità delle guide sono stati installati degli switch fine corsa che comunicano ad Arduino che il carrellino è arrivato alla fine della guida bloccando il movimento del motore per proseguire, ma consentendo di tornare indietro.

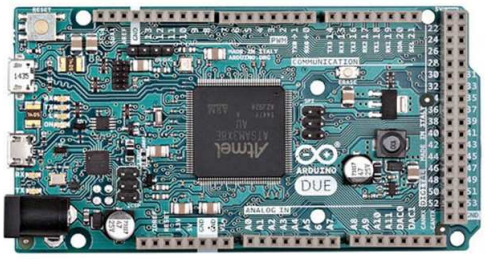
# CAPITOLO 3: Componenti del circuito hardware del sistema di controllo dei motori

## 3.1 Elenco componenti

La parte software è composta da vari elementi, quali:

- Arduino Due;
- 3 driver per il controllo dei 3 motori passo passo;
- 3 motori passo passo bipolari;
- 4 pulsanti;
- 1 potenziometro;
- 1 schermo lcd 16x2;
- 1 joystick;
- 1 rotary encoder;
- 6 switch fine corsa;
- 1 alimentatore esterno per motori (12v per 3A);
- 1 alimentatore esterno per Arduino Due (5v);

1) ARDUINO DUE



La scheda Arduino Due è basata sul microcontrollore Atmel SAM3X8E ARm Cortex-M3.

Arduino Due possiede 54 pin I/O digitali, di cui 12 utilizzati come uscite PWM, 12 pin analogici, 4 UARTS (porte seriali hardware), un clock a 84 MHz, 2 DAC (digital/analog), 2 TWI (per la comunicazione I2C o Two Wire), jack di alimentazione da 2,1 mm con positivo centrale, una JTAG per la programmazione diretta del microcontrollore e per il debug, un bottone di reset ed uno di cancellazione.

Arduino Due risulta come la prima scheda Arduino a 32 bit e con una tensione di funzionamento a 3,3V: questo presuppone una maggiore accortezza nella scelta dei segnali di input e output, in quanto una tensione superiore potrebbe risultare dannosa per l'intera scheda.

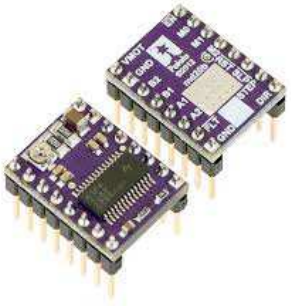
L'alimentazione viene fornita direttamente dalla porta USB oppure mediante un alimentatore esterno.

Arduino Due è dotata di due porte USB: una, denominata “Programming port”, dedicata sia all’alimentazione che alla sua programmazione tramite l’IDE di Arduino, mentre una seconda, denominata “Native port”, che permette di collegare e utilizzare la scheda come una qualunque periferica USB. Quest’ultima porta può essere usata anche come porta seriale virtuale utilizzando l’oggetto “SerialUSB” nel linguaggio di programmazione di Arduino.

Caratteristiche tecniche:

- Microcontroller: AT91SAM3X8E
- Tensione di funzionamento: 3.3V
- Tensione di ingresso (raccomandata): 7-12V
- Tensione di ingresso (limiti): 6-20V
- Pin I/O digitali: 54 (di cui 12 utilizzati come uscite PWM)
- Pin di ingresso analogico: 12
- Uscite analogiche: 2 (DAC)
- Corrente totale sulle linee I/O: 130 mA
- Corrente sul Pin 3,3V: 800 mA
- Corrente sul Pin 5V: 800 mA
- Memoria Flash: 512 KB, totalmente disponibili per le applicazioni dell’utente
- SRAM: 96 KB (due banchi: 64 KB e 32 KB)
- Velocità di clock: 84 MHz

2) DRIVER PER IL CONTROLLO MOTORE PASSO PASSO



Il driver progettato da Pololu è composto da 16 pin, di cui 12 ingressi e 4 uscite. Quest’ultime quattro sono coloro che comunicano direttamente con il motore passo passo.

Ogni fase del motore passo passo bipolare è collegata ad un’uscita del driver tramite cui riceve l’alimentazione in base al funzionamento richiesto. Per esempio nella modalità FULL-STEP verrà alimentata una fase per volta in una sequenza opportuna per far muovere il motore in un senso o nell’altro. Nel caso in cui si vorrà sfruttare il microstepping e quindi utilizzare un avanzamento corrispondente a mezzo step o un quarto di step si alimenteranno due fasi contemporaneamente per poter raggiungere la posizione desiderata.

I 16 ingressi sono composti da:

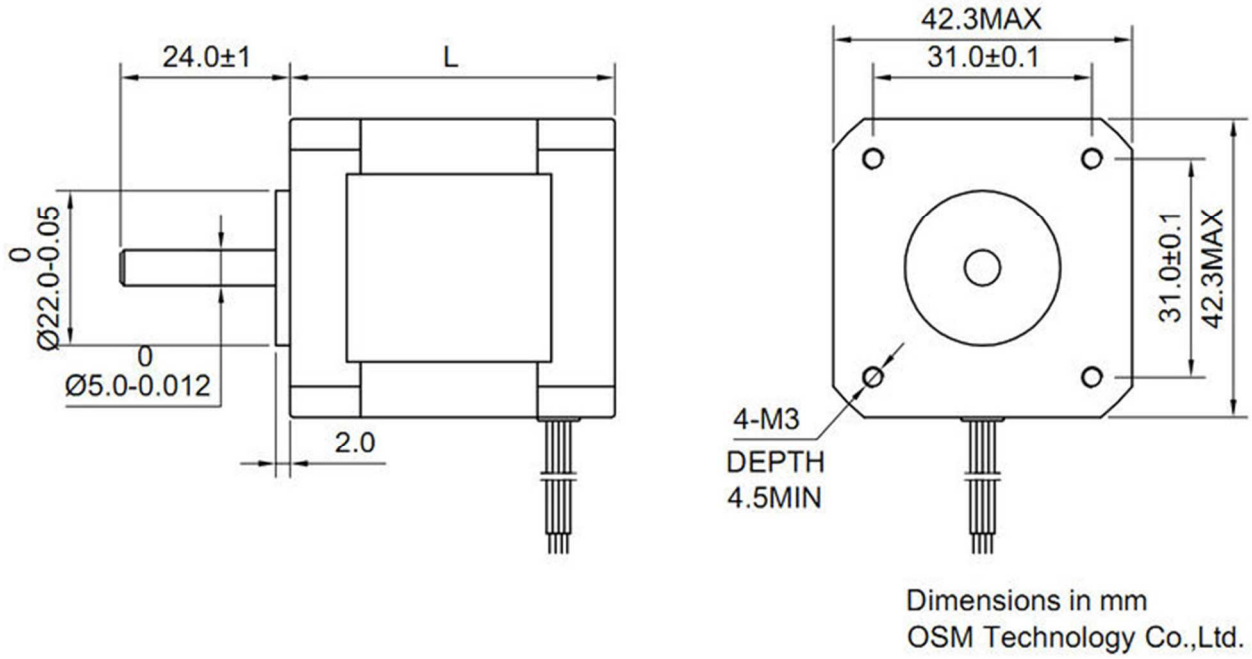
- 2 ingressi collegati all'alimentatore esterno da 12v per 3A tramite cui si alimenta il motore limitando in ingresso, tramite il limitatore di corrente, la corrente d'ingresso ad ogni fase fino ad un ampere.
- 1 ingresso collegato al GND della scheda Arduino
- 2 ingressi collegati all'alimentatore di Arduino ovvero collegati al 3.3v, che nello specifico sono gli ingressi RESET e SLEEP non utilizzati nel codice
- 1 ingresso denominato STEP, tramite il quale Arduino comunica al driver un'onda quadra che impone la cadenza e quindi la frequenza degli step.
- 1 ingresso denominato DIR, che accetta i valori logici ALTO e BASSO, tramite il quale si comunica il verso di rotazione che si vuole imporre al motore
- 3 ingressi denominati M0, M1 e M2 attraverso i quali si comunica la modalità di avanzamento del motore, ovvero se si vuole utilizzare microstepping o no in base alla sequenza di questi 3 parametri come si può vedere dalla tabella n° 1 qui sotto
- 1 ingresso denominato ENABLE che accetta i valori logici ALTO e BASSO, tramite i quali si comunica da Arduino se si vuole fermare e mettere a riposo i motori o azionarli. Con il valore logico ALTO si blocca il motore e con BASSO si permette il suo funzionamento
- 1 ingresso denominato FAULT lasciato scollegato perché non utilizzato dal codice

| MODE0 | MODE1 | MODE2 | Microstep Resolution |
|---|---|---|---|
| Low | Low | Low | Full step |
| High | Low | Low | Half step |
| Low | High | Low | 1/4 step |
| High | High | Low | 1/8 step |
| Low | Low | High | 1/16 step |
| High | Low | High | 1/32 step |
| Low | High | High | 1/32 step |
| High | High | High | 1/32 step |

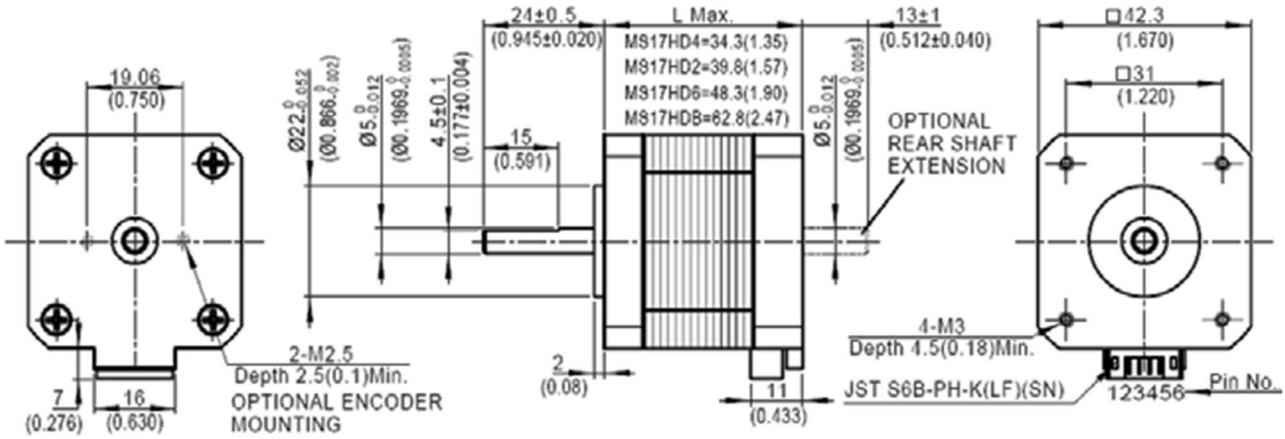
*Tabella n° 1*

3) MOTORE PASSO PASSO

I motori sono tutti e 3 uguali e seguono lo standard dimensionale NEMA17, ovvero ha dimensioni esterne 1.7 x 1.7 inch (43.2 x 43.2 mm).




I motori utilizzati sono siglati 17HD34008-22B.




SPECIFICHE TECNICHE:

Condizioni di esercizio: Ambiente Temperatura: -20 ~ 50;

RH: 90% MAX;

Posizione di montaggio: Asse installazione orizzontale o verticale

Resistenza degli avvolgimenti in corrente continua (25): 2.3|10%

induttanza degli avvolgimenti: 3mH 20%

Cogging: 12mN.m REF.

Coppia di mantenimento: 300mN.m i = 1,5

Frequenza massima di avviamento a vuoto: 1500pps

Frequenza massima di esercizio a vuoto : 8000pps

Aumento di temperatura: <80K

tolleranza Passo angolo: 1.8 gardi ± 5%

Momento di Inerzia rotante: 38g.cm2

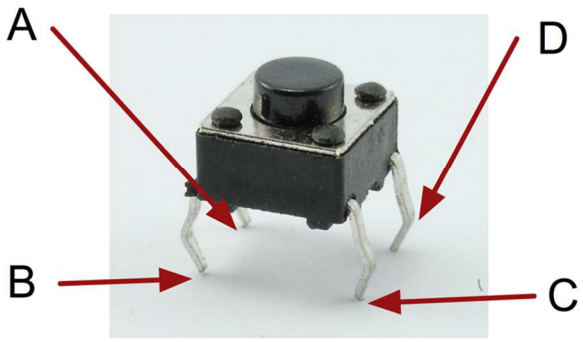
Peso Motore : 0.23kg / PC RIF.

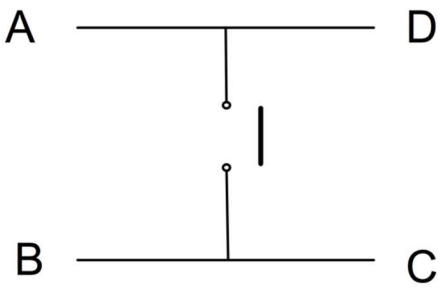
Resistenza di isolamento: resistenza di isolamento freddo dovrebbe essere piu di 100 Mega Ω (tra il nucleo dello statore del motore e terminali)

Rigidità dielettrica: Lo spazio tra il nucleo dello statore del motore e terminale deve essere in grado di sopportare una tensione AC di 600V per 1s senza rompersi.

La corrente di dispersione minore di 1 microA.

4) PULSANTI A BOTTONE




Sono stati utilizzati dei semplici pulsanti a bottone per comunicare ad Arduino alcune informazioni, come per esempio la posizione dell'origine degli assi di riferimento o l'avvio del programma automatico da svolgere per le misure magnetiche.

Come si vede dall'immagine soprastante ogni pulsante è dotato di 4 pin: A, B, C e D.

Sono in comunicazione a due a due se il pulsante è su, viceversa sono tutti in comunicazione se il pulsante viene premuto.

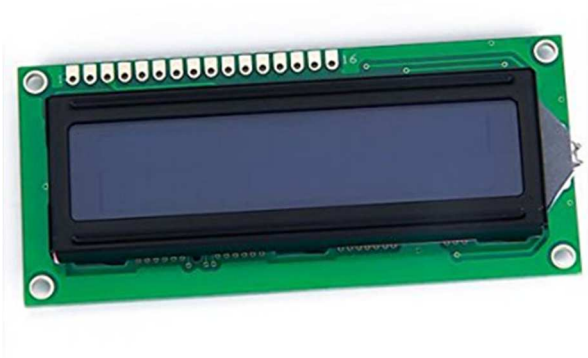
Nel circuito si è collegato il pin B ad un alimentatore di Arduino a 3.3v e il pin A al GND e ad un pin di INPUT di Arduino. Attraverso quest'ultimo il microcontrollore legge lo stato del pulsante, infatti se in INPUT si avrà un valore logico basso, significa che il pulsante non è stato premuto ed è collegato al GND, mentre con un valore logico alto (a 3.3v), significa che si è messo in collegamento i due pin A e B e dunque il pulsante è stato premuto.

5) POTENZIOMETRO E SCHERMO LCD

Il potenziometro è una resistenza regolabile tramite la manopolina in testa. È collegato tra Arduino e lo schermo lcd con la funzione di regolare il contrasto delle scritte con lo sfondo dello schermo.

Lo schermo LCD in figura è uguale a quello utilizzato nel progetto di tesi.

Esso presenta una schermata di sfondo blu, con caratteri bianchi.



Lo schermo è composto da 16 colonne e 2 righe per un totale di 32 caratteri.

Si può comunicare con Arduino tramite un'apposita libreria (LiquidCrystal Library) composta da funzioni utili per scrivere e non solo.

Lo schermo viene collegato ad Arduino tramite 12 collegamenti, di cui 6 comunicano con i pin di INPUT e i restanti 6 vengono collegati all'alimentazione e GND.

Nel progetto lo schermo LCD viene utilizzato per comunicare la posizione in mm delle tre coordinate in cui si trova la sonda.

6) JOYSTICK

Il joystick è un sensore tipo encoder che si collega ad Arduino come INPUT.

Come si vede dall'immagine qui a sinistra il joystick possiede 5 pin:

- GND: collegamento al ground di Arduino
- +5V: collegamento all'alimentazione
- VRx: valore lungo l'asse X
- VRy: valore lungo l'asse Y
- SW: lettura dello stato del pulsante sotto la levetta.

I primi due sono i collegamenti che danno il riferimento al segnale di lettura in ingresso alla scheda. Infatti, lungo ogni asse il joystick invia un segnale analogico ad Arduino con un valore che varia da 0 a 1023. Questo valore viene interpretato da Arduino in base ad una scala di voltaggio in ingresso, infatti è come se dividessimo l'intervallo da 0 a +3,3V in 1024 piccoli intervalli e li numerassimo da 0

a 1023. Inoltre, sotto la levetta vi è un pulsante che comunica al microcontrollore tramite il pin SW con un segnale digitale ALTO o BASSO se è premuto oppure no.

7) ROTARY ENCODER


Il rotary encoder ha 5 pin:

- CLK
- DT
- SW
- +
- GND

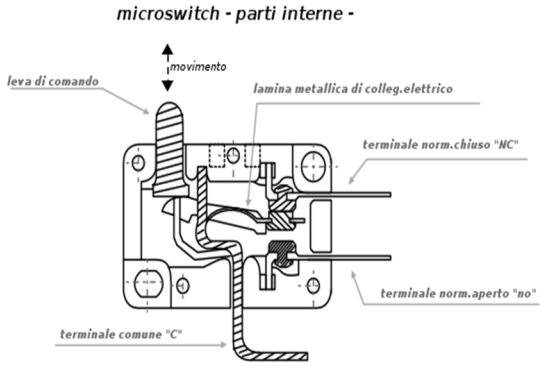
Gli ultimi due pin sono collegati all'alimentazione di Arduino e conferiscono i due estremi su cui lavora il rotary encoder.

Gli altri tre ingressi CLK, DT e SW comunicano ad Arduino lo stato dell'encoder.

Infatti il SW, come per il joystick, comunica se viene premuto o meno il pulsante sottostante la manopolina con un segnale digitale. Mentre gli altri, sempre tramite segnale digitale, comunicano se viene girata la manopolina e in quale senso.

Questa lettura dello stato dell'encoder è possibile grazie alla variazione, o meglio la traslazione, di due onde quadre identiche sfasate di 90°. Nello specifico, avviene che se la manopolina viene fatta girare in senso orario i valori comunicati dai due pin sono uguali. Se invece viene fatta girare in senso antiorario i due valori comunicati sono differenti. In questo modo si può registrare se la manopolina è stata girata in un senso o nell'altro.

8) SWITCH FINECORSA




Lo switch finecorsa ha tre terminali denominati: NC, NO e C.

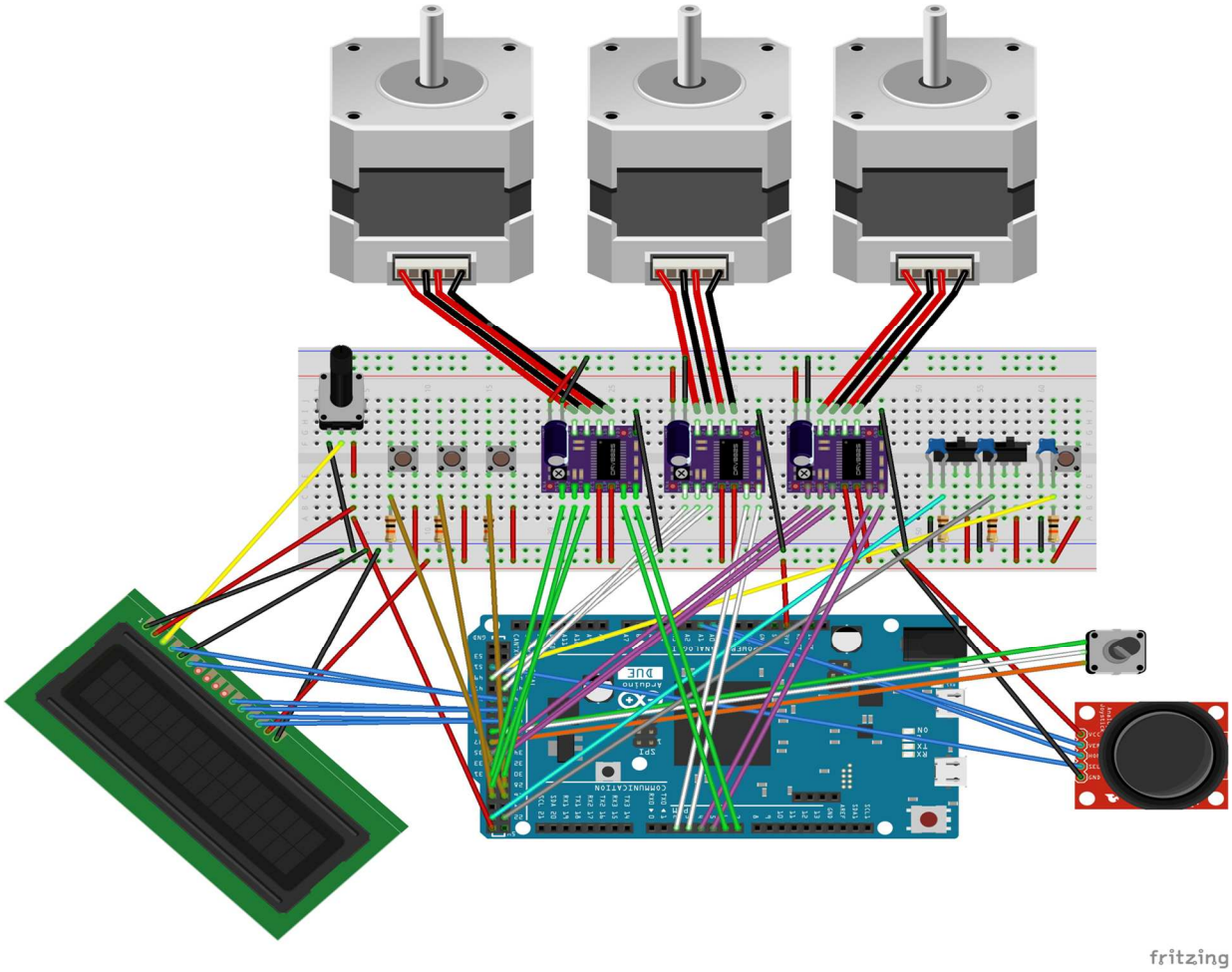
È un interruttore e funziona in modo tale che se premuto mette in comunicazione NO con C, altrimenti collega NC con C.

9) ALIMENTATORE 12v 3A

L’ alimentatore è stato scelto per il voltaggio richiesto dai motori quindi 12v, ma soprattutto per l’amperaggio, infatti ogni motore necessita di 1A per funzionare in modo ottimale.

## 3.2 Realizzazione del circuito elettrico del sistema di controllo dei motori

Il circuito è stato realizzato in fase preliminare con breadboard e cavetti jumper. Si può vedere dallo schema realizzato con Fritzing. La scheda è poi stata realizzata mediante un circuito stampato.




Questo schema vuole riassumere tutti i componenti citati nel sottocapitolo precedente e mostrare come è stato effettuato il cablaggio del sistema completo.

# CAPITOLO 4: Descrizione del codice Arduino per il controllo dei motori

## 4.1 Aspetti generali sul funzionamento del codice

ARDUINO

Programma: MOTORS_CONTROL

Lo scopo del programma è quello di pilotare tre motori a passo tramite appositi driver per poter muovere un sensore di un Gaussmetro nelle tre direzioni dello spazio.

Le misurazioni magnetiche sono svolte al fine di misurare il campo magnetico prodotto da una griglia forata contenente magneti permanenti.

Il sistema si avvale di una struttura composta da tre guide su cui sono montati i motori, ognuno in una slitta direzionata in una delle 3 direzioni spaziali.

Il codice consente di controllare il movimento del sensore secondo un funzionamento manuale, tramite joystick e una manopolina, e un funzionamento automatico, in cui il programma esegue un percorso prestabilito e avviabile premendo un pulsante (buttonProgramPin).

Il sistema necessita di alcune operazioni iniziali (solo la prima è necessaria, le altre due non servono nel caso in cui le guide siano perfettamente parallele alla griglia):

1) assegnazione dell'origine delle tre coordinate tramite pulsante (1);
2) assegnazione al sistema di un secondo punto tramite il pulsante (2), grazie al quale è possibile tarare l'asse X se inclinato rispetto alla griglia su cui vanno effettuate le misurazioni. Inoltre aggiorna anche la posizione del sensore rispetto all'asse Z;
3) assegnazione al sistema di un terzo punto tramite il pulsante (3), grazie al quale è possibile tarare l'asse Y se inclinato rispetto alla griglia su cui vanno effettuate le misurazioni. Inoltre aggiorna anche la posizione del sensore rispetto all'asse Z.

Il programma inoltre controlla se i carrelli di ogni asse arrivano alla fine della guida tramite switch fine corsa, consentendo il movimento per tornare indietro e non per avanzare.

Per quanto riguarda le modalità di funzionamento, quello automatico utilizza una velocità e una frequenza. Infatti i motori sono pilotati con una frequenza di 1.25 kHz e un passo definito in modalità FULL-Step dal driver, che consente di spazzare un angolo di 1.8° con un singolo step. Per una rivoluzione completa il motore effettua, dunque, 200 step.

Il funzionamento manuale, invece, utilizza velocità e frequenza variabili. Questo permette di avere una maggior precisione nei piccoli movimenti e una maggior rapidità nei movimenti più ampi. Grazie

al joystick e alla manopolina si possono imporre principalmente 3 velocità in ogni direzione, sfruttando il microstepping consentito dal driver. Infatti, alla prima velocità avremo un passo molto piccolo, e quindi una maggiore precisione a velocità bassa, per spazzare un angolo giro sono necessari 800 step. Alla seconda velocità, raddoppiamo l'angolo spazzato da ogni singolo step, quindi sono necessari 400 step per un giro completo e avendo la stessa frequenza che pilota la cadenza di ogni step, anche la velocità risulterà raddoppiata. Alla terza ed ultima velocità, il passo anche qui viene raddoppiato e quindi basteranno 200 step per una rivoluzione completa. In questa modalità il motore inizialmente utilizza la stessa frequenza delle due precedenti modalità, ma se si rimane in questa per più tempo la frequenza viene fatta aumentare progressivamente alzando notevolmente la velocità di spostamento del carrellino sulla guida passando gradualmente da una frequenza di 1.25kHz a 1.43kHz.

Il percorso designato e scelto da far compiere al sensore è comunicato ad Arduino tramite un'interfaccia creata con il programma Processing. Da quest'ultima vengono comunicate le coordinate dei vari punti in cui si deve fermare il sensore e vengono salvate in un apposito array che Arduino esegue scorrendo uno a uno. Sebbene il programma Processing possa inviare tramite seriale una stringa e quindi più byte consecutivi, Arduino con la funzione read() riesce a leggere e memorizzare un byte alla volta. Quindi si è dovuto pensare ad una codifica per la comunicazione per non avere problemi di interpretazione dei dati tra i due. Ogni messaggio inviato da Processing e letto da Arduino è composto da 5 caratteri, per esempio "+001\n". Il primo identifica il segno del valore comunicato, a seguire abbiamo un numero composto da 3 cifre, ed infine l'ultimo carattere nuova riga '\n' è stato identificato come la fine del messaggio.

Oltre alla comunicazione da Processing ad Arduino per le posizioni da eseguire lungo il percorso, c'è la comunicazione inversa, in cui Arduino comunica a Processing istante per istante la posizione del sensore. Queste ultime informazioni vengono utilizzate per la rappresentazione grafica sulla griglia e la comunicazione delle coordinate nell'interfaccia di Processing.

## 4.2 Codice

Vedi Appendice A.

## 4.3 Funzioni utilizzate

1) pinMode(nPin, mode)

Funzione già presente nell'IDE di Arduino che serve a impostare nel setup iniziale la modalità mode, ovvero se INPUT o OUTPUT, con cui viene considerato il pin numero nPin.

2) digitalWrite(nPin, value)

Funzione già presente nell'IDE di Arduino che serve ad impostare il valore logico value di output, ovvero HIGH o LOW, del relativo pin numero nPin.

3) digitalRead(nPin, value)

Funzione già presente nell'IDE di Arduino che serve a leggere il valore logico value di input, ovvero HIGH o LOW, del relativo pin numero nPin.

4) analogRead(nPin, value)

Funzione già presente nell'IDE di Arduino che serve a leggere il valore analogico value di input, ovvero un valore interno all'intervallo [0, 1023], del relativo pin numero nPin.

5) delay(nValue)

Funzione già presente nell'IDE di Arduino che serve ad impostare un ritardo di nValue millisecondi, in cui Arduino sospende qualsiasi attività e aspetta il ritardo.

6) delayMicroseconds(nValue)

Funzione già presente nell'IDE di Arduino serve ad impostare un ritardo di nValue microsecondi, in cui Arduino sospende qualsiasi attività e aspetta il ritardo.

7) lcd.begin(column, raw)

Funzione della libreria LiquidCrystal, grazie alla quale si impostano ne setup iniziale le dimensioni dello schermo indicando il numero di righe raw e il numero di colonne column.

8) lcd.setCursor(nRaw, nColumn)

Funzione della libreria LiquidCrystal, attraverso la quale si imposta la casella di riga nRaw e colonna nColumn della matrice del display da cui si inizierà a scrivere.

9) lcd.print(“string”)

Funzione della libreria LiquidCrystal, attraverso la quale si comunica ad Arduino la stringa “string” da scrivere nel display.

10) Serial.begin(baudrate)

Funzione già presente nell’IDE di Arduino che serve all’inizializzazione della comunicazione seriale indicando la velocità baudrate [baud/s].

11) Serial.available()

Funzione già presente nell’IDE di Arduino che monitora ad ogni ciclo la porta seriale individuando se sono presenti dei dati utili nella comunicazione.

12) Serial.read()

Funzione già presente nell’IDE di Arduino che legge un byte alla volta dalla porta seriale.

13) int getEncoderTurn(void)

Funzione utilizzata per la lettura dello stato del rotary encoder, ovvero individua i movimenti effettuati da quest’ultimo distinguendo il numero di scatti e il loro senso di rotazione.

Non richiede alcun parametro e restituisce un intero che assume il significato del numero di scatti effettuati e in quale senso (positivo con rotazione oraria, negativo con rotazione antioraria).

14) int getJoystickValue(int drPin)

Funzione utilizzata per leggere lo stato del joystick, ovvero indicando con drPin il numero del pin a cui è collegato l’input riguardante l’asse X o Y, si legge il valore in ingresso della porta analogica. In base ad esso si determina un valore proporzionale contenuto nell’intervallo [-3; +3] che viene restituito al momento della chiamata.

15) void startGo(int dirLevel, int numRev, int motor)

Funzione utilizzata per il funzionamento automatico. Al momento della chiamata richiede di inserire tre valori, ovvero: il senso di rotazione dirLevel (HIGH per il senso orario, LOW per il senso antiorario), il numero numRev di giri da effettuare ed infine il numero motor del motore (1 per azionare il motore dell'asse X, 2 per quello dell'asse Y e 3 per quello dell'asse Z). La funzione, acquisiti questi parametri, inizia a settare il driver del motore impostando il senso di rotazione dirLevel e la modalità FULL-STEP (200 step per giro). A questo punto, invia un'onda quadra al driver di lunghezza proporzionale al numero di giri da effettuare. Durante questo funzionamento si monitora contemporaneamente anche lo stato dei vari switch finecorsa,nel caso in cui venissero premuti si blocca il movimento ed esce dalla modalità automatica.

16) void speedMode(int dirLevel, int value, int i)

Funzione utilizzata per il funzionamento manuale tramite joystick o rotary encoder. Al momento della chiamata vengono richiesti tre parametri, ovvero: il senso di rotazione dirLevel (HIGH per il senso orario, LOW per il senso antiorario), il numero value compreso tra -3 e +3 che indica la modalità di avanzamento ed infine il numero i del motore (1 per azionare il motore dell'asse X, 2 per quello dell'asse Y e 3 per quello dell'asse Z).

Questa funzione serve per settare i vari parametri nel driver relativo al motore i impostando il senso di rotazione dirLevel e la modalità di avanzamento. Per quest'ultima si intendono 6 possibili stati:

- 0: viene alimentato il pin ENABLE del driver in modo tale da bloccare il motore e tenerlo fermo;
- +1 o -1: viene settato il driver in modalità microstepping pari a ¼ di step. Il segno influisce solo nel senso di rotazione;
- +2 o -2: viene settato il driver in modalità microstepping pari a ½ di step. Il segno influisce solo nel senso di rotazione;
- +3 o -3: viene settato il driver in modalità FULL-STEP. Il segno influisce solo nel senso di rotazione.

Inoltre in questa funzione viene conteggiato ogni periodo dell'onda quadra che si invia al driver. Questo è necessario per determinare la posizione del motore. Il conteggio viene fatto tenendo conto degli angoli spazzati. Per esempio uno step in modalità +1 corrisponde a +1/800 di giro, mentre uno step in modalità +3 corrisponde a +1/200 di giro.

17) void updateCounter(void)

Funzione utilizzata per aggiornare le coordinate stampate nel display LCD. Al momento della chiamata la funzione aggiorna automaticamente le coordinate ricavate dal conteggio degli step effettuati.

18) void serialEvent(void)

Funzione utilizzata per la lettura dei dati inviati tramite seriale e il loro raggruppamento. Infatti Arduino riesce a leggere solamente un byte per volta, dunque rispettando la codifica imposta tra Processing e Arduino, questa funzione raggruppa e crea una stringa lunga 4 caratteri ovvero 4 byte. Quando viene effettuato questo raggruppamento la funzione modifica una variabile flag che comunica l'avvenuto raggruppamento dell'informazione e quindi il suo utilizzo.

# CAPITOLO 5: Descrizione del codice Processing di interfaccia utente per la definizione del percorso di misura

## 5.1 Aspetti generali sul funzionamento del codice

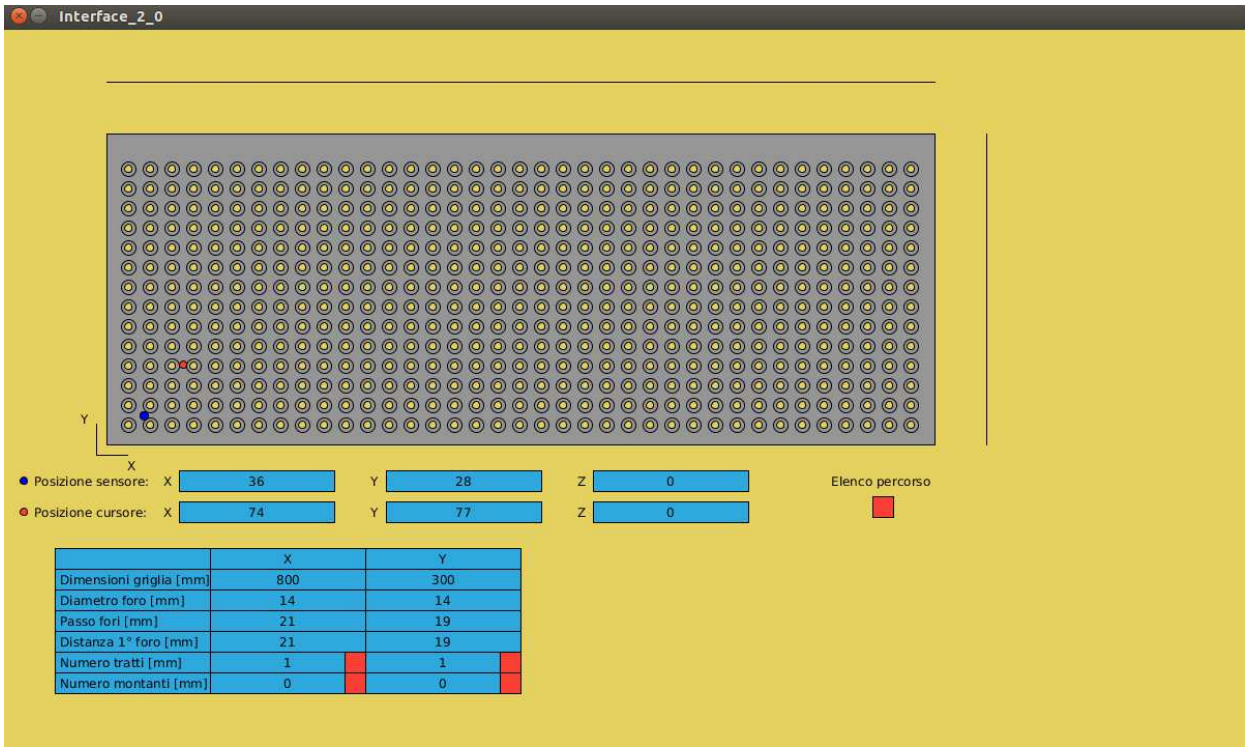
PROCESSING

Programma: INTERFACCIA

Programma utilizzato per interfacciare Arduino al PC con una sequenza di input per muovere la struttura collegata.

Lo scopo di questo programma è quello di rendere più semplice l'impostazione di un percorso da far eseguire al sensore lungo la griglia.

Grazie a questa interfaccia si può avere un disegno in 2D nello schermo del PC dove cliccare le posizioni in cui si vuole misurare il campo magnetico.




Il disegno schematico della griglia si costruisce inserendo alcuni dati all'interno della tabella presente all'avvio del programma, tra i vari parametri richiesti ci sono:

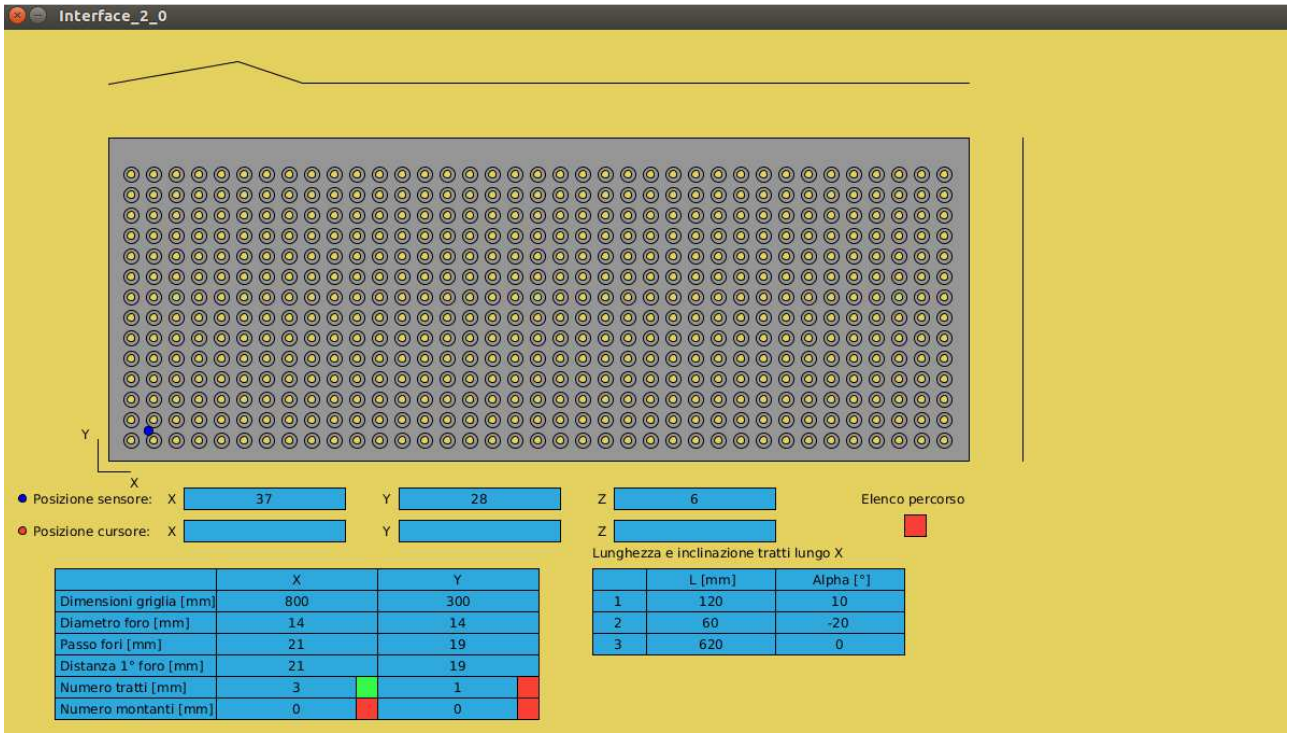
- dimensioni in X e Y della griglia racchiusa dalla struttura di misurazione;
- dimensioni fori dando un diametro in X e Y, questo perché c'è la possibilità di impostare una

forma ellittica ai fori;

- passo fori in X e in Y;
- distanza in X e in Y del primo foro rispetto dall'origine (0, 0, 0);
- numero tratti in X e in Y, inteso come il numero di tratti con diversa inclinazione, ovvero numero di tratti della retta spezzata che compone il profilo della griglia lungo le due direzioni;
- numero montanti in X e in Y, in cui si va ad indicare il numero dei montanti in direzione orizzontale o verticale.

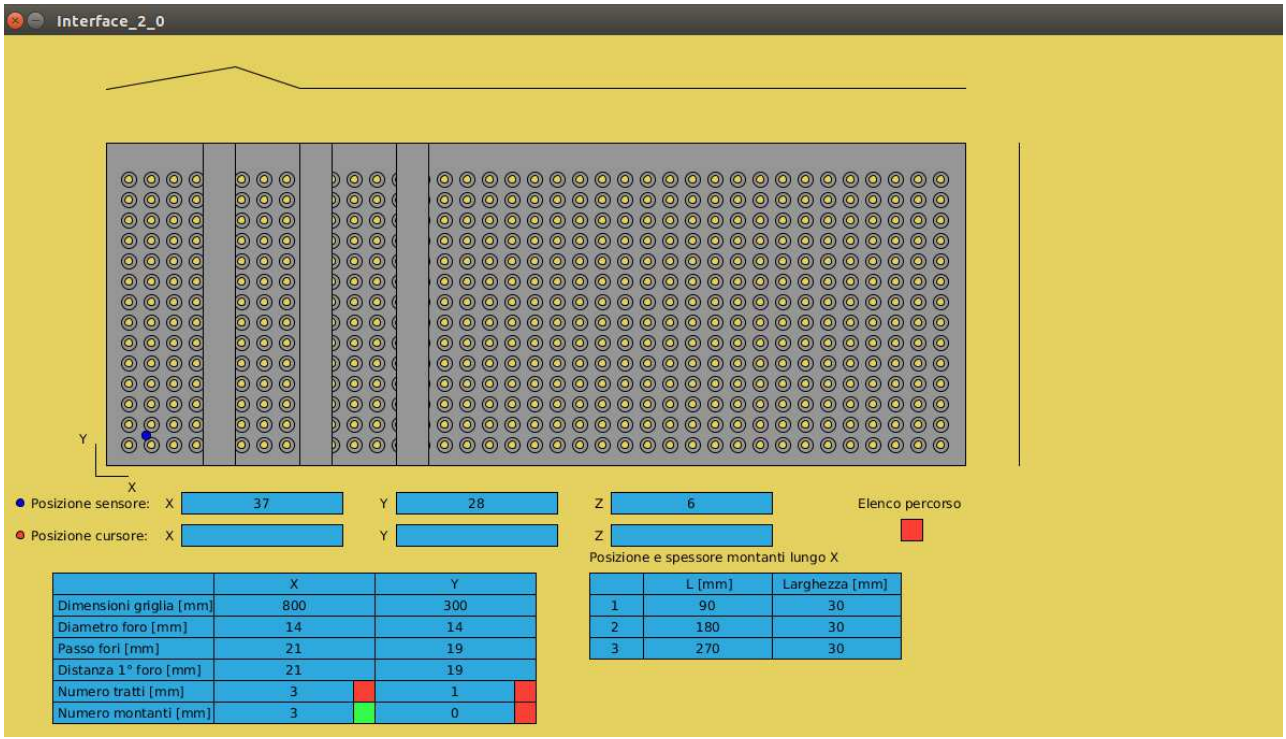
Per queste ultime due voci vengono associate delle tabelle aggiuntive attivabili, e quindi rese visibili, cliccando sul riquadro di colore rosso a destra della casella, nelle quali si possono specificare valori aggiuntivi come la lunghezza e l'inclinazione del tratto in questione ecc.

Nello specifico per quanto riguarda la tabella aggiuntiva collegata alla casella numero tratti in X o quella in Y, si richiede di scrivere la lunghezza del tratto in [mm] e l'inclinazione in [°].




Di default, il programma tiene conto di un unico tratto di lunghezza pari alla dimensione della griglia con inclinazione 0°. Aggiungendo un tratto l'ultima casella non si può modificare in quanto in essa viene scritta la distanza calcolata come dimensione complessiva della griglia meno la somma dei tratti che la precedono, mentre le inclinazioni sono tutte modificabili. Inoltre entrando in questa tabella si genera una linea spezzata sopra la griglia o a destra in base alla coordinata X o Y, che sta ad indicare il profilo della griglia in quella direzione per avere un'immagine complessiva dei tratti che compongono il profilo con i dati della tabella.

Per quanto riguarda la tabella aggiuntiva associata al numero dei montanti, verrà richiesto di immettere la distanza del montante in [mm] dall'origine e lo spessore, sempre in [mm], del montante.



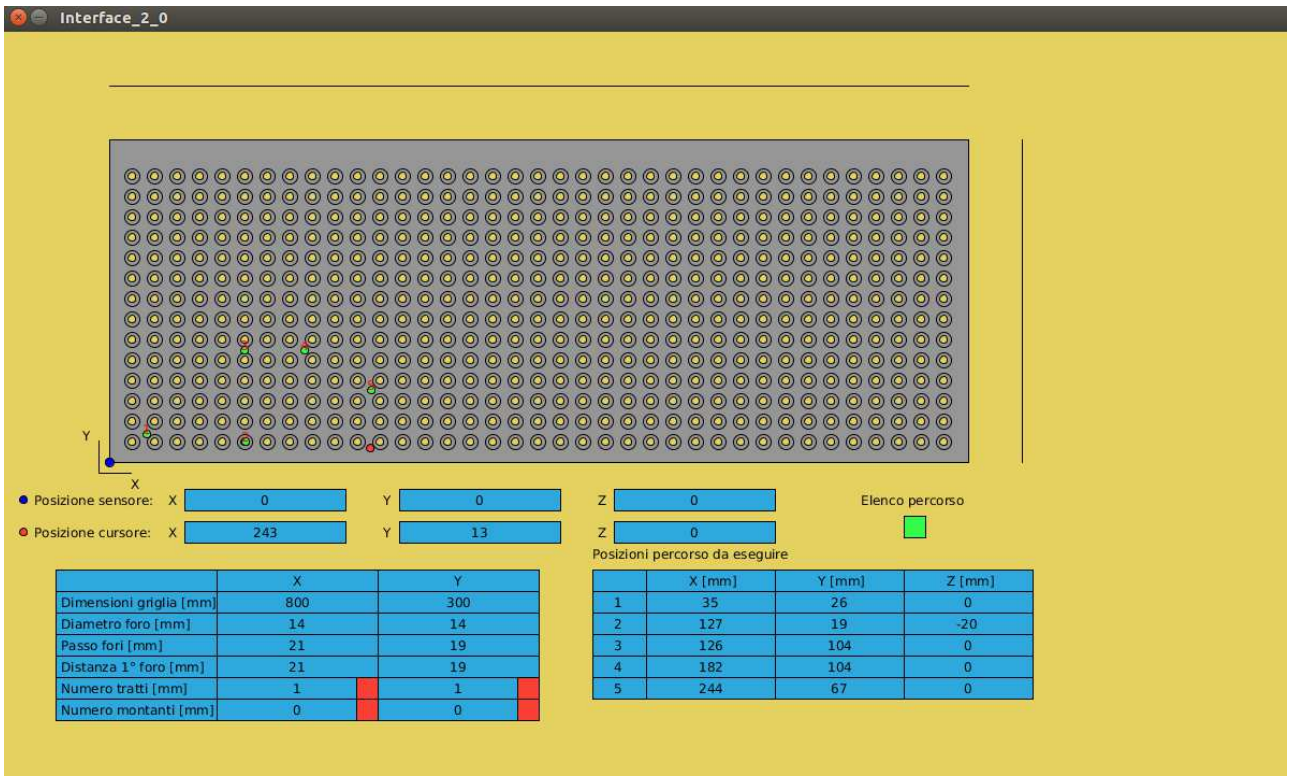

| | X | Y |
|---|---|---|
| Dimensioni griglia [mm] | 800 | 300 |
| Diametro foro [mm] | 14 | 14 |
| Passo fori [mm] | 21 | 19 |
| Distanza 1° foro [mm] | 21 | 19 |
| Numero tratti [mm] | 3 | 1 |
| Numero montanti [mm] | 3 | 0 |

| | L [mm] | Larghezza [mm] |
|---|---|---|
| 1 | 90 | 30 |
| 2 | 180 | 30 |
| 3 | 270 | 30 |

Scrivendo questi dati nella tabella compariranno nell'immagine della griglia dei rettangoli schematici che copriranno i fori sottostanti, in questo modo avremo un'immagine schematica della griglia su cui il sensore si muoverà.

Altre caratteristiche visibili nell'interfaccia sono le sei caselle sottostanti la griglia in cui compariranno le tre coordinate della posizione reale del sensore comunicate da Arduino e quelle del punto in cui viene posizionato il cursore del mouse all'interno della griglia. Alla destra di queste ultime tre caselle c'è un pulsante rosso con sopra la scritta "Elenco percorso:".

Questo pulsante, se premuto, diventa verde e fa comparire alla destra della tabella principale una tabella intitolata "Posizioni percorso da eseguire" composta da 4 colonne e un numero di righe che dipende dal numero di posizioni cliccate nella griglia. Ad ogni riga viene scritto il numero della posizione e le sue tre coordinate che verranno comunicate ad Arduino.




La comunicazione con Arduino avviene tramite la porta seriale (USB - "/dev/ttyACM0") nella quale avviene una comunicazione bilaterale in quanto vi è uno scambio di informazioni tra i due. Attraverso l'interfaccia si comunica ad Arduino le tre coordinate delle posizioni cliccate nella griglia. La comunicazione viene attivata ad ogni click del mouse sopra la griglia e il messaggio è così composto: (numero motore) + (posizione assoluta). Queste due informazioni vengono ripetute per tutte e tre le coordinate, dove il "numero motore" identifica con 1 la coordinata X, 2 la coordinata Y e 3 la coordinata Z.

Ultimo dettaglio riguardante la coordinata Z letta ed inviata ad Arduino: questa coordinata può assumere solamente due valori, ovvero 0 o -20. Questo perché con lo zero si comunica ad Arduino di fermarsi a filo della griglia, mentre con -20 si comunica ad Arduino che la posizione che si è scelta è un foro in cui si può entrare col sensore.

La scelta di questi due valori si basa sul colore del pixel nella griglia in cui viene cliccata la posizione, nello specifico:

- grigio corrisponde a 0, in quanto sta ad indicare una parte della griglia;
- giallo come lo sfondo corrisponde a -20, in quanto sta ad indicare un foro della griglia in cui si può entrare.

Inoltre ad ogni posizione comunicata ad Arduino, quindi trasmettendo le tre coordinate del punto, si comunica un quarto comando relativo al motore in Z in cui si invia una posizione di altezza di 50 mm sopra lo zero per la sicurezza negli spostamenti.

## 5.2 Codice

Vedi Appendice B.

## 5.3 Funzioni utilizzate

1) void setup()

Funzione già presente in Processing, necessaria per l'avvio del codice, in quanto comunica al programma quali parametri impostare, per esempio la dimensione della finestra di interfaccia o la porta seriale dove comunicare con Arduino o il carattere che segna la fine della comunicazione.

2) void draw()

Funzione già presente in Processing, necessaria per disegnare all'interno della finestra di interfaccia. Grazie ad essa è possibile variare lo stato di ogni singolo pixel interno alla finestra di lavoro e permette dunque a livello grafico di costruire forme geometriche o scritte e di cambiare il colore dello sfondo o di qualsiasi elemento presente.

3) void mousePressed()

Funzione già presente in Processing, necessaria per la gestione degli eventi di input da mouse. Infatti, si attiva solamente nell'eventualità che un tasto del mouse venga premuto, dunque all'interno di essa di può costruire un sistema di causa-effetto a cui si associa ad ogni posizione del cursore nel momento in cui viene premuto un tasto una modifica nell'interfaccia.

4) void keyPressed()

Funzione già presente in Processing, necessaria per la gestione degli eventi di input da tastiera. Infatti, si attiva solamente nell'eventualità che un tasto venga premuto, dunque all'interno di essa di può costruire un sistema di causa effetto a cui si associa ad ogni tasto una modifica nell'interfaccia come ad esempio la scrittura di un numero.

6) int color(int R, int G, int B)

Funzione presente in Processing, utilizzata per associare ad una variabile int il valore assunto dal colore scelto in base ai tre parametri: R, G, B. Ogni parametro può assumere un valore compreso tra 0 e 255. Questo valore indica l'intensità del colore in questione, dove R

corrisponde a rosso, G al verde e B al blu. Inoltre si può scrivere un unico valore di input per questa funzione e corrisponde allo stesso colore ottenuto ponendo tutti e tre i parametri uguali a quel valore. Se sono R, G e B sono tutti uguali si ottiene una scala di grigio: con 0 otteniamo il nero, con 255 il bianco.

5) background(color)

Funzione presente in Processing, utilizzata per l'impostazione del colore color per lo sfondo della finestra di lavoro.

6) Integer.parseInt(String x)

Funzione presente in Processing, utilizzata per la conversione della stringa x in un intero.

7) Float.parseFloat(String x)

Funzione presente in Processing, utilizzata per la conversione della stringa x in un float.

8) Integer.parseString(int x)

Funzione presente in Processing, utilizzata per la conversione del numero intero x in una stringa.

9) fill(color)

Funzione presente in Processing, utilizzata per la scelta del colore color da utilizzare per il riempimento delle figure costruite successive a questa funzione.

10) rect(int x, int y, int w, int h)

Funzione presente in Processing, utilizzata per costruire un rettangolo all'interno della finestra di lavoro. Al momento della sua chiamata, sono richiesti quattro parametri, ovvero le coordinate x e y dello spigolo superiore sinistro, cioè il numero di pixel x da destra e il numero di pixel y dall'alto. Gli altri due parametri richiesti sono relativi alla dimensione del rettangolo che si vuole disegnare ovvero la sua larghezza w e la sua altezza h, espressi anche loro in numero di pixel.

11) ellipse(int x, int y, int dx, int dy)

Funzione presente in Processing, utilizzata per costruire un'ellisse all'interno della finestra di lavoro. Al momento della sua chiamata, sono richiesti quattro parametri, ovvero:
le coordinare x e y del centro dell'ellisse, cioè il numero di pixel x da destra e il numero di pixel y dall'alto. Gli altri due parametri richiesti sono relativi alla dimensione dell'ellisse che si vuole disegnare ovvero il diametro in x dx e quello in y dy, espressi anche loro in numero di pixel.

12) text(String s, int x, int y)

Funzione presente in Processing, utilizzata per inserire nella finestra di lavoro un testo. Al momento della sua chiamata sono richiesti 3 parametri, ovvero il testo da scrivere sotto forma di stringa come s o come intero int o carattere char e le coordinare x e y del punto da cui iniziare la scritta, cioè il numero di pixel x da destra e il numero di pixel y dall'alto.

13) line(int x1, int y1, int x2, int y2)

Funzione presente in Processing, utilizzata per disegnare nella finestra di lavoro una linea. Al momento della sua chiamata sono richiesti 4 parametri, ovvero le coordinate x e y del punto iniziale (x1, y1) e del punto finale (x2, y2) del segmento.

14) delay(int x)

Funzione presente in Processin, utilizzata per impostare un ritardo di x millisecondi, in cui Processing sospende qualsiasi attività e aspetta il ritardo.

15) .write(String s)

Funzione presente in Processing, all'interno della libreria corrispondente alla comunicazione seriale, utilizzata per inviare dati. Quest'ultimo possono essere di tipo stringa come s oppure possono essere anche di tipo int o char.

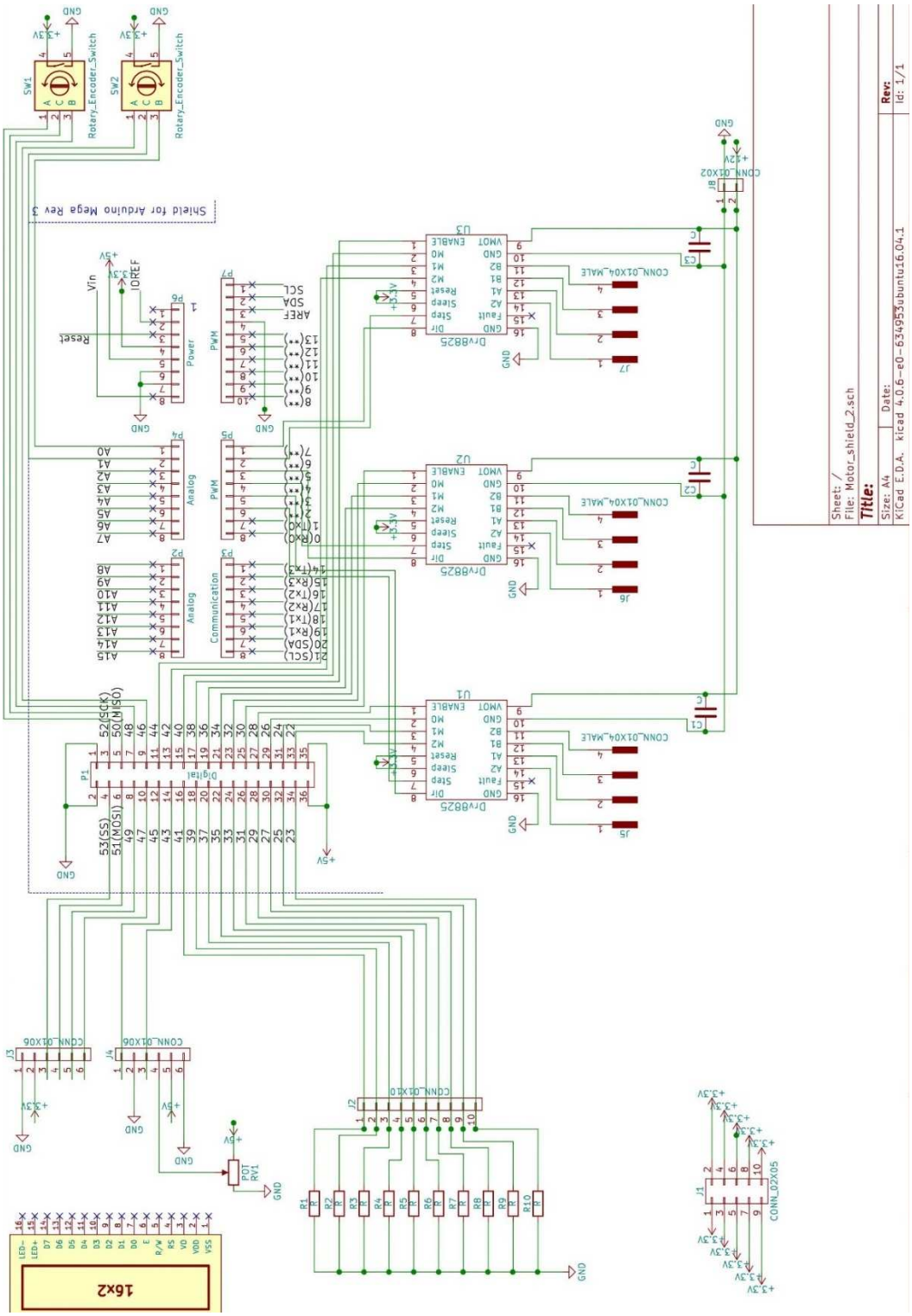
16) boolean overRect(int x, int y, int dx, int dy)

Funzione utilizzata per riconoscere se il cursore del mouse è sovrastante un rettangolo che ha lo spigolo superiore sinistro di coordinate x e y espresse in pixel e dimensioni date da w pixel di larghezza e h pixel di altezza. Se il cursore è all'interno del rettangolo la funzione restituisce true, altrimenti false.

# CAPITOLO 6: Realizzazione di una scheda per il circuito elettrico del sistema di controllo dei motori mediante Kicad

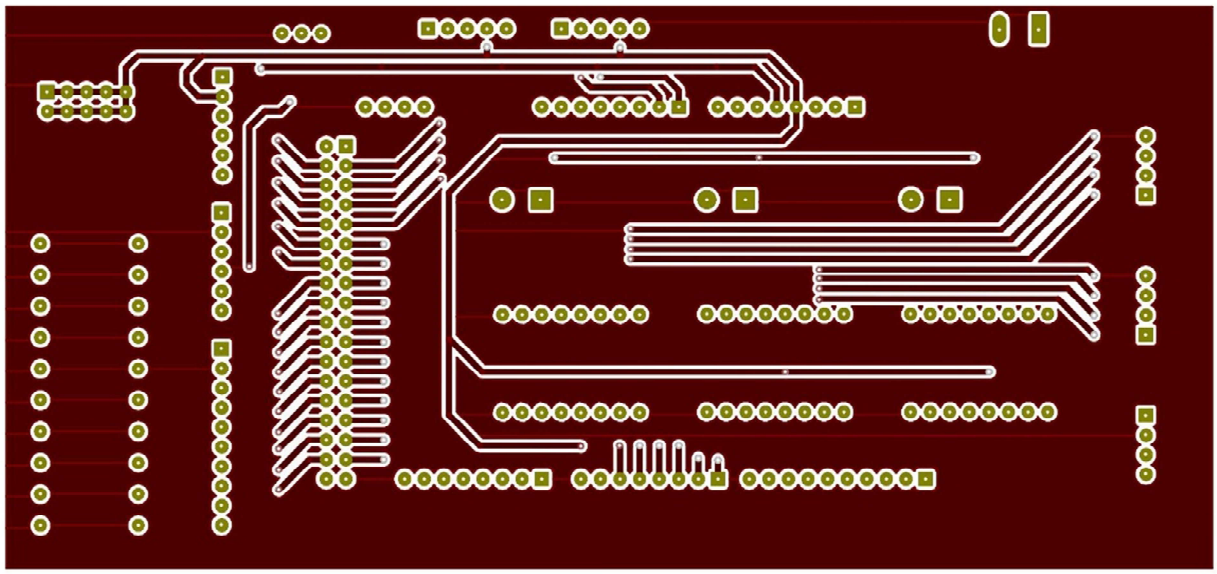
Per ovviare al cablaggio ingombrante tramite bredboard e cavetti, è stata disegnata una pcb da collegare direttamente sopra Arduino con il programma Kicad.

Grazie a quest'ultimo è stato possibile disegnare la pcb partendo da uno schema semplice dell'intero circuito, ovvero il seguente:

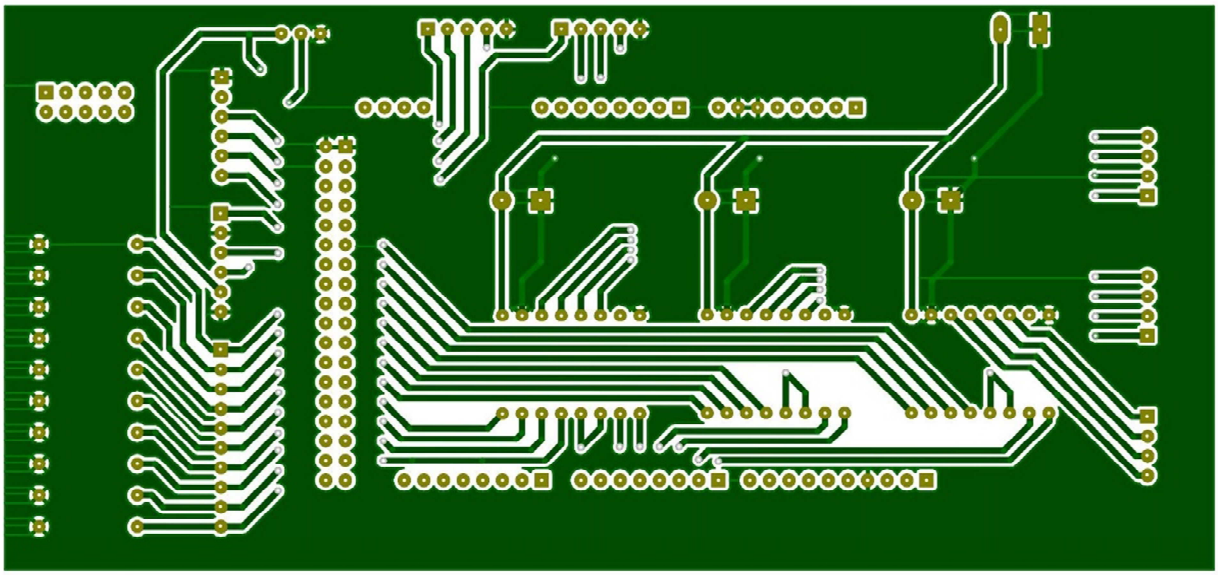


Attraverso le librerie presenti nel programma sono stati scelti i componenti da inserire nella pcb ed infine come ultimo step, è stato possibile collocare e disporre a piacimento i vari elementi del circuito con i propri collegamenti, ottenendo come risultato finale il seguente:

1) primo layer (lato rivolto verso l'alto)

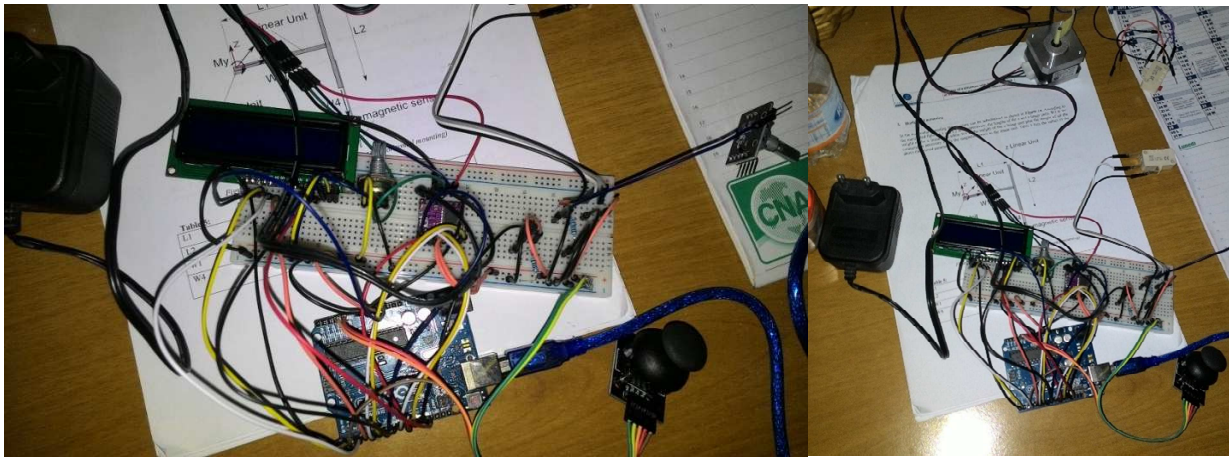


2) secondo layer (lato rivolto verso il basso a contatto con Arduino)



# CAPITOLO 7: Sviluppo ed evoluzione del progetto, realizzazione e collaudo

Il sistema per la misurazione del campo magnetico in 3 dimensioni è stato progettato in sei fasi.
La prima è stata quella di uno studio preliminare riguardo il controllo e azionamento dei motori a passo tramite Arduino, da cui è emersa la scelta dei vari componenti finalizzati alla prototipazione del sistema. Infatti, sono stati scelti dei driver in grado di controllare motori a passo anche con la tecnica del microstepping, la quale offre una maggiore precisione negli spostamenti. Inoltre, grazie alla scheda tecnica da catalogo della struttura esistente, è stato possibile avere una stima della coppia richiesta per far muovere i carrelli lungo ogni guida. Da queste ultime informazioni sono stati scelti dei motori in grado di offrire una coppia sufficiente ai movimenti richiesti per lo spostamento del sensore.
La seconda fase è di scomposizione e semplificazione del problema considerando un'unica direzione e quindi riconducendo il controllo di un solo motore in un sistema monoassiale. Questa semplificazione in termini di hardware e software ha permesso di sfruttare al meglio le funzioni di applicazione dei driver utilizzati per il pilotaggio dei motori a passo. Infatti con la gestione di un singolo driver è stato possibile concentrare lo studio nel movimento di un motore e quindi focalizzare l'attenzione sui vari parametri da imporre tramite Arduino.
Per questa seconda fase è stato richiesto l'utilizzo di un Arduino Uno, un driver 8825, un motore a passo, due switch finecorsa, un display LCD e un rotary encoder. Il cablaggio di questo sistema non è risultato complicato grazie anche all'utilizzo di unabreadboard e cavetti jumper. Vedi figura sottostante.



Lo scopo di questa fase è stato quello di pilotare un motore a passo tramite un encoder rotativo o un joystick. Grazie ad Arduino si è potuto leggere in input il segnale generato dall'encoder rotativo o dal joystick e, di conseguenza, associare ad ogni input una risposta da parte del motore. Arduino,

leggendo lo stato dell'encoder, inviava determinati segnali al driver. Nello specifico: il rotary encoder ruotando la manopolina aveva un movimento a scatti e ad ogni scatto coincideva una variazione dello stato con +1 o -1, variando in una scala di valori che andava da +3 a -3. Lo stesso sistema veniva sfruttato per il joystick, solamente che per quest'ultimo la lettura del valore in input era differente, infatti questo componente inviava ad Arduino un input analogico con un valore che va da 0 a 1023. Al fine di usare lo stesso sistema impiegato per il rotary encoder, per il joystick è stato applicato un adattamento di scala tra i due intervalli di valori [0, 1023] e [-3, 3]: in questo modo si è semplificata e uguagliata la gestione dei motori con diversi input.

Il numero che veniva letto in input coincideva con un differente stato del motore, infatti:

- 0: stop del motore;
- +1 o -1: avanzamento del motore con microstepping pari a ¼ step;
- +2 o -2: avanzamento del motore con microstepping pari a ½ step;
- +3 o -3: avanzamento del motore senza microstepping sfruttando l'intero step del motore.

In base al segno di questo input si è impostato il senso di rotazione, ovvero orario per valori positivi e antiorario per valori negativi.

Arduino, in base al numero in input appena illustrato, inviava come output dei segnali digitali al driver con lo scopo di pilotare il motore. Questi segnali sono proporzionali alle specifiche del driver, infatti per la gestione del microstepping è sufficiente guardare la tabella fornita dal produttore:

| MODE0 | MODE1 | MODE2 | Microstep Resolution |
|---|---|---|---|
| Low | Low | Low | Full step |
| High | Low | Low | Half step |
| Low | High | Low | 1/4 step |
| High | High | Low | 1/8 step |
| Low | Low | High | 1/16 step |
| High | Low | High | 1/32 step |
| Low | High | High | 1/32 step |
| High | High | High | 1/32 step |

Per quanto riguarda il verso di rotazione è stato necessario inviare un segnale HIGH o LOW al pin DIR, mentre per l'avanzamento degli step o microstep tutto si è regolato in base ad un segnale digitale, cioè ad onda quadra inviato al pin STEP. Quest'ultimo definiva la frequenza degli step o microstep e la loro durata.

Sempre nella seconda fase, oltre alla gestione del driver e del motore pilotati tramite rotary encoder e joystick, si sono inseriti successivamente uno schermo LCD e due switch finecorsa. Il primo ha avuto il compito di rendere visibile il conteggio dei giri effettuati dall'albero motore. Questo conto avveniva in Arduino con il conteggio dei periodi dell'onda quadra inviata al driver, ovvero tenendo conto del numero di step fatti e divisi per il numero di step o microstep per giro tramite la formula:

n° giri = (n° step o microstep effettuati) / (n° step o microstep per giro)

Esempio: se tramite rotary encoder si è ottenuto un input pari a +2, questo ha signifcato un output da parte di Arduino pari a:

- ENABLE = LOW;
- M0 = HIGH;
- M1 = LOW;
- M2 = LOW;
- DIR = HIGH;
- STEP = onda quadra con frequenza f.

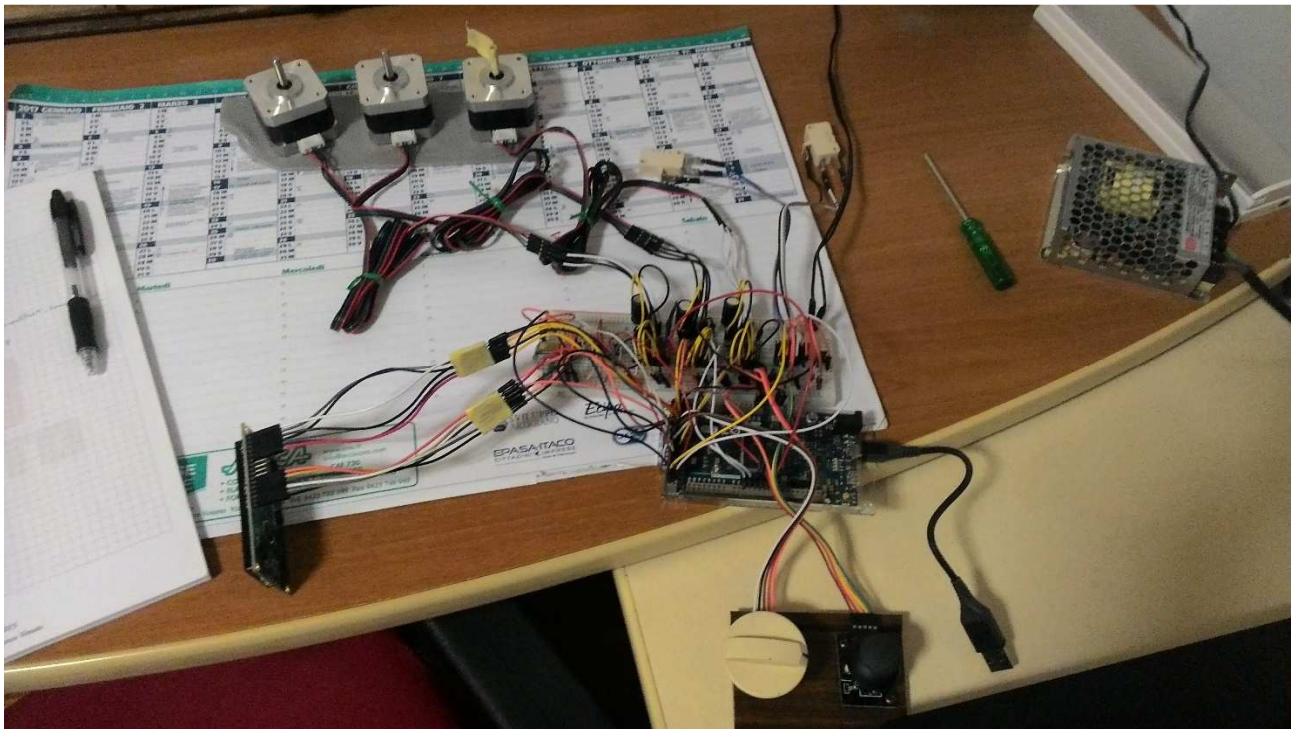
Tramite una funzione apposita si è tenuto conto del numero di step effettuati, ovvero il numero di periodi dell'onda quadra inviati al driver e il microstepping utilizzato. In questo caso, avendo microstepping pari a ½ step, si è ottenuto che per effettuare un'intera rivoluzione da parte dell'albero motore sono necessari 400 microstep, in quanto il motore a passo da catalogo è un motore a 200 step.

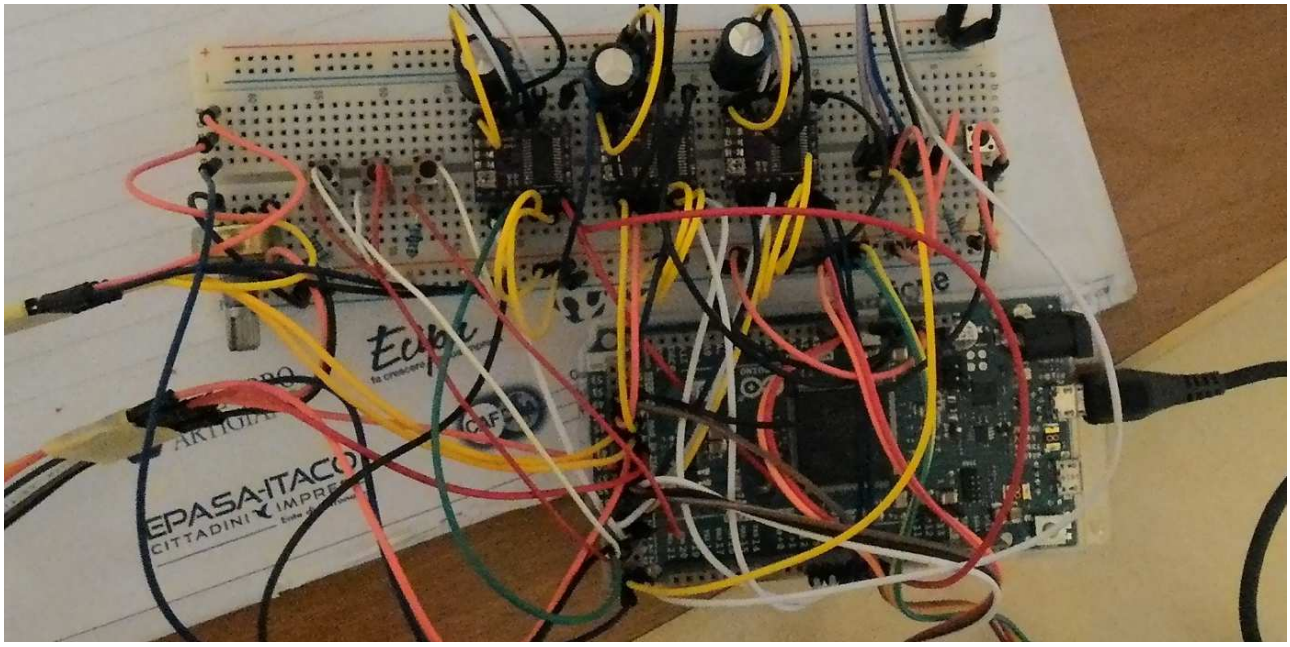
In questo modo non è stato utilizzato alcun sistema esterno per la misurazione della posizione, guadagnando dei vantaggi in termini di ingombro e di costi a scapito, però, di possibili problematiche in caso di perdita del passo, poiché Arduino non se ne accorgerebbe.

Gli ultimi componenti aggiunti in questa seconda fase sono stati i due switch finecorsa. Questi non sono altro che dei comuni interruttori che inviano ad Arduino un segnale LOW se premuti e HIGH viceversa. Lo scopo di questi due componenti è quello di indicare ad Arduino tramite segnale digitale appena descritto, se il carrellino è arrivato a fine corsa della guida. Nel caso in cui uno degli switch venga premuto, Arduino arresterà l'avanzamento del carrellino e quindi bloccherà il motore in quel senso di rotazione, consentendo di ritornare indietro con senso di rotazione inverso.

La terza fase è stata caratterizzata dalla stesura del codice pensato non più al problema monoassiale, ma a quello tridimensionale. Il passaggio ha richiesto di replicare per tre motori le istruzioni e funzioni usate nel singolo motore nella fase precedente. Per controllare in modo manuale il movimento dei tre motori sono stati utilizzati un joystick per la gestione degli assi X e Y e il rotary

encoder per l'asse Z. Nella stesura del codice si sono tenuti presenti anche i due switch finecorsa per ogni asse e le posizioni stampate nello schermo LCD. Portata a termine la stesura del codice, è stato assemblato l'intero sistema, come si vede in figura.






Dalla figura si possono notare alcuni componenti non ancora citati: dopo il cablaggio e la verifica del corretto funzionamento del sistema sopra descritto, sono stati aggiunti 4 pulsanti. Ad ogni pulsante è stata associata una certa operazione, secondo cui, partendo da sinistra, il primo tasto corrisponde all'azzeramento delle tre coordinate, segnalando così al sistema la posizione dell'origine, il secondo

coincide alla taratura dell'asse X e il terzo dell'asse Y. Il quarto ed ultimo pulsante, invece, se premuto, avvia il funzionamento automatico.

Nello specifico, la taratura è stata pensata nell'ovviare al problema di una possibile inclinazione dell'asse della guida rispetto la griglia, per ipotesi dritta. La taratura inoltre richiede alcune operazioni preliminari, ovvero il posizionamento di tre punti sulla griglia ad una distanza certa (come ad esempio i punti: (0, 0, 0) ossia l'origine degli assi; (200, 0, 0) cioè un punto distante 20cm rispetto l'origine solo in X; (0, 200, 0) ossia un punto distante 20cm rispetto l'origine solo in Y). In questo modo si comunica ad Arduino che la distanza percorsa e registrata nell'istante in cui ci si posiziona in questi punti noti, corrisponde alla distanza appena descritta. Attraverso il teorema di Pitagora si vanno a calcolare il nuovo numero di step per giro e la traslazione che subisce la coordinata Z ad ogni giro.

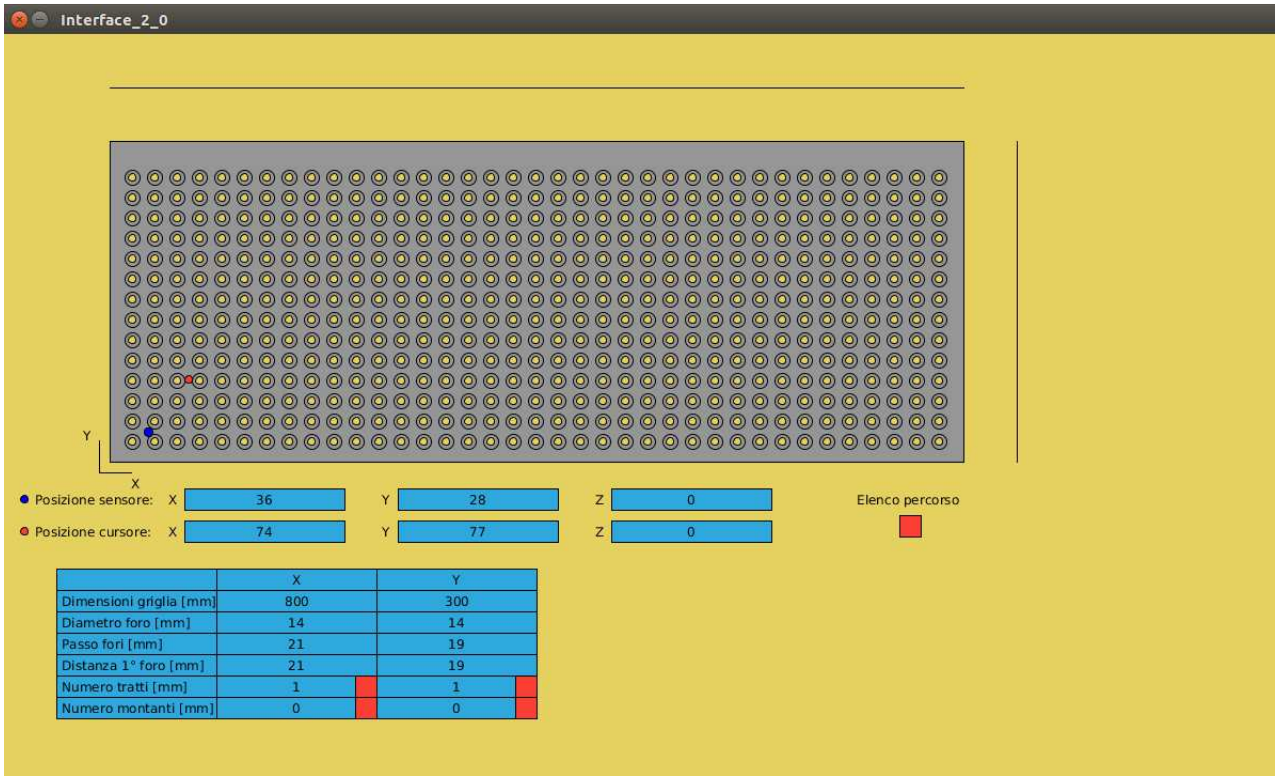
Per quanto riguarda il funzionamento automatico, questo è stato pensato in modo tale che introducendo in ingresso un elenco di coordinate, Arduino faccia muovere i motori in un determinato ordine al fine di raggiungere le posizioni scelte. Per comunicare le posizioni, si è pensato di sfruttare un array e di compilarlo all'inizio pagina del codice per comodità. L'array è stato così strutturato: {numero motore da avviare; posizione assoluta da raggiungere; numero di motore da avviare; posizione assoluta da raggiungere; …}. Per "numero motore da avviare" si intende:

- 1 per il motore X;
- 2 per Y;
- 3 per Z.

Nella quarta fase della programmazione, si è aggiunto al sistema un'interfaccia utente con lo scopo di facilitare l'immissione delle posizioni nel funzionamento automatico del sistema. Per costruire una finestra di interfaccia è stato utilizzato il programma open source "Processing".

Questo programma è adatto e consigliato per la comunicazione seriale con Arduino ed è programmabile con linguaggio Java. L'idea principale è stata quella di ricreare in modo schematico una griglia tramite l'inserimento di alcuni dati da parte dell'utente all'interno di apposite tabelle.

L’interfaccia è così strutturata:




Come si può vedere, nella metà superiore della finestra è disegnata la griglia schematica al cui interno sono posti i fori, sopra di essa e alla sua destra sono state create due linee spezzate per raffigurare il profilo degli assi X e Y. Nella metà inferiore, invece, sono state poste le tabelle dei dati. All’avvio si troverà: una tabella 3x7 che viene identificata come tabella principale, sei caselle associate alle tre coordinate del sensore e del cursore nella griglia e un pulsante rosso dedicato all’elenco delle posizioni selezionate per il percorso. La tabella principale rimane sempre fissa nella schermata e contiene i parametri più importanti. Come si può vedere dall’immagine, in questa tabella sono inserite le dimensioni della griglia, le dimensioni dei fori e il loro passo in X e in Y. Inoltre viene richiesto anche il numero di tratti e il numero di montanti per asse. In queste due ultime righe per ogni casella a fianco al numero a destra è stato inserito un pulsante rosso. Se quest’ultimo viene premuto appare alla destra della tabella principale una tabella aggiuntiva relativa alla casella in cui vi è posto il pulsante. Ad esempio per quanto riguarda la tabella aggiuntiva corrispondente al numero di tratti, vengono richieste la lunghezza e l’inclinazione di ognuno di essi. Invece per il numero di montanti vengono richieste la distanza dall’origine e la loro larghezza.

Attraverso l’inserimento di tutti i dati presenti nelle tabelle viene ricostruita una griglia che vuole essere simile alla realtà ridimensionando in modo proporzionale le dimensioni della griglia e la posizione e dimensione dei fori. Inoltre verranno costruiti i profili in X e in Y tramite una linea spezzata a lato di ciascun asse e l’inserimento di rettangoli rappresentanti i montanti.

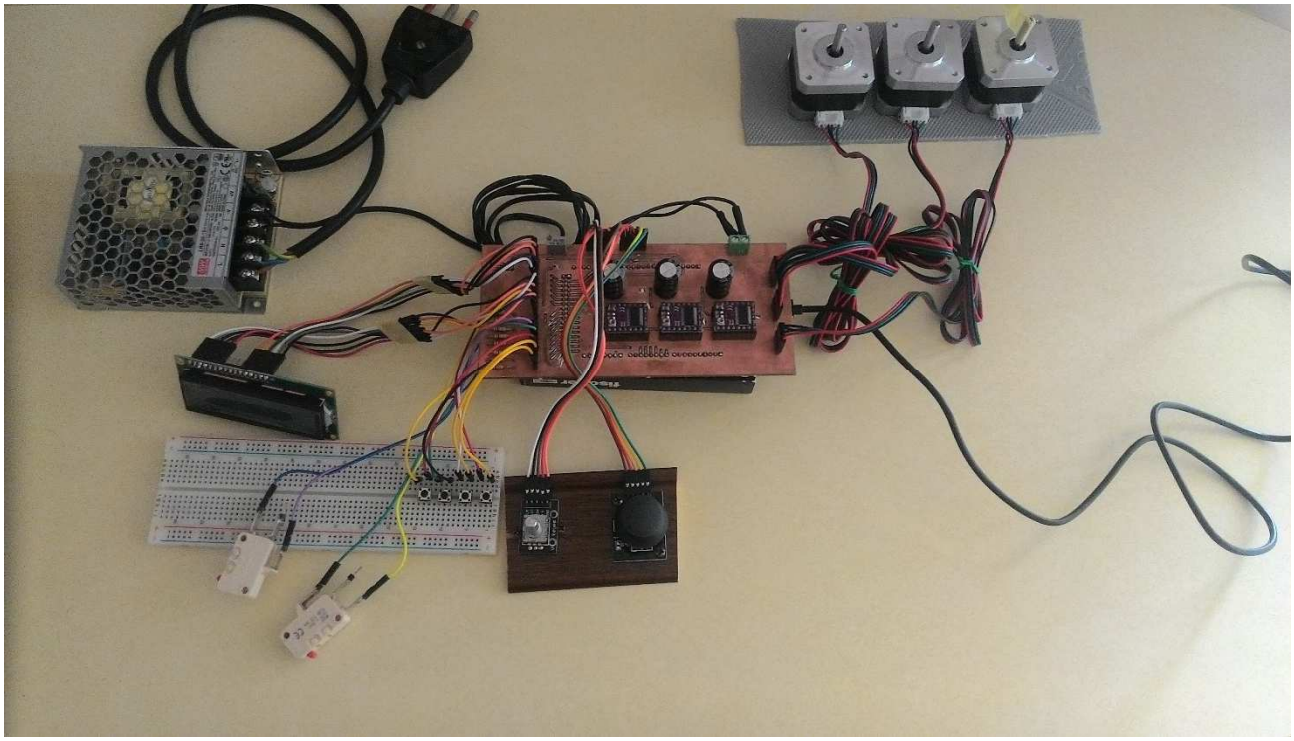
Altra tabella importante è quella relativa all'elenco posizioni selezionate, la quale è attivabile cliccando sul pulsante posto sotto la griglia a sinistra. In questa griglia appaiono le coordinate delle posizioni selezionate tramite click sinistro del mouse sopra la griglia. Se invece viene premuto il tasto destro viene cancellata l'ultima posizione inserita.

L'interfaccia, al momento dell'inserimento delle posizioni cliccate nella griglia, invia subito le coordinate ad Arduino che le salva nell'array sopra citato. In questo modo è stato possibile interfacciare l'utente con una schermata intuitiva piuttosto che agire direttamente dal codice di Arduino.

Nella quinta fase si è testata la coppia di un motore alzando un peso di 1kg. Lo scopo del test è stato quello di capire se la dimensione del motore usato in fase preliminare è sufficiente nel caso peggiore d'esercizio.

Il test è stato effettuato in laboratorio collegando uno dei motori ad una guida della struttura e legando al carrellino un peso di 1kg circa. Dopo alcuni movimenti si è notato che il motore ha avuto un'ottima risposta all'input del joystick senza alcuna perdita di passi e dunque si è concluso con la scelta di un motore simile a quello utilizzato.

La sesta e ultima fase ha visto l'utilizzo del programma Kicad per il disegno e progettazione di una scheda da poter collegare direttamente ad Arduino ovviando così al problema del cablaggio ingombrante tramite breadboard.



# CAPITOLO 8: Proposte

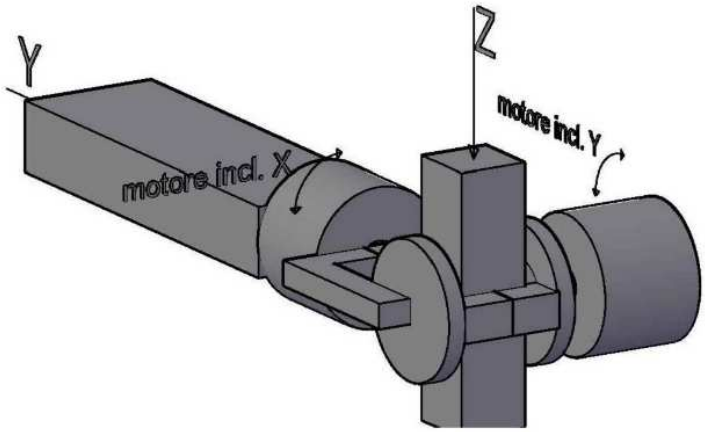
Il sistema progettato può sicuramente essere migliorato sia a livello hardware che software.

Una prima proposta riguarda l'aggiunta di encoder ottici esterni per una migliore precisione nella posizione del sensore. Questi ultimi, utilizzati ad esempio nelle comuni stampanti, garantirebbero una posizione accurata e reale del sensore, indipendentemente dal funzionamento del sistema. Infatti nel progetto di tesi, per la determinazione della posizione, viene effettuato un conteggio dei periodi dell'onda quadra generata per il controllo dei driver, rendendo il sistema debole in caso di perdita di passi dovuta ad esempio all'impuntamento di un motore.

Una seconda proposta corrisponde all'inserimento di un sensore in grado di segnalare un allarme al sistema in caso di sforzo o impuntamento da parte della sonda a contatto con la griglia. Ricercando in internet, un possibile sistema, adottato nella maggior parte dei casi da macchinari CNC, è l'utilizzo di un sensore in grado di misurare la coppia erogata da ogni motore captando lo sforzo o la dissipazione di energia eccessiva causata dal bloccaggio di esso. Questa opzione è svantaggiosa in termini di costi e gestione dei sensori. Una seconda soluzione ben più economica è l'utilizzo di un filo in tensione attorno la sonda, che, al contatto con la griglia scaricando una certa corrente, segnali al sistema l'avvenuta collisione. Una terza soluzione ben più semplice è quella di posizionare un interruttore nella parte superiore della sonda, in modo tale che, nel caso in cui la sonda si scontri con la griglia, prema il pulsante. Quest'ultima opzione molto semplice garantisce una maggiore sicurezza per prevenire una possibile collisione della sonda sulla griglia, con l'unico svantaggio di essere efficace solamente nel caso di movimenti perpendicolari e non laterali, come ad esempio all'interno dei fori.

Una terza proposta riguarda l'aggiunta di due motori a passo di piccole dimensioni per l'aggiornamento dell'inclinazione dell'asse Z. Questa modifica garantirebbe l'ortogonalità del sensore lungo tutta la griglia, permettendo, dunque, di seguire perpendicolarmente i suoi profili che risultano essere a tratti inclinati di qualche grado uno rispetto all'altro. La progettazione di questo componente non costituirebbe una grossa spesa, in quanto, progettato con programma CAD, si potrebbe stampare tramite stampante 3D e i motori, essendo di piccole dimensioni, hanno un basso costo. Il sistema, infatti, posizionato alla giunzione dell'asse Z con l'asse Y, non deve supportare una grossa coppia per il peso molto ridotto della guida minore che costituisce l'asse Z.

Questo componente potrebbe avere una forma simile a quella in figura:




Una quarta proposta ha come obiettivo il miglioramento della comunicazione tra Processing e Arduino. Nello specifico, si potrebbe aggiungere la possibilità di immettere una nuova posizione o di modificarla manualmente. Questa miglioria ovvierebbe al problema del posizionamento accurato del mouse sulla griglia schematica nell'interfaccia.

Una quinta proposta, riguardante sempre la parte software del sistema, è l'integrazione del Gaussmetro al controllo di posizionamento. Questa comporterebbe una semplificazione a livello di gestione del sistema, in quanto nello stesso programma si potrebbero gestire insieme la movimentazione del sensore e la descrizione dei risultati ottenuti dalla misurazione dello strumento in base alla sua posizione. Questa modifica non è sicuramente semplice da effettuare, in quanto richiederebbe tempo nello studiare a fondo il funzionamento del Gaussmetro e la modalità di output dei suoi dati. Il vantaggio che si può ottenere sta sicuramente nel rendere il sistema di misurazione completo e autonomo.

# CONCLUSIONI

Il progetto di tesi ultimato ha risposto in modo ottimale ai test effettuati. Il controllo della struttura tramite joystick e rotary encoder, per quanto riguarda il comando manuale, e l'interfaccia, per quello automatico, risultano funzionali all'applicazione per cui è richiesta la struttura.

Nonostante il progetto risulti completo, può essere oggetto di miglioramenti come è stato spiegato nel capitolo delle proposte.

La progettazione e la costruzione della pcb ha portato notevoli vantaggi in termini di semplicità e di ingombro. Infatti il cablaggio della struttura risulta meno confusionario e più ordinato, ma soprattutto funzionale per i collegamenti agli elementi esterni come motori, driver, joystick ecc.

Un'altra caratteristica vantaggiosa, secondo me, è il funzionamento indipendente di Arduino rispetto all'interfaccia, ovvero la struttura può essere pilotata anche senza l'avvio di quest'ultima. Questo aspetto anche se banale, garantisce un funzionamento della struttura guidata tramite joystick e rotary encoder, garantendo dunque un uso rapido senza le varie impostazioni richieste dall'interfaccia riguardo la griglia su cui effettuare le misurazioni.

# APPENDICE A

```
#include <LiquidCrystal.h>  // richiamo della libreria per gestire lo schermo lcd

LiquidCrystal lcd(43, 45, 47, 49, 51, 53);  // inizializzazione dello schermo lcd indicando i pin a cui viene collegato lo schermo

// definiti come costanti tutti i pin a cui vengono collegati i vari collegamenti al sistema di controllo
#define swPin 46
#define dryPin A1
#define drxPin A0
#define swEncPin 52
#define clkPin 50
#define dtPin 48
#define switchXSxPin 33
#define switchXDxPin 31
#define switchYSxPin 33
#define switchYDxPin 31
#define switchZSxPin 33
#define switchZDxPin 31
#define buttonProgramPin 29
#define buttonOnePin 27
#define buttonTwoPin 25
#define buttonThreePin 23

#define MOTOR_X 1 // definizione della costante che identifica il motore collegato alla guida lungo l'asse X
#define stepXPin 2
#define dirXPin 3
#define enableXPin 28
#define M0XPin 26
#define M1XPin 24
#define M2XPin 22

#define MOTOR_Y 2 // definizione della costante che identifica il motore collegato alla guida lungo l'asse Y
#define stepYPin 4
#define dirYPin 5
#define enableYPin 30
#define M0YPin 32
#define M1YPin 34
#define M2YPin 36

#define MOTOR_Z 3 // definizione della costante che identifica il motore collegato alla guida lungo l'asse Z
#define stepZPin 6
#define dirZPin 7
#define enableZPin 38
#define M0ZPin 40
#define M1ZPin 42
#define M2ZPin 44
```

```
int encoderVal, joystickValX, joystickValY; // inizializzazione varibili globali che comunicano la posizione della manopolina e del joystick

int dirLevel; // variabile utilizzata per comunicare ai driver la direzione di rotazione del motore in questione
int dpause, flagX, flagY, flagZ, flag;  // variabili che determinano la frequenza dell' onda quadra generata

// array che descrive il percorso indicando prima il motore da far azionare e poi il numero di giri da effettuare
int program[] = {MOTOR_X, 24, MOTOR_Y, 21, MOTOR_Z, -20, MOTOR_Z, 20, MOTOR_Y, 10, MOTOR_X, -16};
int program2[42];
int count;

long int countX, countRevX, countY, countRevY, countZ, countRevZ, countRevX_old, countRevY_old, countRevZ_old; // variabili per il conteggio degli step e delle rivoluzioni eseguite da ogni singolo motore
// variabili utilizzate per la taratura degli assi, quindi quanti step per un giro e traslazione lungo Z
int stepForRevX, stepForRevY, stepForRevZ, traslationZ_axis_X, traslationZ_axis_Y;

int tollerance, distanceHoleX, distanceHoleY, firstHoleX, firstHoleY;

String inputString = "";         // a string to hold incoming data
boolean stringComplete = false;

void setup()
{
  pinMode(swPin, INPUT);
  pinMode(drxPin, INPUT);
  pinMode(dryPin, INPUT);
  pinMode(swEncPin, INPUT);
  pinMode(clkPin, INPUT);
  pinMode(dtPin, INPUT);
  pinMode(switchXSxPin, INPUT);
  pinMode(switchXDxPin, INPUT);
  pinMode(switchYSxPin, INPUT);
  pinMode(switchYDxPin, INPUT);
  pinMode(switchZSxPin, INPUT);
  pinMode(switchZDxPin, INPUT);
  pinMode(buttonProgramPin, INPUT);
  pinMode(buttonOnePin, INPUT);
  pinMode(buttonTwoPin, INPUT);
  pinMode(buttonThreePin, INPUT);
  digitalWrite(swPin, HIGH);
  digitalWrite(swEncPin, HIGH);

  pinMode(stepXPin, OUTPUT);
  pinMode(dirXPin, OUTPUT);
  pinMode(enableXPin, OUTPUT);
  pinMode(M0XPin, OUTPUT);
  pinMode(M1XPin, OUTPUT);
```

```
pinMode(M2XPin, OUTPUT);

pinMode(stepYPin, OUTPUT);
pinMode(dirYPin, OUTPUT);
pinMode(enableYPin, OUTPUT);
pinMode(M0YPin, OUTPUT);
pinMode(M1YPin, OUTPUT);
pinMode(M2YPin, OUTPUT);

pinMode(stepZPin, OUTPUT);
pinMode(dirZPin, OUTPUT);
pinMode(enableZPin, OUTPUT);
pinMode(M0ZPin, OUTPUT);
pinMode(M1ZPin, OUTPUT);
pinMode(M2ZPin, OUTPUT);

countX =  0;
countRevX = 0;
stepForRevX = 800;
countY =  0;
countRevY = 0;
stepForRevY = 800;
countZ =  0;
countRevZ = 0;
stepForRevZ = 800;
traslationZ_axis_X = 0;
traslationZ_axis_Y = 0;


countRevX_old = 0;
countRevY_old = 0;
countRevZ_old = 0;

dpause = 400;
flagX = 0;
flagY = 0;
flagZ = 0;

tollerance = 3;
distanceHoleX = 21;
distanceHoleY = 19;
firstHoleX = 0;
firstHoleY = 0;

lcd.begin(16, 2);
lcd.setCursor(0,0);
lcd.print("X mm  Y mm  Z mm");
lcd.setCursor(0,1);
lcd.print("+000  +000  +000");

inputString.reserve(4);
```

```
  Serial.begin(115200);
  count = 0;
  for(int i = 0; i < (sizeof(program2)/sizeof(int)); i++)
    program2[i] = 0;
}

void loop()
{
  // lettura rotary encoder

  int change = getEncoderTurn();
  encoderVal = encoderVal + change;

    if(encoderVal == 1)
    {
      digitalWrite(enableZPin, LOW);
      digitalWrite(M0ZPin, LOW);
      digitalWrite(M1ZPin, LOW);
      digitalWrite(M2ZPin, LOW);
      digitalWrite(dirZPin, HIGH);
      for(int i = 0; i < 200 && digitalRead(switchZDxPin) == HIGH; i++)
      {
        digitalWrite(stepZPin, HIGH);
        delayMicroseconds(500);
        digitalWrite(stepZPin, LOW);
        delayMicroseconds(500);
        countZ = countZ + 4;
      }
    }
    else if(encoderVal == -1)
    {
      digitalWrite(enableZPin, LOW);
      digitalWrite(M0ZPin, LOW);
      digitalWrite(M1ZPin, LOW);
      digitalWrite(M2ZPin, LOW);
      digitalWrite(dirZPin, LOW);
      for(int i = 0; i < 200 && digitalRead(switchZSxPin) == HIGH; i++)
      {
        digitalWrite(stepZPin, HIGH);
        delayMicroseconds(500);
        digitalWrite(stepZPin, LOW);
        delayMicroseconds(500);
        countZ = countZ - 4;
      }
    }
    encoderVal = 0;
    digitalWrite(enableZPin, HIGH);
```

```
 // lettura joystick

  joystickValX = getJoystickValue(drxPin);  // acquisisce un numerocompreso tra -3 e 3 in base alla posizione del joystick nell' asse X
  joystickValY = getJoystickValue(dryPin);  // acquisisce un numerocompreso tra -3 e 3 in base alla posizione del joystick nell' asse Y
  if(digitalRead(swPin) == LOW)  // se premuta la levetta del joystick porta il valore a 0 e ferma i motori
  {
    joystickValX = 0;
    joystickValY = 0;
    delay(1000);  // tempo di 1 s per riportare la levetta alla posizione di stallo
  }

  // lettura switch fine corsa

  // se viene premuto lo switch di fine corsa in testa alla guida consente di tornare indietro, ma non di avanzare
  if(digitalRead(switchXDxPin) == LOW && joystickValX > 0)
    joystickValX = 0;
  else if(digitalRead(switchXDxPin) == LOW && joystickValX < 0)
    joystickValX = joystickValX;

  // se viene premuto lo switch di fine corsa in fondo alla guida consente di tornare indietro, ma non di avanzare
  if(digitalRead(switchXSxPin) == LOW && joystickValX < 0)
    joystickValX = 0;
  else if(digitalRead(switchXSxPin) == LOW && joystickValX > 0)
    joystickValX = joystickValX;

  // se viene premuto lo switch di fine corsa in testa alla guida consente di tornare indietro, ma non di avanzare
  if(digitalRead(switchYDxPin) == LOW && joystickValY > 0)
    joystickValY = 0;
  else if(digitalRead(switchYDxPin) == LOW && joystickValY < 0)
    joystickValY = joystickValY;

  // se viene premuto lo switch di fine corsa in fondo alla guida consente di tornare indietro, ma non di avanzare
  if(digitalRead(switchYSxPin) == LOW && joystickValY < 0)
    joystickValY = 0;
  else if(digitalRead(switchYSxPin) == LOW && joystickValY > 0)
    joystickValY = joystickValY;

  // se viene premuto lo switch di fine corsa in testa alla guida consente di tornare indietro, ma non di avanzare
  if(digitalRead(switchZDxPin) == LOW && encoderVal > 0)
    encoderVal = 0;
  else if(digitalRead(switchZDxPin) == LOW && encoderVal < 0)
    encoderVal = encoderVal;
```

```
  // se viene premuto lo switch di fine corsa in fondo alla guida consente di tornare indietro, ma non di avanzare
  if(digitalRead(switchZSxPin) == LOW && encoderVal < 0)
    encoderVal = 0;
  else if(digitalRead(switchZSxPin) == LOW && encoderVal > 0)
    encoderVal = encoderVal;

  // funzionamento manuale
  // richiama la funzione speedMode() per le impostazioni di avanzamento dei motori

  speedMode(dirLevel, joystickValX, MOTOR_X);
  speedMode(dirLevel, joystickValY, MOTOR_Y);
  //speedMode(dirLevel, encoderVal, MOTOR_Z);


  // genera l'onda quadra per un singolo step o microstep con frequenza 1.25 kHz se la varibile flag è minore di 1000.
  // la variabile flag indica la permanenza dei motori nelle modalità FULL-STEP e di conseguenza alza la frequenza per aumentare la velocità.
  // Si parte da una frequenza minima di 1.25 kHz fino ad una massima di 1.43 kHz (dpause = 350us)
  flag = flagX + flagY + flagZ;

  switch(flag)
  {
    case 0:
      dpause = 400;
      break;
    case 1000:
      dpause = 395;
      break;
    case 1500:
      dpause = 390;
      break;
    case 2000:
      dpause = 385;
      break;
    case 2500:
      dpause = 380;
      break;
    case 3000:
      dpause = 375;
      break;
    case 3500:
      dpause = 370;
      break;
    case 4000:
      dpause = 365;
      break;
    case 4500:
      dpause = 360;
      break;
```

```
    case 5000:
      dpause = 355;
      break;
    case 5500:
      dpause = 350;
      break;
  }
  // se il joystick non è nella posizione di stallo lungo l'asse X mette in uscita al stepXPin un gradino alto per creare l'onda quadra
  if(joystickValX != 0)
  {
    digitalWrite(enableXPin, LOW);
    digitalWrite(stepXPin, HIGH);
  }
  else
    digitalWrite(enableXPin, HIGH);
  // se il joystick non è nella posizione di stallo lungo l'asse Y mette in uscita al stepYPin un gradino alto per creare l'onda quadra
  if(joystickValY != 0)
  {
    digitalWrite(enableYPin, LOW);
    digitalWrite(stepYPin, HIGH);
  }
  else
    digitalWrite(enableYPin, HIGH);
  // se il rotary encoder non è nella posizione di stallo mette in uscita al stepZPin un gradino alto per creare l'onda quadra
  if(encoderVal != 0)
  {
    digitalWrite(enableZPin, LOW);
    digitalWrite(stepZPin, HIGH);
  }
  else
    digitalWrite(enableZPin, HIGH);
  // se il joystick e il rotary encoder sono nella posizione di stallo e quindi i motori non sono in movimento si stampano le coordinate lungo i tre assi
  if(joystickValX == 0 && joystickValY == 0 && encoderVal == 0)
    updateCounter();
  countRevX = countX / stepForRevX;  // salva il valore di giri dopo l' avanzamento di uno step
  String message = String(countRevX) + ',';
  countRevY = countY / stepForRevY;  // salva il valore di giri dopo l' avanzamento di uno step
  message = message + String(countRevY) + ',';
  countRevZ = (countZ + (countRevX * traslationZ_axis_X) + (countRevY * traslationZ_axis_Y)) / stepForRevZ;  // salva il valore di giri dopo l' avanzamento di uno step
  message = message + String(countRevZ) + ',' + "ciao";
  if(countRevX != countRevX_old || countRevY != countRevY_old || countRevZ != countRevZ_old)
  {
    Serial.println(message);
  }
  countRevX_old = countRevX;
  countRevY_old = countRevY;
```

```
  countRevZ_old = countRevZ;
  delayMicroseconds(dpause);  // pausa di "dpause" microsecondi
  digitalWrite(stepXPin, LOW);  // abbasssa il valore LOW l' uscita allo stepXPin
  digitalWrite(stepYPin, LOW);  // abbasssa il valore LOW l' uscita allo stepYPin
  digitalWrite(stepZPin, LOW);  // abbasssa il valore LOW l' uscita allo stepZPin
  delayMicroseconds(dpause);  // pausa di "dpause" microsecondi


  /* funzionamento automatico premendo il pulsante
   *  In questa modalità il programma prende l' array ogni due elementi per volta. Nel primo elemento viene comunicato il motore da azionare e il secondo indica il numero di
   *  giri da effettuare, tenendo conto del segno con cui vengono richiesti poichè per convenzione poniamo rotazione oraria per valori positivi e rotazione antioraria
   *  per valori negativi
   */
  if(digitalRead(buttonProgramPin) == HIGH)  // verifica se viene premuto il pulsante
  {
    int switchPin, switchDxPin, switchSxPin, numRev;
    switchPin = switchXDxPin;
    int value = 0;
    // ciclo che esegue il percorso automatico, prendendo istruzione su quale motore da attivare e sul numero di giri dall' array, eseguendo tutti i valori al suo interno
    for(int i = 0; i < (sizeof(program2)/sizeof(int)) && digitalRead(switchPin) == HIGH && value == 0; i = i + 2)
    {
      // impostazioni iniziali: salvataggio degli switch fine corsa e numero di giri da eseguire in base al motore da far muovere program[i]
      switch(program2[i])
      {
        case MOTOR_X:
          switchDxPin = switchXDxPin;
          switchSxPin = switchXSxPin;
          numRev = program2[i + 1] - countRevX;  // salva il numero (program[i + 1]) di giri da far eseguire al motore MOTOR_X dato dalla sequenza prestabilita
          break;
        case MOTOR_Y:
          switchDxPin = switchYDxPin;
          switchSxPin = switchYSxPin;
          numRev = program2[i + 1] - countRevY;  // salva il numero (program[i + 1]) di giri da far eseguire al motore MOTOR_X dato dalla sequenza prestabilita
          break;
        case MOTOR_Z:
          switchDxPin = switchZDxPin;
          switchSxPin = switchZSxPin;
          numRev = program2[i + 1] - countRevZ;
          break;
      }

      // controlla il segno del valore i-esimo dell'array
      if(numRev > 0)  // se il valore i-esimo è positivo, imposta una rotazione oraria(dirLevel = HIGH) e lo switch fine corsa quello destro, ovvero quello che può incontrare
      {
```

```
        dirLevel = HIGH;
        switchPin = switchDxPin;
      }
      else  // se il valore i-esimo è negativo, imposta una rotazione antioraria(dirLevel = LOW) e lo switch fine corsa quello sinistro, ovvero quello che può incontrare
      {
        dirLevel = LOW;
        switchPin = switchSxPin;
      }
      startGo(dirLevel, abs(numRev), program2[i]);  // richiama la funzione startGo per eseguire il numero di giri prestabilito
      updateCounter();  // stampa sul display le posizioni quando ogni motore ha eseguito il numero di giri prestabilito
      delay(5000);  // effettua una pausa di 5000 ms, ovvero 5 s, prima di passare al valore successivo, cioè all' i-esimo + 2

      if(program2[i + 2] == 0)
        value = 1;
    }
    for(int i = 0; i < (sizeof(program2)/sizeof(int)); i++)
    {
      program2[i] = 0;
    }
    count = 0;
  }

  // premuto il primo tasto si identifica l'origine delle tre coordinate (0, 0, 0)

  if(digitalRead(buttonOnePin) == HIGH)  // verifica se viene premuto il pulsante
  {
    countRevX = 0;
    countRevY = 0;
    countRevZ = 0;
    countX = 0;
    countY = 0;
    countZ = 0;
  }

  // premuto il secondo tasto si identifica il punto (200, 0, 0)

  if(digitalRead(buttonTwoPin) == HIGH)  // verifica se viene premuto il pulsante
  {
    stepForRevX = countX/200;  // sapendo che sono a 200 mm dall' origine divido il numero di step eseguiti per 200 trovando un nuovo valore di step necessari ad un giro in X
    delay(1000);
    countY = 0;
    countZ = 0;
    countRevY = 0;
    countRevZ = 0;
    // calcola la traslazione lungo l' asse Z dovuta all' inclinazione dell' asse X, ovvero il numero di step in verticale ad ogni giro in X
    traslationZ_axis_X = sqrt((800 * 800) - (stepForRevX * stepForRevX));
```

```
    delay(1000);
  }

  // premuto il terzo tasto si identifica il punto (0, 200, 0)

  if(digitalRead(buttonThreePin) == HIGH)  // verifica se viene premuto il pulsante
  {
    countX = 0;
    stepForRevY = countY/200;  // sapendo che sono a 200 mm dall' origine divido il numero di step
eseguiti per 200 trovando un nuovo valore di step necessari ad un giro in Y
    delay(1000);
    countZ = 0;
    // calcola la traslazione lungo l'asse Z dovuta all' inclinazione dell'asse Y, ovvero il numero di
step in verticale ad ogni giro in Y
    traslationZ_axis_Y = sqrt((800 * 800) - (stepForRevY * stepForRevY));
    delay(1000);
    countRevX = 0;
    countRevZ = 0;
  }

  if (stringComplete)
  {
    if(inputString.substring(0,1) == "+")
      program2[count] = 1;
    else if(inputString.substring(0,1) == "-")
      program2[count] = -1;
    program2[count] = program2[count] * inputString.substring(1,4).toInt();
    count++;
    // pulisce la stringa
    inputString = "";
    stringComplete = false;
  }

}


void serialEvent() {
  while (Serial.available() > 0) {
    // riceve un nuovo byte
    char inChar = (char)Serial.read();

    // aggiunge il byte ricevuto ad inputString
    inputString += inChar;
    // se il carattere entrante è “a capo”, viene settato un flag
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}

// funzione utilizzata per leggere lo stato del rotary encoder, restituendo +1 o -1 in base al senso in
cui viene fatto ruotare la rotella del rotary encoder
```

```
int getEncoderTurn(void)
{
  static int oldA = HIGH;
  static int oldB = HIGH;
  int result = 0;
  int newA = digitalRead(clkPin);
  int newB = digitalRead(dtPin);
  if(newA != oldA || newB != oldB)
  {
    if(oldA == HIGH && newA == LOW)
    {
      result = (oldB * 2 - 1);
    }
  }
  oldA = newA;
  oldB = newB;
  return result;
}

// funzioni utilizzate per leggere lo stato del joystick, restitundo 3,2,1,0,-1,-2,-3 in base alla
posizione del joystick

int getJoystickValue(int drPin)
{
  int joystickValue;
  int analogVal = analogRead(drPin);
  if(analogVal >= 0 && analogVal <= 20)
    joystickValue = -3;
  else if(analogVal > 20 && analogVal <= 250)
    joystickValue = -2;
  else if(analogVal > 250 && analogVal <= 400)
    joystickValue = -1;
  else if(analogVal > 400 && analogVal <= 600)
    joystickValue = 0;
  else if(analogVal > 600 && analogVal <= 750)
    joystickValue = 1;
  else if(analogVal > 750 && analogVal <= 850)
    joystickValue = 2;
  else if(analogVal > 850 && analogVal <= 1023)
    joystickValue = 3;
  return joystickValue;
}

// funzione utilizzata per eseguire una rotazione del motore (motor) inserendo il numero di giri
(numRev) e il senso di rotazione (dirLevel)

void startGo(int dirLevel, int numRev, int motor)
{
  int dir, M0, M1, M2, stepPin, switchPin, switchDxPin, switchSxPin, stepForRev, enable;
  // impostazioni iniziali: per ogni motore vengono salvati i valori dei vari pin del driver per pilotare
il motore in questione
```

```
 switch(motor)
  {
    case MOTOR_X:
      dir = dirXPin;
      M0 = M0XPin;
      M1 = M1XPin;
      M2 = M2XPin;
      stepPin = stepXPin;
      switchDxPin = switchXDxPin;
      switchSxPin = switchXSxPin;
      stepForRev = stepForRevX;
      enable = enableXPin;
      break;
    case MOTOR_Y:
      dir = dirYPin;
      M0 = M0YPin;
      M1 = M1YPin;
      M2 = M2YPin;
      stepPin = stepYPin;
      switchDxPin = switchYDxPin;
      switchSxPin = switchYSxPin;
      stepForRev = stepForRevY;
      enable = enableYPin;
      break;
    case MOTOR_Z:
      dir = dirZPin;
      M0 = M0ZPin;
      M1 = M1ZPin;
      M2 = M2ZPin;
      stepPin = stepZPin;
      switchDxPin = switchZDxPin;
      switchSxPin = switchZSxPin;
      stepForRev = stepForRevZ;
      enable = enableZPin;
      break;
  }
  // imposta una risoluzione di 200 step per giro (FULL-STEP)
  digitalWrite(M0, LOW);
  digitalWrite(M1, LOW);
  digitalWrite(M2, LOW);
  digitalWrite(enable, LOW);
  // calcola il numero di step da eseguire per effettuare il numero di giri(numRev) * step ncessari per
ogni giro (diviso 4 perchè siamo in modalità FULL-STEP)
  int numStep = numRev * stepForRev / 4;

  if(dirLevel == HIGH)  // se abbiamo una rotazione oraria, salviamo in switchPin il numero della
porta dello switch di destra
    switchPin = switchDxPin;
  else  // se abbiamo una rotazione antioraria, salviamo in switchPin il numero della porta dello
switch di sinistra
    switchPin = switchSxPin;
```

```
  digitalWrite(dir, dirLevel);  // inviamo al driver del motore il senso di rotazione che vogliamo (oraria == HIGH, antioraria == LOW)

  // ciclo che genera l'onda quadra da inviare al driver del motore tramite il pin stepPin con durata proporzionale al numero di giri scelti
  // e controllando ad ogni step se viene raggiunto lo switch finecorsa che arresta la rotazione
  for(int i = 1; i <= numStep && digitalRead(switchPin) == HIGH; i++)  // && digitalRead(switchPin) == HIGH (da aggiungere come condizione)
  {
    // genero un impulso dell'onda quadra
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(400);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(400);
    if(dirLevel == HIGH)  // se abbiamo una rotazione oraria, aggiorniamo il conteggio degli step aggiungendo 4 perchè siamo in FULL-STEP
    {
      switch(motor)
      {
        case MOTOR_X:
          countX = countX + 4;
          break;
        case MOTOR_Y:
          countY = countY + 4;
          break;
        case MOTOR_Z:
          countZ = countZ + 4;
          break;
      }
    }
    else  // se abbiamo una rotazione antioraria, aggiorniamo il conteggio degli step sottraendo 4 perchè siamo in FULL-STEP
    {
      switch(motor)
      {
        case MOTOR_X:
          countX = countX - 4;
          break;
        case MOTOR_Y:
          countY = countY - 4;
          break;
        case MOTOR_Z:
          countZ = countZ - 4;
          break;
      }
    }
    countRevX = countX / stepForRevX;  // salva il valore di giri dopo l' avanzamento di uno step
    countRevY = countY / stepForRevY;  // salva il valore di giri dopo l' avanzamento di uno step
    countRevZ = (countZ + (countRevX * traslationZ_axis_X) + (countRevY * traslationZ_axis_Y)) / stepForRevZ;  // salva il valore di giri dopo l' avanzamento di uno step
    if(countRevX != countRevX_old || countRevY != countRevY_old || countRevZ != countRevZ_old)
```

```
    {
      Serial.println(String(countRevX) + ',' + String(countRevY) + ',' + String(countRevZ) + ',' +
"ciao");
    }
    countRevX_old = countRevX;
    countRevY_old = countRevY;
    countRevZ_old = countRevZ;
  }
  digitalWrite(enable, HIGH);
}

// funzione utilizzata per il controllo manuale del motore tramite joystick o rotary encoder
// in ingresso ha il senso di rotazione (dirLevel), un valore compreso tra -3 e 3 (value) che identifica
la velocità e il numero di step per giro
// e infine il numero che identifica il motore da azionare

void speedMode(int dirLevel, int value, int i)
{
  int dir, M0, M1, M2;
  switch(i)  // impostazioni iniziali: salvataggio dei valori dei pin inerenti al motore i
  {
    case MOTOR_X:
      dir = dirXPin;
      M0 = M0XPin;
      M1 = M1XPin;
      M2 = M2XPin;
      break;
    case MOTOR_Y:
      dir = dirYPin;
      M0 = M0YPin;
      M1 = M1YPin;
      M2 = M2YPin;
      break;
    case MOTOR_Z:
      dir = dirZPin;
      M0 = M0ZPin;
      M1 = M1ZPin;
      M2 = M2ZPin;
      break;
  }
  if(value == 0)  // se il valore del rotary encoder o del joystick è 0, non invia alcuna onda quadra al
driver fermando il motore
  {
    switch(i)  // impostazioni iniziali: salvataggio dei valori dei pin inerenti al motore i
    {
      case MOTOR_X:
        flagX = 0;
        break;
      case MOTOR_Y:
        flagY = 0;
        break;
      case MOTOR_Z:
```

```
      flagZ = 0;
      break;
    }
  }
  else  // se il valore del rotary encoder è diverso da 0, cambia  velocità in base ad esso sfruttando il microstepping
  {
    switch(value)
    {
     case 1:
      dirLevel = HIGH;  // rotazione oraria
      // impostazione per un quarto di step, ovvero 800 step/giro
      digitalWrite(M0, LOW);
      digitalWrite(M1, HIGH);
      digitalWrite(M2, LOW);
      // aggiorna la variabile count, tenendo conto di aver fatto un microstep ovvero 1/800 di giro
      switch(i)
      {
       case MOTOR_X:
        countX = countX + 1;
        break;
       case MOTOR_Y:
        countY = countY + 1;
        break;
       case MOTOR_Z:
        countZ = countZ + 1;
        break;
      }
      flagX = 0;
      flagY = 0;
      flagZ = 0;
      break;
     case 2:
      dirLevel = HIGH;  // rotazione oraria
      // impostazione per un mezzo di step, ovvero 400 step/giro
      digitalWrite(M0, HIGH);
      digitalWrite(M1, LOW);
      digitalWrite(M2, LOW);
      // aggiorna la variabile count, tenendo conto di aver fatto un microstep ovvero 2/800 di giro (1/400)
      switch(i)
      {
       case MOTOR_X:
        countX = countX + 2;
        break;
       case MOTOR_Y:
        countY = countY + 2;
        break;
       case MOTOR_Z:
        countZ = countZ + 2;
        break;
      }
```

```
  flagX = 0;
  flagY = 0;
  flagZ = 0;
  break;
 case 3:
  dirLevel = HIGH;  // rotazione oraria
  // impostazione per uno step, ovvero 200 step/giro
  digitalWrite(M0, LOW);
  digitalWrite(M1, LOW);
  digitalWrite(M2, LOW);
  // aggiorna la variabile count, tenendo conto di aver fatto uno step ovvero 4/800 di giro (1/200)
  switch(i)
  {
   case MOTOR_X:
    countX = countX + 4;
    flagX++;
    break;
   case MOTOR_Y:
    countY = countY + 4;
    flagY++;
    break;
   case MOTOR_Z:
    countZ = countZ + 4;
    flagZ++;
    break;
  }
  break;
 case -1:
  dirLevel = LOW;  // rotazione antioraria
  // impostazione per un quarto di step, ovvero 800 step/giro
  digitalWrite(M0, LOW);
  digitalWrite(M1, HIGH);
  digitalWrite(M2, LOW);
  // aggiorna la variabile count, tenendo conto di aver fatto un microstep ovvero -1/800 di giro
  switch(i)
  {
   case MOTOR_X:
    countX = countX - 1;
    break;
   case MOTOR_Y:
    countY = countY - 1;
    break;
   case MOTOR_Z:
    countZ = countZ - 1;
    break;
  }
  flagX = 0;
  flagY = 0;
  flagZ = 0;
  break;
 case -2:
  dirLevel = LOW;  // rotazione antioraria
```

```
      // impostazione per un mezzo di step, ovvero 400 step/giro
      digitalWrite(M0, HIGH);
      digitalWrite(M1, LOW);
      digitalWrite(M2, LOW);
      // aggiorna la variabile count, tenendo conto di aver fatto un microstep ovvero -2/800 di giro (-
1/400)
      switch(i)
      {
        case MOTOR_X:
          countX = countX - 2;
          break;
        case MOTOR_Y:
          countY = countY - 2;
          break;
        case MOTOR_Z:
          countZ = countZ - 2;
          break;
      }
      flagX = 0;
      flagY = 0;
      flagZ = 0;
      break;
    case -3:
      dirLevel = LOW;  // rotazione antioraria
      // impostazione per uno step, ovvero 200 step/giro
      digitalWrite(M0, LOW);
      digitalWrite(M1, LOW);
      digitalWrite(M2, LOW);
      // aggiorna la variabile count, tenendo conto di aver fatto uno step ovvero -4/800 di giro (-
1/200)
      switch(i)
      {
        case MOTOR_X:
          countX = countX - 4;
          flagX++;
          break;
        case MOTOR_Y:
          countY = countY - 4;
          flagY++;
          break;
        case MOTOR_Z:
          countZ = countZ - 4;
          flagZ++;
          break;
      }
      break;
    }
   digitalWrite(dir, dirLevel);  // imposta il senso di rotazione del motore
  }
}

// funzione per il conteggio giri
```

```
// cambiando il denominatore possiamo contare non solo i giri, ma anche frazioni di giro senza interferire con il movimento del motore
// in questo caso con 800 contiamo un giro in quanto con la velocità minima effettuiamo 800 steps per un giro

void updateCounter(void)
{
  // stampa il segno "+" o "-" in base al numero di giri salvati se positivi o negativi
  if(countRevX >= 0)
  {
    lcd.setCursor(0, 1);
    lcd.print("+");
  }
  else if(countRevX < 0)
  {
    lcd.setCursor(0, 1);
    lcd.print("-");
  }

  if(countRevY >= 0)
  {
    lcd.setCursor(6, 1);
    lcd.print("+");
  }
  else if(countRevY < 0)
  {
    lcd.setCursor(6, 1);
    lcd.print("-");
  }

  if(countRevZ >= 0)
  {
    lcd.setCursor(12, 1);
    lcd.print("+");
  }
  else if(countRevZ < 0)
  {
    lcd.setCursor(12, 1);
    lcd.print("-");
  }

  // metodo di conteggio a fermata del motore
  // ogni volta che i motori si fermano, viene richiamata questa funzione che stampa nello schermo lcd i valori delle posizioni di avanzamento di tutti e 3 i motori

  lcd.setCursor(1, 1);
  if(abs(countRevX) >= 10 && abs(countRevX) / 100 == 0 && abs(countRevX) / 1000 == 0)
    lcd.print("0");
  else if(abs(countRevX) >= 0 && abs(countRevX) / 10 == 0 && abs(countRevX) / 100 == 0 && abs(countRevX) / 1000 == 0)
    lcd.print("00");
  lcd.print(abs(countRevX));
```

```
  lcd.setCursor(7, 1);
  if(abs(countRevY) >= 10 && abs(countRevY) / 100 == 0 && abs(countRevY) / 1000 == 0)
    lcd.print("0");
  else if(abs(countRevY) >= 0 && abs(countRevY) / 10 == 0 && abs(countRevY) / 100 == 0 &&
abs(countRevY) / 1000 == 0)
    lcd.print("00");
  lcd.print(abs(countRevY));

  lcd.setCursor(13, 1);
  if(abs(countRevZ) >= 10 && abs(countRevZ) / 100 == 0 && abs(countRevZ) / 1000 == 0)
    lcd.print("0");
  else if(abs(countRevZ) >= 0 && abs(countRevZ) / 10 == 0 && abs(countRevZ) / 100 == 0 &&
abs(countRevZ) / 1000 == 0)
    lcd.print("00");
  lcd.print(abs(countRevZ));
  }

// funzione per la mappatura dei fori
// Conoscendo la coordinata del primo foro (firstHoleX, firstHoleY, 0) e il passo dei fori lungo le
due direzioni (distanceHoleX per l'asse X e distanceHoleY per l'asse Y)
// la funzione individua la posizione di ogni foro.
// La funzione restituisce HIGH se la posizione (x, y) in cui si trova è sopra un quadrato di
tolleranza interno ed inferiore ad ogni foro. Al contrario, restituisce LOW
// se la posizione non è interna a questi quadrati di tolleranza.
// La tolleranza attorno al centro del foro è calcolata in funzione alla dimensione del foro e della
sonda in modo tale che durante il funzionamento automatico non ci siano disguidi
// nell' inserimento della sonda attraverso la griglia.

int mappingHole(void)
{
  int ctrl = 0;
  int n = (countRevX - firstHoleX) / distanceHoleX;
  int r = (countRevX - firstHoleX) % distanceHoleX;
  if(r > 10 && r <= 20)
    n++;

  n = n * distanceHoleX;
  if((countRevX - firstHoleX) <= n + tollerance && (countRevX - firstHoleX) >= n - tollerance)
    ctrl = ctrl + 1;
  else
    ctrl = 0;

  n = (countRevY - firstHoleY) / distanceHoleY;
  r = (countRevY - firstHoleY) % distanceHoleY;
  if(r > 10 && r <= 20)
    n++;

  n = n * distanceHoleY;
  if((countRevY - firstHoleY) <= n + tollerance && (countRevY - firstHoleY) >= n - tollerance)
    ctrl = ctrl + 1;
  else
    ctrl = 0;
```

```
  if(ctrl == 2)
    return HIGH;
  else
    return LOW;
}
```

# APPENDICE B

```
import processing.serial.*;

// altezza del sensore quando è in movimento
// parametro composto da 4 caratteri, il primo indica il segno mentre dal secondo al quarto viene scritto il numero composto obligatoriamente da 3 cifre per essere interpretato correttamente da Arduino
String heightSensorMove = "+050";

// inizializzazione variabili per la costruzione della griglia
int positionGridX, positionGridY, widthGrid, heightGrid, distanceHoleX, distanceHoleY, distanceFirstHoleX, distanceFirstHoleY, nRaw, nColumn, diameterHoleX, diameterHoleY;
int diameterProbe;
// inizializzazione variabili per i profili della griglia
int lengthLineWidth, lengthLineHeight;
int ciao, ciao2, ciao3;
// inizializzazione variabili tabelle
int widthBox, heightBox;
int widthTable1, heightTable1, nRawTable1, nColumnTable1, positionX_Table1, positionY_Table1;
int widthTable2, heightTable2, nRawTable2, nColumnTable2, positionX_Table2, positionY_Table2;
int widthTable3, heightTable3, nRawTable3, nColumnTable3, positionX_Table3, positionY_Table3;
int widthTable4, heightTable4, nRawTable4, nColumnTable4, positionX_Table4, positionY_Table4;
int widthTable5, heightTable5, nRawTable5, nColumnTable5, positionX_Table5, positionY_Table5;
int widthTable6, heightTable6, nRawTable6, nColumnTable6, positionX_Table6, positionY_Table6;

// inizializzazioni variabili per rettangoli contenenti le coordinate x, y, z
int widthRect, heightRect;
int posizionX_rectX, posizionX_rectY, posizionX_rectZ;
int positionY_rectX, positionY_rectY, positionY_rectZ;

// inizializzazione variabili dimensione pulsanti
int widthButton, heightButton;

// inizializzazione variabili per il percorso
int track, index;

// inizializzazione stringhe che indicano al programma alcune variabili come le dimensioni della griglia
String dimensionGridX = "600";
String dimensionGridY = "300";
String diamHole = "7";
String distHoleX = "21";
String distHoleY = "19";
String distFirstHoleX = "3";
String distFirstHoleY = "3";
String nLineX = "11";
String nLineY = "5";
String angle = "0";
String lengthLine = "";
String nStrutX = "5";
```

```
String nStrutY = "1";
String distStrut = "0";
String widthStrut = "0";

String positionX = "";
String positionY = "";
String positionZ = "";

// inizializzzazione colori schermata
int button_Off = color(250, 63, 53);
int button_On = color(53, 250, 74);
int background = color(229, 209, 95);
int table1_Off = color(47, 168, 222);
int table1_On = color(47, 168, 150);

// inizializzazone array delle tabelle e array pulsanti
int[] button = new int[5];
int[] elementTable1 = new int[21];
int elementTable2[] = new int[60];
int elementTable3[] = new int[60];
int elementTable4[] = new int[60];
int elementTable5[] = new int[60];
int elementTable6[] = new int[60];
String elementMainTable[] = new String[21];
String nLineElementX[] = new String[60];
String nLineElementY[] = new String[60];
String strutElementX[] = new String[60];
String strutElementY[] = new String[60];
int trackPosition[] = new int[60];

// inizializzazione variabili per il seriale
Serial myPort;

String positionSensorX = "0";
String positionSensorY = "0";
String positionSensorZ = "0";
String serial = null;

/* Funzione setup
  Questa funzione viene eseguita una sola volta al momento dell'avvio del programma.
  Funzione in cui si impostano i valori iniziali come il riempimento dei vari array con valori nulli e
alcune misure di default come le dimensioni
  della griglia, fori ecc.
  Altra impostazione iniziale che viene eseguita è la dimensione della finestra in cui viene stampata
l'interfaccia con l'utente.
  Inoltre in questa funzione si indica aal programma su quale porta seriale comunicare con Arduino
e quale sia il carattere che indica la fine del messaggio da leggere
  o da inviare.

*/
void setup()
{
```

```
  size(1200, 700);  // impostazione dimensione finestra in cui verrà visualizzata l'interfaccia, nello specifico 1200 * 700 pixel
  // riempimento di tutti gli array globali con valore nullo
  for(int i = 0; i < button.length; i++)
    button[i] = 0;
  for(int i = 0; i < elementTable1.length; i++)
    elementTable1[i] = 0;
  for(int i = 0; i < elementTable2.length; i++)
    elementTable2[i] = 0;
  for(int i = 0; i < elementTable3.length; i++)
    elementTable3[i] = 0;
  for(int i = 0; i < elementTable4.length; i++)
    elementTable4[i] = 0;
  for(int i = 0; i < elementTable5.length; i++)
    elementTable5[i] = 0;
  for(int i = 0; i < elementTable6.length; i++)
    elementTable6[i] = 0;
  for(int i = 0; i < elementMainTable.length; i++)
    elementMainTable[i] = "0";
  // riempimento di alcuni elementi dell'array elementMainTable, ovvero dei valori impostati di default nella tabella principale
  elementMainTable[4] = "800";  // parametro inerente la larghezza lungo X della griglia
  elementMainTable[5] = "300";  // parametro inerente l'altezza lungo Y della griglia
  elementMainTable[7] = "14";  // parametro inerente al diametro lungo X dei fori
  elementMainTable[8] = "14";  // parametro inerente al diametro lungo Y dei fori
  elementMainTable[10] = "21";  // parametro inerente al passo dei fori lungo X
  elementMainTable[11] = "19";  // parametro inerente al passo dei fori lungo Y
  elementMainTable[13] = "21";  // parametro inerente alla distanza lungo X del primo foro
  elementMainTable[14] = "19";  // parametro inerente alla distanza lungo Y del primo foro
  elementMainTable[16] = "1";  // parametro inerente al numero di tratti con diversa inclinazione lungo X
  elementMainTable[17] = "1";  // parametro inerente al numero di tratti con diversa inclinazione lungo Y
  // riempimento di tutti gli array globali con valore nullo
  for(int i = 0; i < nLineElementX.length; i++)
    nLineElementX[i] = "0";
  nLineElementX[4] = elementMainTable[4];  // si imposta di default il valore alla seconda riga e seconda colonna della tabella il valore della tabella principale
  for(int i = 0; i < nLineElementY.length; i++)
    nLineElementY[i] = "0";
  nLineElementY[4] = elementMainTable[5];  // si imposta di default il valore alla seconda riga e seconda colonna della tabella il valore della tabella principale
  for(int i = 0; i < strutElementX.length; i++)
    strutElementX[i] = "0";
  for(int i = 0; i < strutElementY.length; i++)
    strutElementY[i] = "0";
  for(int i = 0; i < trackPosition.length; i++)
    trackPosition[i] = 0;
  index = 5;  // si imposta il valore index a 5 poichè è la posizione dell'array in cui va salvato la coordinata X al primo click interno alla griglia
  track = 0;  // si imposta il valore track a 0 poichè è il numero che indica le posizioni cliccate sulla griglia
```

```
  diameterProbe = 6;  // valore del diametro della sonda per calcolare la tolleranza all'interno dei fori per indicare se può entrare oppure no nel foro

  // inizializzazione porta seriale per la comunicazione con Arduino
  myPort = new Serial(this, "/dev/ttyACM0", 115200);  // si indica il nome della porta /dev/ttyACM0 e il baudrate 115200 che dev'essere uguale  con quelo impostato in Arduino
  myPort.bufferUntil('\n');  // si comunica quale sia il carattere che segna la fine di ogni messaggio
}

/* Funzione Draw
  Funzione che viene eseguita ad ogni ciclo del programma attraverso la quale si può disegnare e scrivere nell'interfaccia in questione.
  Ogni modifica che si visualizza nella finestra creata è opera di questa funzione che attraverso funzioni predefinite come rect(), ellipse(), ecc. si possono
  rappresentare nello schermo figure semplici come rettangoli, cerchi ecc. Oltre a quetse forme geometriche basilari grazie a quetsa funzione si possono raffigurare scritte,
  linee e colori.
  Si possono impostare i colori che si vogliono per colorare le figure al loro interno, per il bordo, per le scritte, per sfondo e quindi tutta la finestra dell'interfaccia.
  La scelta del colore si effettua attaverso quattro parametri che vanno da 0 a 255 e hanno un funzionamento simile ad un RGB. Infatti il primo parametro indica l'intensità
  del rosso, il secondo del verde, il terzo del blu e il quarto la sua trasparenza (quest'ultimo si può omettere tenendo una trasparenza pari a 0 e quindi assente).
*/

void draw()
{
  background(background);  // impostazione del colore dello sfondo della finestra

  positionGridX = 100;  // parametro che indica la coordinata X in pixel da dove inizia la griglia
  positionGridY = 100;  // parametro che indica la coordinata Y in pixel da dove inizia la griglia

  // se la casella (2, 2) della tabella principale è di colore chiaro leggi e salva in widthGrid la larghezza della griglia
  if(elementTable1[4] == 0)
    widthGrid = Integer.parseInt(elementMainTable[4]);
  // se la casella (2, 3) della tabella principale è di colore chiaro leggi e salva in heightGrid l'altezza della griglia
  if(elementTable1[5] == 0)
    heightGrid = Integer.parseInt(elementMainTable[5]);
  // se la casella (3, 2) della tabella principale è di colore chiaro leggi e salva in diameterHoleX il diametro lungo X dei fori
  if(elementTable1[7] == 0)
    diameterHoleX = Integer.parseInt(elementMainTable[7]);
  // se la casella (3, 3) della tabella principale è di colore chiaro leggi e salva in diameterHoleY il diametro lungo Y dei fori
  if(elementTable1[8] == 0)
    diameterHoleY = Integer.parseInt(elementMainTable[8]);
  // se la casella (4, 2) della tabella principale è di colore chiaro leggi e salva in distanceHoleX il passo dei fori lungo X
  if(elementTable1[10] == 0)
    distanceHoleX = Integer.parseInt(elementMainTable[10]);
```

```
  // se la casella (4, 3) della tabella principale è di colore chiaro leggi e salva in distanceHoleY il passo dei fori lungo Y
  if(elementTable1[11] == 0)
    distanceHoleY = Integer.parseInt(elementMainTable[11]);
  nRaw = heightGrid / distanceHoleY - 1;  // calcola il numero delle righe dei fori
  nColumn = widthGrid / distanceHoleX - 1;  // calcola il numero delle colonne dei fori
  // se la casella (5, 2) della tabella principale è di colore chiaro leggi e salva in distanceHoleX la distanza lungo X del primo foro dall'origine
  if(elementTable1[13] == 0)
    distanceFirstHoleX = Integer.parseInt(elementMainTable[13]);
  // se la casella (5, 3) della tabella principale è di colore chiaro leggi e salva in distanceHoleY la distanza lungo Y del primo foro dall'origine
  if(elementTable1[14] == 0)
    distanceFirstHoleY = Integer.parseInt(elementMainTable[14]);

  fill(0);  // imposta il riempimeno di colore nero
  // disegna la legenda degli assi X e Y in basso a sinistra della griglia per orientare i due assi lungo la griglia
  text("X", positionGridX + 20, positionGridY + heightGrid + 25);
  text("Y", positionGridX - 25, positionGridY + heightGrid - 20);
  line(positionGridX - 10, positionGridY + heightGrid + 10, positionGridX + 20, positionGridY + heightGrid + 10);
  line(positionGridX - 10, positionGridY + heightGrid + 10, positionGridX - 10, positionGridY + heightGrid - 20);

  fill(150);  // imposta il riempimeno di colore grigio
  rect(positionGridX, positionGridY, widthGrid, heightGrid);  // disegna il rettangolo rappresentante la griglia
  // ciclo innestato per disegnare in tutte le nRaw righe e le nColumn colonne un cerchio rappresentante un foro
  for(int i = 0; i < nRaw; i++)
  {
    for(int j = 0; j < nColumn; j++)
    {
      fill(150);  // imposta il riempimeno di colore grigio per i fori
      // disegna un cerchio rappresentante il foro con le dimensioni impostate dalla griglia principale sotto le voci diametro dei fori in X e in Y
      ellipse(positionGridX + distanceFirstHoleX + j * distanceHoleX, positionGridY + heightGrid - distanceFirstHoleY - i * distanceHoleY, diameterHoleX, diameterHoleY);
      fill(background);  // imposta il riempimeno di colore uguale allo sfondo per le tolleranze interne ai fori
      // disegna un cerchio interno ai fori ad indicare la tolleranza in cui può entrare la sonda nei fori, hanno un diametro calcolato facendo la differenza tra il diametro del foro e della sonda
      ellipse(positionGridX + distanceFirstHoleX + j * distanceHoleX, positionGridY + heightGrid - distanceFirstHoleY - i * distanceHoleY, diameterHoleX - diameterProbe - 1, diameterHoleY - diameterProbe - 1);
    }
  }

  // Sezione del programma in cui si disegnano i profili della griglia sopra e a sinistra di quest'ultima
  fill(150);  // imposta il riempimeno di colore grigio
```

```
  // disegna il profilo in X della griglia controllando da quanti tratti è composta e con quale inclinazione
  if(elementTable1[16] == 0 && Integer.parseInt(elementMainTable[16]) > 0)  // se la casella (6, 2) della tabella principale è di colore chiaro e se all' interno c'è un valore positivo
  {
    float posX = positionGridX;
    float posY = positionGridY - 50;
    // ciclo che legge ogni parametro inserito nella tabella associata al pulsante a destra della casella inerente al numero di tratti in X
    // si parte da g = 4, poichè è il numero della posizione della casella prima riga e seconda colonna all'interno dell'array in cui vengono salvati i vari parametri della tabella
    // l'incremento di g è di 3 poichè salta alla riga successiva della tabella in cui è posto l'elemento dell'array utile per il disegno della linea
    for(int g = 4; g <= 3 * (Integer.parseInt(elementMainTable[16]) + 1) + 2; g += 3)
    {
      if(elementTable2[g] == 0 && elementTable2[g + 1] == 0)
      {
        float angle = - 1 * Float.parseFloat(nLineElementX[g + 1]) * 3.14 / 180.0;  // viene salvata in angle il valore convertito in radianti dell'angolo di inclinazione del tratto i-esimo
        // viene disegnato il segmento i-esimo di lunghezza e inclinazione dati dalla lettura dei dati interni alla tabella associata
        line(posX, posY, posX + Float.parseFloat(nLineElementX[g]), posY + Float.parseFloat(nLineElementX[g]) * sin(angle));
        posX = posX + Float.parseFloat(nLineElementX[g]);  // viene aggiornata la variabile posX per partire al prossimo ciclo dal punto di fine del segmento i-esimo
        posY = posY + Float.parseFloat(nLineElementX[g]) * sin(angle);  // viene aggiornata la variabile posX per partire al prossimo ciclo dal punto di fine del segmento i-esimo
      }
    }
  }

  // disegna il profilo in Y della griglia controllando da quanti tratti è composta e con quale inclinazione
  if(elementTable1[17] == 0 && Integer.parseInt(elementMainTable[17]) > 0)  // se la casella (6, 3) della tabella principale è di colore chiaro e se all' interno c'è un valore positivo
  {
    float posX = positionGridX + widthGrid + 50;
    float posY = positionGridY + heightGrid;
    // ciclo che legge ogni parametro inserito nella tabella associata al pulsante a destra della casella inerente al numero di tratti in Y
    // si parte da g = 4, poichè è il numero della posizione della casella prima riga e seconda colonna all'interno dell'array in cui vengono salvati i vari parametri della tabella
    // l'incremento di g è di 3 poichè salta alla riga successiva della tabella in cui è posto l'elemento dell'array utile per il disegno della linea
    for(int g = 4; g <= 3 * (Integer.parseInt(elementMainTable[17]) + 1) + 2; g += 3)
    {
      if(elementTable3[g] == 0 && elementTable3[g + 1] == 0)
      {
        float angle = - 1 * Float.parseFloat(nLineElementY[g + 1]) * 3.14 / 180.0;  // viene salvata in angle il valore convertito in radianti dell'angolo di inclinazione del tratto i-esimo
        // viene disegnato il segmento i-esimo di lunghezza e inclinazione dati dalla lettura dei dati interni alla tabella associata
```

```
      line(posX, posY, posX + Float.parseFloat(nLineElementY[g]) * sin(angle), posY -
Float.parseFloat(nLineElementY[g]));
      posX = posX + Float.parseFloat(nLineElementY[g]) * sin(angle);  // viene aggiornata la
variabile posX per partire al prossimo ciclo dal punto di fine del segmento i-esimo
      posY = posY - Float.parseFloat(nLineElementY[g]);  // viene aggiornata la variabile posX per
partire al prossimo ciclo dal punto di fine del segmento i-esimo
     }
    }
  }

  // disegna i montanti verticali con distanza X dall'origine degli assi e con uno spessore impostabile
  if(elementTable1[19] == 0 && Integer.parseInt(elementMainTable[19]) > 0)  // se la casella (7, 2)
della tabella principale è di colore chiaro e se all' interno c'è un valore positivo
  {
   for(int g = 4; g <= 3 * (Integer.parseInt(elementMainTable[19]) + 1) + 2; g += 3)
   {
    if(elementTable4[g] == 0 && elementTable4[g + 1] == 0)
    {
     // disegna un rettangolo con inizio structElementX[g] e spessore structElementX[g+1], valori
letti dalla tabella associata al pulsante alla destra della
     // casella inerente al numero dei montanti in X
     rect(positionGridX + Integer.parseInt(strutElementX[g]), positionGridY,
Integer.parseInt(strutElementX[g + 1]), heightGrid);
    }
   }
  }

  // disegna i montanti orizzontali con distanza Y dall'origine degli assi e con uno spessore
impostabile
  if(elementTable1[20] == 0 && Integer.parseInt(elementMainTable[20]) > 0)  // se la casella (7, 3)
della tabella principale è di colore chiaro e se all' interno c'è un valore positivo
  {
   for(int g = 4; g <= 3 * (Integer.parseInt(elementMainTable[20]) + 1) + 2; g += 3)
   {
    if(elementTable5[g] == 0 && elementTable5[g + 1] == 0)
    {
     // disegna un rettangolo con inizio structElementY[g] e spessore structElementY[g+1], valori
letti dalla tabella associata al pulsante alla destra della
     // casella inerente al numero dei montanti in Y
     rect(positionGridX, positionGridY + Integer.parseInt(strutElementY[g]), widthGrid,
Integer.parseInt(strutElementY[g + 1]));
    }
   }
  }

  fill(0,0,255);
  ellipse(20, positionGridY + heightGrid + 35, 7, 7);
  fill(button_Off);
  ellipse(20, positionGridY + heightGrid + 65, 7, 7);
  fill(0);  // imposta il riempimento di colore nero
  // scrive vicino alle caselle dela posizione del cursore all'interno della griglia
  text("Posizione sensore:", 30, positionGridY + heightGrid + 40);
```

```
  text("Posizione cursore:", 30, positionGridY + heightGrid + 70);
  text("X", 155, positionGridY + heightGrid + 40);
  text("Y", 355, positionGridY + heightGrid + 40);
  text("Z", 555, positionGridY + heightGrid + 40);
  text("X", 155, positionGridY + heightGrid + 70);
  text("Y", 355, positionGridY + heightGrid + 70);
  text("Z", 555, positionGridY + heightGrid + 70);
  text("Elenco percorso", 800, positionGridY + heightGrid + 40);
  fill(table1_Off);  // imposta il colore della tabella // disegna le tre caselle in cui si potranno leggere le coordinate del cursore all'interno della griglia
  rect(170, positionGridY + heightGrid + 25, widthBox, heightBox);
  rect(370, positionGridY + heightGrid + 25, widthBox, heightBox);
  rect(570, positionGridY + heightGrid + 25, widthBox, heightBox);
  // disegna le tre caselle in cui si potranno leggere le coordinate del cursore all'interno della griglia
  rect(170, positionGridY + heightGrid + 55, widthBox, heightBox);
  rect(370, positionGridY + heightGrid + 55, widthBox, heightBox);
  rect(570, positionGridY + heightGrid + 55, widthBox, heightBox);

  // verifica se è stato cliccato (colore verde) o no (colore rosso) il pulsante sotto la scritta "elenco percorso:"
  if(button[4] == 0)
    fill(button_Off);
  else
    fill(button_On);
  // disegna il quadrato raffigurante il pulsante sotto la scritta "Elenco percorso:"
  rect(840, positionGridY + heightGrid + 50, heightBox, heightBox);

  // salva nelle variabili positionX e positionY le coordinate X e Y assunte dal cursore spostandosi all'interno della griglia
  positionX = Integer.toString(mouseX - positionGridX);
  positionY = Integer.toString(positionGridY + heightGrid - mouseY);
  // salva in positionZ il valore 0 se il cursore all'interno della griglia è sopra un pixel di colore grigio oppure -20 se il pixel è di colore uguale allo sfondo
  if(get(mouseX, mouseY) == background)
    positionZ = Integer.toString(-20);
  else
    positionZ = Integer.toString(0);


  if(overRect(positionGridX, positionGridY, widthGrid, heightGrid))  // controllo se il cursore è all'interno della griglia
  {
    if(mousePressed && (mouseButton == LEFT))  // se viene premuto il tasto sinistro del mouse
    {
      // salva le coordinate X, Y e Z del punto cliccato nella griglia nell'array trackPosition che tiene in sequenza tutte le posizioni cliccate da inviare ad Arduino
      trackPosition[index] = Integer.parseInt(positionX);
      trackPosition[index + 1] = Integer.parseInt(positionY);
      trackPosition[index + 2] = Integer.parseInt(positionZ);

      int flag = 0;  //variabile utilizzata per uscire dal ciclo
```

```
    int lengthLine = 0;  // variabile che tiene conto della lunghezza dei vari tratti considerati per il calcolo delle coordinate da inviare ad Arduino
    int posX = 0;  // coordinata X da inviare ad Arduino
    int posY = 0;  // coordinata Y da inviare ad Arduino
    int posZ = trackPosition[index + 2];  // coordinata Z da inviare ad Arduino, dove si imposta come valore iniziale il valore letto dalla griglia
    // ciclo utilizzato per leggere tutti gli elementi presenti nell'array nLineElementX[] dove sono salvati i parametri indicanti la lunghezza dei vari tratti con la loro inclinazione
    // si parte da i = 4, poichè è il numero della posizione della casella prima riga e seconda colonna all'interno dell'array in cui vengono salvati i vari parametri della tabella associata
    // l'incremento di i è di 3 poichè salta alla riga successiva della tabella in cui è posto l'elemento dell'array utile per il calcolo della coordinata X
    for(int i = 4; i < nLineElementX.length && flag == 0; i += 3)
    {
     // controllo se la posizione cliccata dalla griglia è interna al tratto i-esimo della tabella associata
      if(trackPosition[index] <= lengthLine + Integer.parseInt(nLineElementX[i]))
      {
       // calcolo la coordinata di X da inviare ad Arduino come il cateto di un triangolo rettangolo avente ipotenusa uguale alla lunghezza del tratto meno la coordinata X e angolo compreso uguale all'angolo di inclinazione del tratto
        posX = posX + Math.round((trackPosition[index] - lengthLine) * cos(Float.parseFloat(nLineElementX[i + 1]) * 3.14 / 180.0));
        // calcolo la coordinata di Z da inviare ad Arduino come il cateto di un triangolo rettangolo avente ipotenusa uguale alla lunghezza del tratto meno la coordinata X e angolo opposto uguale all'angolo di inclinazione del tratto
        posZ = posZ + Math.round((trackPosition[index] - lengthLine) * sin(Float.parseFloat(nLineElementX[i + 1]) * 3.14 / 180.0));
        flag = 1;  // aggiornamento variabile che mi permette di uscire dal ciclo
      }
      else
      {
       // calcolo la coordinata di X da inviare ad Arduino come il cateto di un triangolo rettangolo avente ipotenusa uguale alla lunghezza del tratto e angolo compreso uguale all'angolo di inclinazione del tratto
        posX = posX + Math.round(Float.parseFloat(nLineElementX[i]) * cos(Float.parseFloat(nLineElementX[i + 1]) * 3.14 / 180.0));
        // calcolo la coordinata di Z da inviare ad Arduino come il cateto di un triangolo rettangolo avente ipotenusa uguale alla lunghezza del tratto e angolo opposto uguale all'angolo di inclinazione del tratto
        posZ = posZ + Math.round(Float.parseFloat(nLineElementX[i]) * sin(Float.parseFloat(nLineElementX[i + 1]) * 3.14 / 180.0));
        // aggiornamento della lunghezza lengthLine per il calcolo successivo al tratto i-esimo + 1
        lengthLine = lengthLine + Integer.parseInt(nLineElementX[i]);
      }
    }
    // azzeramento delle variabili flag e lengthLine per l'esecuzione di un ciclo simile per la coordinata Y
    flag = 0;
    lengthLine = 0;
    // ciclo utilizzato per leggere tutti gli elementi presenti nell'array nLineElementY[] dove sono salvati i parametri indicanti la lunghezza dei vari tratti con la loro inclinazione
```

```
    // si parte da i = 4, poichè è il numero della posizione della casella prima riga e seconda colonna all'interno dell'array in cui vengono salvati i vari parametri della tabella associata
    // l'incremento di i è di 3 poichè salta alla riga successiva della tabella in cui è posto l'elemento dell'array utile per il calcolo della coordinata Y
    for(int i = 4; i < nLineElementY.length && flag == 0; i += 3)
    {
     // controllo se la posizione cliccata dalla griglia è interna al tratto i-esimo della tabella associata
      if(trackPosition[index + 1] <= lengthLine +Integer.parseInt(nLineElementY[i]))
      {
       // calcolo la coordinata di Y da inviare ad Arduino come il cateto di un triangolo rettangolo avente ipotenusa uguale alla lunghezza del tratto meno la coordinata Y e angolo compreso uguale all'angolo di inclinazione del tratto
        posY = posY + Math.round((trackPosition[index + 1] - lengthLine) * cos(Float.parseFloat(nLineElementY[i + 1]) * 3.14 / 180.0));
       // calcolo la coordinata di Z da inviare ad Arduino come il cateto di un triangolo rettangolo avente ipotenusa uguale alla lunghezza del tratto meno la coordinata Y e angolo opposto uguale all'angolo di inclinazione del tratto
        posZ = posZ + Math.round((trackPosition[index + 1] - lengthLine) * sin(Float.parseFloat(nLineElementY[i + 1]) * 3.14 / 180.0));
        flag = 1;  // aggiornamento variabile che mi permette di uscire dal ciclo
      }
      else
      {
       // calcolo la coordinata di Y da inviare ad Arduino come il cateto di un triangolo rettangolo avente ipotenusa uguale alla lunghezza del tratto e angolo compreso uguale all'angolo di inclinazione del tratto
        posY = posY + Math.round(Float.parseFloat(nLineElementY[i]) * cos(Float.parseFloat(nLineElementY[i + 1]) * 3.14 / 180.0));
       // calcolo la coordinata di Z da inviare ad Arduino come il cateto di un triangolo rettangolo avente ipotenusa uguale alla lunghezza del tratto e angolo opposto uguale all'angolo di inclinazione del tratto
        posZ = posZ + Math.round(Float.parseFloat(nLineElementY[i]) * sin(Float.parseFloat(nLineElementY[i + 1]) * 3.14 / 180.0));
       // aggiornamento della lunghezza lengthLine per il calcolo successivo al tratto i-esimo + 1
        lengthLine = lengthLine + Integer.parseInt(nLineElementY[i]);
      }
    }

    /* invio delle coordinate ad Arduino tramite porta seriale con una sequenza di informazioni in cui ogni coordinata viene accopiata ad un numero da 1 a 3.
       questo numero indica il motore da attivare:
        - 1: motore X
        - 2: motore Y
        - 3: motore Z
      Ogni messaggio è intervallato da un ritardo di 100ms in modo tale da dare tempo alla comunicazione esatta delle coordinate senza incappare nella perdita dei dati
    */
    String a = "";
    myPort.write("+001\n");
    delay(100);
    //myPort.write(trackPosition[index]);
```

```
  // myPort.write(posX);
  if(posX >= 0)
    a = "+";
  else
    a = "-";
  if(posX == 0)
    a = a + "000";
  else if(posX / 100 == 0 && posX / 10 == 0 && posX != 0)
    a = a + "00" + Integer.toString(abs(posX));
  else if(posX / 100 == 0 && posX / 10 != 0)
    a = a + "0" + Integer.toString(abs(posX));
  else if(posX / 100 != 0 && posX / 10 != 0)
    a = a + Integer.toString(abs(posX));
  myPort.write(a);
  myPort.write("\n");
  delay(100);
  myPort.write("+002\n");
  delay(100);
  //myPort.write(trackPosition[index + 1]);
 // myPort.write(posY);
  if(posY >= 0)
    a = "+";
  else
    a = "-";
  if(posY == 0)
    a = a + "000";
  else if(posY / 100 == 0 && posY / 10 == 0 && posY != 0)
    a = a + "00" + Integer.toString(abs(posY));
  else if(posY / 100 == 0 && posY / 10 != 0)
    a = a + "0" + Integer.toString(abs(posY));
  else if(posY / 100 != 0 && posY / 10 != 0)
    a = a + Integer.toString(abs(posY));
  myPort.write(a);
  myPort.write("\n");
  delay(100);
  myPort.write("+003\n");
  delay(100);
  //myPort.write(-1 * trackPosition[index + 2]);
//  myPort.write(-1 * posZ);
  if(posZ >= 0)
    a = "+";
  else
    a = "-";
  if(posZ == 0)
    a = a + "000";
  else if(posZ / 100 == 0 && posZ / 10 == 0 && posZ != 0)
    a = a + "00" + Integer.toString(abs(posZ));
  else if(posZ / 100 == 0 && posZ / 10 != 0)
    a = a + "0" + Integer.toString(abs(posZ));
  else if(posZ / 100 != 0 && posZ / 10 != 0)
    a = a + Integer.toString(abs(posZ));
  myPort.write(a);
```

```
      myPort.write("\n");
      delay(100);
      myPort.write("+003\n");
      delay(100);
      myPort.write(heightSensorMove);
      myPort.write("\n");
      fill(button_On);
      index += 4;  // aggiornamento variabile index indicando il numero dell'array delle posizioni in
cui verrà salvata la prossima coordinata X
      track++;  // aggiornamento numero conteggio delle posizioni
      delay(500);  // ritardo di 500ms
    }
    else if(mousePressed && (mouseButton == RIGHT))  // se invece viene premuto il tasto destro
    {
      // vengono cancellata l'ultima posizione inserita e vengono aggiornati i due parametri index e
track
      trackPosition[index] = 0;
      trackPosition[index + 1] = 0;
      trackPosition[index + 2] = 0;
      fill(button_Off);
      index -= 4;
      track--;
      delay(500);
    }
    else  // se non viene permuto alcun tasto del mouse ma si posiziona il cursore sopra la griglia si
imposta il colore di riempimento uguale a rosso
      fill(button_Off);
    ellipse(mouseX, mouseY, 7, 7);  // viene disegnato un cerchio di piccole dimensioni per indicare
la posizione del mouse all'interno della griglia
    fill(0);  // impostazione riempimento di colore nero
    textAlign(CENTER);  // funzione utile per allineare al centro della casella il testo
    // scrittura delle coordinate nelle caselle sottostanti la griglia
    text(positionX, 170 + widthBox / 2, positionGridY + heightGrid + 70);
    text(positionY, 370 + widthBox / 2, positionGridY + heightGrid + 70);
    text(positionZ, 570 + widthBox / 2, positionGridY + heightGrid + 70);
  }
  // ciclo utilizzato per il disegno delle posizioni cliccate nella griglia
  for(int i = 0; i < track; i++)
  {
    // disegna un cerchio di colore verde nelle coordinate del punto cliccato nella griglia
    fill(button_On);
    ellipse(positionGridX + trackPosition[i * 4 + 5], positionGridY + heightGrid - trackPosition[i * 4
+ 6], 7, 7);
    // scrive in rosso il numero della posizione cliccata vicino al pallino verde
    fill(250, 10, 10);
    text(i + 1, positionGridX + trackPosition[i * 4 + 5], positionGridY + heightGrid - trackPosition[i
* 4 + 6]);
  }
```

```
  /* Tabella Principale
    Questa tabella è fissa nella schermata e contiene tutti i parametri inerenti alla costruzione della griglia.
    In questa tabella oltre alle caselle di inserimento dei dati nelle varie voci in essa, ci sono 4 pulsanti rossi, i quali se vengono cliccati cambiano colore in verde e
    aprono una tabella a destra di dimensione variabile.

    La tabella è composta da 7 righe e 3 colonne.
    La seconda colonna è inerente all'asse X e la terza a quello Y.
    Le voci presenti nelle righe sono: 1) dimensioni griglia, 2) diametro foro, 3) passo fori, 4) distanza primo foro, 5) numero tratti e 6) numero montanti.
  */
  widthBox = 150;  // larghezza casella
  heightBox = 20;  // altezza casella
  nRawTable1 = 7;  // numero righe tabella
  nColumnTable1 = 3;  // numero colonne tabella
  widthTable1 = nColumnTable1 * widthBox;  // larghezza tabella
  heightTable1 = nRawTable1 * heightBox;  // altezza tabella
  positionX_Table1 = 50;  // posizione X spigolo superiore sinistro della tabella nella finestra
  positionY_Table1 = positionGridY + heightGrid + 100;  // posizione Y spigolo superiore sinistro della tabella nella finestra
  fill(table1_Off);  // impostazione riempimento di colore azzurro per colorare le caselle
  int count = 0;  // inizializzazione variabile count per il conteggio delle caselle
  // ciclo innestato per costruire la tabella casella per casella
  for(int i = 0; i < nRawTable1; i++)
  {
    for(int j = 0; j < nColumnTable1; j++)
    {
      // la casella assume un colore blu chiaro se elementTable1[] è uguale a 0 o un colore blu scuro se elementTable1[] è uguale a 1
      if(elementTable1[count] == 1)
        fill(table1_On);
      else
        fill(table1_Off);
      // costruisce la asella disegnando un rettangolo
      rect(positionX_Table1 + j * widthBox, positionY_Table1 + i * heightBox, widthBox, heightBox);
      if(i > 0 && j > 0)
      {
        fill(0);
        textAlign(CENTER);
        // viene scritto all'interno della casella il valore salvato nell'array elementMainTable[]
        text(elementMainTable[count], positionX_Table1 + j * widthBox + widthBox / 2, positionY_Table1 + 15 + i * heightBox);
      }
      count++;  // aggiornamento conteggio
      textAlign(LEFT);
    }
  }
  // vengono scritte tutte le voci della prima riga e della prima colonna
  fill(0);
  text("X", positionX_Table1 + 1 * widthBox + widthBox / 2 - 4, positionY_Table1 + 15);
```

```
  text("Y", positionX_Table1 + 2 * widthBox + widthBox / 2 - 4, positionY_Table1 + 15);
  text("Dimensioni griglia [mm]", positionX_Table1 + 5, positionY_Table1 + 1 * heightBox + 15);
  text("Diametro foro [mm]", positionX_Table1 + 5, positionY_Table1 + 2 * heightBox + 15);
  text("Passo fori [mm]", positionX_Table1 + 5, positionY_Table1 + 3 * heightBox + 15);
  text("Distanza 1° foro [mm]", positionX_Table1 + 5, positionY_Table1 + 4 * heightBox + 15);
  text("Numero tratti [mm]", positionX_Table1 + 5, positionY_Table1 + 5 * heightBox + 15);
  text("Numero montanti [mm]", positionX_Table1 + 5, positionY_Table1 + 6 * heightBox + 15);
  fill(button_Off);
  count = 0;  // azzeramento della variabile per il conteggio
  // ciclo innestato per disegnare i 4 pulsanti nelle caselle di X e Y nelle righe inerenti al numero di
tratti e il numero di montanti
  for(int i = 0; i < 2; i++)
  {
    for(int j = 0; j < 2; j++)
    {
      // il riempimento per colorare il pulsante è verde se attivato oppure rosso se non viene attivato
      if(button[count] == 1)
        fill(button_On);
      else
        fill(button_Off);
      // viene disegnato il quadrato utile per raffigurare il pulsante
      rect(positionX_Table1 + (2 + j) * widthBox - heightBox, positionY_Table1 + (5 + i) *
heightBox, heightBox, heightBox);
      count++;  // aggiornamento variabile conteggio
    }
  }

/* Tabella: Lunghezza e inclinazione tratti lungo X
  Tabella attivata tramite il pulsante alla destra nella casella del numero tratti in X.
  Ha una dimensione variabile in base al numero di tratti in X. Infatti ha un numero di righe uguali a
tale parametro sommato di uno, mentre le colonne sono fisse e sono tre
  La prima riga contiene il titolo della seconda e terza colonna, che son rispettivamente: "Lunghezza
[mm]" e "Inclinazione[°]".
  Mentre in ogni riga si indica il numero del tratto, lunghezza e inclinazione.
*/
  widthTable2 = 50;  // larghezza casella prima colonna
  widthTable3 = 120;  // larghezza casella seconda e terza colonna
  heightBox = 20;  // altezza casella
  nColumnTable2 = 3;  // numero colonne
  positionX_Table2 = positionX_Table1 + 500;  // posizione X spigolo superiore sinistro della
tabella
  positionY_Table2 = positionGridY + heightGrid + 100; // posizione Y spigolo superiore sinistro
della tabella
  fill(table1_Off);
  count = 0;  // azzeramento variabile conteggio
  if(button[0] == 1)  // controllo se il pulsante è attivo e quindi verde
  {
    fill(0);
    text("Lunghezza e inclinazione tratti lungo X", positionX_Table2, positionY_Table2 - 10);  //
scrive il titolo della tabella
    // calcola il numero di righe in base al numero scritto nella tabella principale sotto la voce numero
tratti in X
```

```
   if(elementTable1[16] == 0)
   {
    if(nRawTable2 > 1)
      nLineElementX[3 * nRawTable2 - 2] = "0";
    nRawTable2 = Integer.parseInt(elementMainTable[16]) + 1;  // il +1 tiene conto della prima riga
in cui vengono scritte le info sulle colonne
   }
   //ciclo innestato per la costruzione della tabella associata al numero di tratti in X
   for(int i = 0; i < nRawTable2; i++)
   {
    for(int j = 0; j < nColumnTable2; j++)
    {
     if(j <= 1)  // nelle prime colonne widthTable=50, alla terza colonna viene uguagliato a 120
       widthTable2 = 50;
     else
       widthTable2 = 120;
     if(j == 0)  // nella prima colonna widthTable3=50, alla seconda colonna viene uguagliato a 120
       widthTable3 = 50;
     else
       widthTable3 = 120;
     if(elementTable2[count] == 1)  // colora  la casella in modo diverso se è stata attivata o no
       fill(table1_On);
     else
       fill(table1_Off);
     // disegna la casella
     rect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3, heightBox);
     // salta la prima riga e la prima colonna per scrivere gli elementi dell'array nLineElementX[]
ovvero scrive la lunghezza e l'inclinazione del tratto
     if(i > 0 && j > 0)
     {
      fill(0);
      textAlign(CENTER);
      if(j == 1 && i == nRawTable2 - 1)  // si posiziona nella seconda colonna e ultima riga
      {
       int x = 0;  // inizializza una variabile utile nel calcolo dell'ultimo tratto
       if(elementTable1[4] == 0)  // controllo se è attiva o no la casella (2, 2) della tabella
principale
         x = Integer.parseInt(elementMainTable[4]);  // salva in x il valore della casella (2, 2) della
tabella principale, ovvero sotto la voce dimensione griglia in X
       // ciclo utilizzato per calcolare la lunghezza delll'ultimo tratto in base alla lunghezza dei
tratti precedenti
       // la variabili g inizia dal valore 4 perchè corrisponde al numero della poisizione della
casella (2, 2) della tabella associata
       // l'incremento di g è uguale a 3 per posizionarsi nella casella sottostante
       for(int g = 4; g < 3 * (nRawTable2 - 1) - 1; g += 3)
       {
        if(elementTable2[g] == 0)  // controllo se la casella è attiva o no
          x = x - Integer.parseInt(nLineElementX[g]);  // sottraggo a x il valore g-esimo letto
       }
       nLineElementX[3 * nRawTable2 - 2] =  Integer.toString(x);  // converto in stringa il valore x
appena calcolato
```

```
          // scrivo nella casella ultima riga seconda colonna ill risultato che corrisponde alla lunghezza dell'ultimo tratto
          text(nLineElementX[3 * nRawTable2 - 2], positionX_Table2 + widthTable3 / 2, positionY_Table2 + (nRawTable2 - 1) * heightBox + 15);
         }
         else
          //scrivo nella casella (i, g) il valore tratto dall'array nLineElementX[]
          text(nLineElementX[count], positionX_Table2 + widthTable3 / 2, positionY_Table2 + i * heightBox + 15);
        }
        positionX_Table2 = positionX_Table2 + widthTable3;  // aggiornamento coordinata X nella finestra dello spigolo superiore sinistro della casella
        count++;  // aggiornamento variabile conteggio posizione array
        textAlign(LEFT);
      }
      positionX_Table2 = positionX_Table1 + 500;  // aggiornamento coordinata X nella finestra dello spigolo superiore sinistro della casella
    }
    fill(0);  // riempimento di colore nero
    // scrittura caselle prima riiga seconda e terza colonna
    text("L [mm]", positionX_Table2 + 50 + widthTable2 / 2 - 20, positionY_Table2 + 15);
    text("Alpha [°]", positionX_Table2 + 50 + 1 * widthTable2 + widthTable2 / 2 - 25, positionY_Table2 + 15);
    // scrittura prima colonna e dalla seconda riga in giù del numero del tratto corrispondente
    for(int i = 1; i < nRawTable2; i++)
    {
      text(i, positionX_Table2 + 20, positionY_Table2 + i * heightBox + 15);
    }
  }

/* Tabella: Lunghezza e inclinazione tratti lungo Y
  Tabella attivata tramite il pulsante alla destra nella casella del numero tratti in Y.
  Ha una dimensione variabile in base al numero di tratti in Y. Infatti ha un numero di righe uguali a tale parametro sommato di uno, mentre le colonne sono fisse e sono tre
  La prima riga contiene il titolo della seconda e terza colonna, che son rispettivamente: "Lunghezza [mm]" e "Inclinazione[°]".
  Mentre in ogni riga si indica il numero del tratto, lunghezza e inclinazione.
*/
  widthTable2 = 50;  // larghezza casella prima colonna
  widthTable3 = 120;  // larghezza casella seconda colonna
  heightBox = 20;  // altezza casella

  nColumnTable3 = 3;  // numero di colonne
  positionX_Table2 = positionX_Table1 + 500;  // coordinata X spigolo superiore sinistro della prima casella della tebella
  positionY_Table2 = positionGridY + heightGrid + 100;  // coordinata Y spigolo superiore sinistro della prima casella della tebella
  fill(table1_Off);  // riempimento di colore chiaro della tabella
  count = 0;  // azzeramento variabile conteggio elementi dell'array
  if(button[1] == 1)  // controllo se la tabella è stata attivata tramite il pulsante correlato
  {
    fill(0);  // riempimento di colore  nero
```

```
   text("Lunghezza e inclinazione tratti lungo Y", positionX_Table2, positionY_Table2 - 10);  // scrittura titolo tabella  sopra di essa
   // calcola il numero di righe in base al numero scritto nella tabella principale sotto la voce numero tratti in Y
   if(elementTable1[17] == 0)
   {
    if(nRawTable3 > 1)
      nLineElementY[3 * nRawTable3 - 2] = "0";
    nRawTable3 = Integer.parseInt(elementMainTable[17]) + 1;  // il +1 tiene conto della prima riga in cui vengono scritte le info sulle colonne
   }
   //ciclo innestato per la costruzione della tabella associata al numero di tratti in Y
   for(int i = 0; i < nRawTable3; i++)
   {
    for(int j = 0; j < nColumnTable3; j++)
    {
     if(j <= 1)  // nelle prime colonne widthTable=50, alla terza colonna viene uguagliato a 120
      widthTable2 = 50;
     else
      widthTable2 = 120;
     if(j == 0)  // nella prima colonna widthTable3=50, alla seconda colonna viene uguagliato a 120
      widthTable3 = 50;
     else
      widthTable3 = 120;
     if(elementTable3[count] == 1)  // colora  la casella in modo diverso se è stata attivata o no
      fill(table1_On);
     else
      fill(table1_Off);
     // disegna la casella
     rect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3, heightBox);
     // salta la prima riga e la prima colonna per scrivere gli elementi dell'array nLineElementY[] ovvero scrive la lunghezza e l'inclinazione del tratto
     if(i > 0 && j > 0)
     {
      fill(0);
      textAlign(CENTER);
      if(j == 1 && i == nRawTable3 - 1)  // si posiziona nella seconda colonna e ultima riga
      {
       int x = 0;  // inizializza una variabile utile nel calcolo dell'ultimo tratto
       if(elementTable1[5] == 0)  // controllo se è attiva o no la casella (2, 3) della tabella principale
         x = Integer.parseInt(elementMainTable[5]);  // salva in x il valore della casella (2, 3) della tabella principale, ovvero sotto la voce dimensione griglia in Y
       // ciclo utilizzato per calcolare la lunghezza delll'ultimo tratto in base alla lunghezza dei tratti precedenti
       // la variabili g inizia dal valore 4 perchè corrisponde al numero della poisizione della casella (2, 3) della tabella associata
       // l'incremento di g è uguale a 3 per posizionarsi nella casella sottostante
       for(int g = 4; g < 3 * (nRawTable3 - 1) - 1; g += 3)
       {
        if(elementTable3[g] == 0)  // controllo se la casella è attiva o no
          x = x - Integer.parseInt(nLineElementY[g]);  // sottraggo a x il valore g-esimo letto
```

```
          }
          nLineElementY[3 * nRawTable3 - 2] =  Integer.toString(x);  // converto in stringa il valore x appena calcolato
          // scrivo nella casella ultima riga seconda colonna ill risultato che corrisponde alla lunghezza dell'ultimo tratto
          text(nLineElementY[3 * nRawTable3 - 2], positionX_Table2 + widthTable3 / 2, positionY_Table2 + (nRawTable3 - 1) * heightBox + 15);
         }
         else
          //scrivo nella casella (i, g) il valore tratto dall'array nLineElementY[]
          text(nLineElementY[count], positionX_Table2 + widthTable3 / 2, positionY_Table2 + i * heightBox + 15);
        }
        positionX_Table2 = positionX_Table2 + widthTable3;  // aggiornamento coordinata X nella finestra dello spigolo superiore sinistro della casella
        count++;  // aggiornamento variabile conteggio posizione array
        textAlign(LEFT);
      }
      positionX_Table2 = positionX_Table1 + 500;  // aggiornamento coordinata X nella finestra dello spigolo superiore sinistro della casella
    }
    fill(0);  // riempimento di colore nero
    // scrittura caselle prima riga seconda e terza colonna
    text("L [mm]", positionX_Table2 + 50 + widthTable2 / 2 - 20, positionY_Table2 + 15);
    text("Alpha [°]", positionX_Table2 + 50 + 1 * widthTable2 + widthTable2 / 2 - 25, positionY_Table2 + 15);
    // scrittura prima colonna e dalla seconda riga in giù del numero del tratto corrispondente
    for(int i = 1; i < nRawTable3; i++)
    {
      text(i, positionX_Table2 + 20, positionY_Table2 + i * heightBox + 15);
    }
  }

/* Tabella: Posizione e spessore montanti lungo X
  Tabella attivata tramite il pulsante alla destra nella casella del numero montanti in X.
  Ha una dimensione variabile in base al numero di montanti in X. Infatti ha un numero di righe uguali a tale parametro sommato di uno, mentre le colonne sono fisse e sono tre
  La prima riga contiene il titolo della seconda e terza colonna, che son rispettivamente: "L [mm]" e "Larghezza [mm]". Il primo tiene conto della distanza del montante dall'origine
  mentre il secondo del suo spessore.
  In ogni riga a partire dalla seconda si indicano il numero del tratto, L e larghezza.
*/
  widthTable2 = 50;  // larghezza della casella della prima colonna
  widthTable3 = 120;  // larghezza della casella della seconda e terza colonna
  heightBox = 20;  // altezza della casella
  if(elementTable1[19] == 0)  // controllo se la casella della tabella principale inerente al numero di montanti in X non è attiva
    nRawTable4 = Integer.parseInt(elementMainTable[19]) + 1;  // salvo il numero delle righe e aggiungo uno per la prima riga con i titoli delle colonne
  nColumnTable4 = 3;  // numero delle colonne
  positionX_Table2 = positionX_Table1 + 500;  // coordinata X dello spigolo superiore sinistro della prima casella
```

```
  positionY_Table2 = positionGridY + heightGrid + 100;  // coordinata Y dello spigolo superiore
sinistro della prima casella
  fill(table1_Off);
  count = 0;  // azzeramento della variabile utilizzata per il conteggio degli elementi dell'array
  if(button[2] == 1)  // controllo se la tabella è stata attivata tramite il pulsante corrispondente
  {
    fill(0);  // riempimento di colore nero
    // scrittura titolo tabella sopra di essa
    text("Posizione e spessore montanti lungo X", positionX_Table2, positionY_Table2 - 10);
    // ciclo innestato per la costruzione della tabella seguendo l'ordine: prima riga->prima, seconda e
terza colonna, seconda riga->prima, seconda e terza colonna, ecc.
    for(int i = 0; i < nRawTable4; i++)
    {
      for(int j = 0; j < nColumnTable4; j++)
      {
        if(j <= 1)  // nelle prime colonne widthTable=50, alla terza colonna viene uguagliato a 120
          widthTable2 = 50;
        else
          widthTable2 = 120;
        if(j == 0)  // nella prima colonna widthTable3=50, alla seconda colonna viene uguagliato a 120
          widthTable3 = 50;
        else
          widthTable3 = 120;
        if(elementTable4[count] == 1)  // colora  la casella in modo diverso se è stata attivata o no
          fill(table1_On);
        else
          fill(table1_Off);
        // disegna la casella
        rect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3, heightBox);
        // salta la prima riga e la prima colonna per scrivere gli elementi dell'array strutElementX[]
ovvero scrive la lunghezza e lo spessore del montante
        if(i > 0 && j > 0)
        {
          fill(0);  // rimepimento di colore nero
          textAlign(CENTER);
          // scrivo nella casella (i, g) il valore tratto dall'array strutElementX[]
          text(strutElementX[count], positionX_Table2 + widthTable3 / 2, positionY_Table2 + i *
heightBox + 15);
        }
        positionX_Table2 = positionX_Table2 + widthTable3;  // aggiornamento coordinata X nella
finestra dello spigolo superiore sinistro della casella
        textAlign(LEFT);
        count++;  // aggiornamento variabile conteggio elementi dell'array strutElementX[]
      }
      positionX_Table2 = positionX_Table1 + 500;  // aggiornamento coordinata X nella finestra dello
spigolo superiore sinistro della prima casella
    }
    fill(0);  // riempimento di colore nero
    // scrittura caselle prima riga seconda e terza colonna
    text("L [mm]", positionX_Table2 + 50 + widthTable2 / 2 - 20, positionY_Table2 + 15);
    text("Larghezza [mm]", positionX_Table2 + 50 + 1 * widthTable2 + widthTable2 / 2 - 50,
positionY_Table2 + 15);
```

```
    // scrittura prima colonna e dalla seconda riga in giù del numero del montante corrispondente
    for(int i = 1; i < nRawTable4; i++)
    {
      text(i, positionX_Table2 + 20, positionY_Table2 + i * heightBox + 15);
    }
  }

/* Tabella: Posizione e spessore montanti lungo Y
  Tabella attivata tramite il pulsante alla destra nella casella del numero montanti in Y.
  Ha una dimensione variabile in base al numero di montanti in Y. Infatti ha un numero di righe uguali a tale parametro sommato di uno, mentre le colonne sono fisse e sono tre
  La prima riga contiene il titolo della seconda e terza colonna, che son rispettivamente: "L [mm]" e "Larghezza [mm]". Il primo tiene conto della distanza del montante dall'origine
  mentre il secondo del suo spessore.
  In ogni riga a partire dalla seconda si indicano il numero del tratto, L e larghezza.
*/
  widthTable2 = 50;  // larghezza casella prima colonna
  widthTable3 = 120;  // larghezza casella seconda e terza colonna
  heightBox = 20;  // altezza casella
  if(elementTable1[20] == 0)  // controllo se la casella della tabella principale inerente al numero di montanti in Y non è attiva
    nRawTable5 = Integer.parseInt(elementMainTable[20]) + 1;  // salvo il numero delle righe e aggiungo uno per la prima riga con i titoli delle colonne
  nColumnTable5 = 3;  // numero delle colonne
  positionX_Table2 = positionX_Table1 + 500;  // coordinata X dello spigolo superiore sinistro della prima casella
  positionY_Table2 = positionGridY + heightGrid + 100;  // coordinata Y dello spigolo superiore sinistro della prima casella
  fill(table1_Off);
  count = 0;  // azzeramento della variabile utilizzata per il conteggio degli elementi dell'array
  if(button[3] == 1)  // controllo se la tabella è stata attivata tramite il pulsante corrispondente
  {
    fill(0);  // riempimento di colore nero
    // scrittura titolo tabella sopra di essa
    text("Posizione e spessore montanti lungo Y", positionX_Table2, positionY_Table2 - 10);
    // ciclo innestato per la costruzione della tabella seguendo l'ordine: prima riga->prima, seconda e terza colonna, seconda riga->prima, seconda e terza colonna, ecc.
    for(int i = 0; i < nRawTable5; i++)
    {
      for(int j = 0; j < nColumnTable5; j++)
      {
        if(j <= 1)  // nelle prime colonne widthTable=50, alla terza colonna viene uguagliato a 120
          widthTable2 = 50;
        else
          widthTable2 = 120;
        if(j == 0)  // nella prima colonna widthTable3=50, alla seconda colonna viene uguagliato a 120
          widthTable3 = 50;
        else
          widthTable3 = 120;
        if(elementTable5[count] == 1)  // colora  la casella in modo diverso se è stata attivata o no
          fill(table1_On);
        else
```

```
        fill(table1_Off);
       // disegna la casella
       rect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3, heightBox);
       // salta la prima riga e la prima colonna per scrivere gli elementi dell'array strutElementY[]
ovvero scrive la lunghezza e lo spessore del montante
       if(i > 0 && j > 0)
       {
        fill(0);  // riempimento di colore nero
        textAlign(CENTER);
        // scrivo nella casella (i, g) il valore tratto dall'array strutElementY[]
        text(strutElementY[count], positionX_Table2 + widthTable3 / 2, positionY_Table2 + i *
heightBox + 15);
       }
       positionX_Table2 = positionX_Table2 + widthTable3;  // aggiornamento coordinata X nella
finestra dello spigolo superiore sinistro della casella
       textAlign(LEFT);
       count++;  // aggiornamento variabile conteggio elementi dell'array strutElementY[]
     }
     positionX_Table2 = positionX_Table1 + 500;  // aggiornamento coordinata X nella finestra dello
spigolo superiore sinistro della prima casella
   }
   fill(0);  // riempimento di colore nero
   // scrittura caselle prima riga seconda e terza colonna
   text("L [mm]", positionX_Table2 + 50 + widthTable2 / 2 - 20, positionY_Table2 + 15);
   text("Larghezza [mm]", positionX_Table2 + 50 + 1 * widthTable2 + widthTable2 / 2 - 50,
positionY_Table2 + 15);
   // scrittura prima colonna e dalla seconda riga in giù del numero del montante corrispondente
   for(int i = 1; i < nRawTable5; i++)
   {
    text(i, positionX_Table2 + 20, positionY_Table2 + i * heightBox + 15);
   }
  }

/* Tabella: Posizioni percorso da eseguire
  Tabella attivata tramite il pulsante alla destra delle caselle chhe indicano le coordinate del cursore
all'interno della griglia.
  Ha una dimensione variabile in base al numero di posizioni cliccate nella griglia. Infatti ha un
numero di righe uguali a tale parametro sommato di uno, mentre le colonne sono fisse e sono
quattro
  La prima riga contiene il titolo della seconda, terza e quarta colonna, che son rispettivamente: "X
[mm]", "Y [mm]" e "Z [mm]".
  In ogni riga eccetto la prima si indicano il numero della posizione e le sue coordinate X, Y e Z.
*/
  widthTable2 = 50;  // larghezza casella prima colonna
  widthTable3 = 120;  // larghezza casella seconda, terza e quarta colonna
  heightBox = 20;  // altezza casella
  nRawTable6 = track + 1;  // numero di righe, in base al numero di posizioni cliccate sommato a
uno per la prima riga che contiene i titoli delle colonne
  nColumnTable6 = 4;  // numero di colonne
  positionX_Table2 = positionX_Table1 + 500;  // coordinata X dello spigolo superiore sinistro della
prima casella
```

```
  positionY_Table2 = positionGridY + heightGrid + 100;  // coordinata Y dello spigolo superiore
sinistro della prima casella
  fill(table1_Off);
  count = 0;  // azzeramento della variabile utilizzata per il conteggio degli elementi dell'array
  if(button[4] == 1)  // controllo se la tabella è stata attivata tramite il pulsante corrispondente
  {
    fill(0);  // riempimento di colore nero
    // scrittura titolo tabella sopra di essa
    text("Posizioni percorso da eseguire", positionX_Table2, positionY_Table2 - 10);
    // ciclo innestato per la costruzione della tabella seguendo l'ordine: prima riga->prima, seconda e
terza colonna, seconda riga->prima, seconda e terza colonna, ecc.
    for(int i = 0; i < nRawTable6; i++)
    {
      for(int j = 0; j < nColumnTable6; j++)
      {
        if(j <= 1)  // nelle prime colonne widthTable=50, alla terza colonna viene uguagliato a 120
          widthTable2 = 50;
        else
          widthTable2 = 120;
        if(j == 0)  // nella prima colonna widthTable3=50, alla seconda colonna viene uguagliato a 120
          widthTable3 = 50;
        else
          widthTable3 = 120;
        if(elementTable6[count] == 1)  // colora  la casella in modo diverso se è stata attivata o no
          fill(table1_On);
        else
          fill(table1_Off);
        // disegna la casella
        rect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3, heightBox);
        // salta la prima riga e la prima colonna per scrivere gli elementi dell'array trackPosition[]
ovvero scrive le coordinate delle posizioni cliccate
        if(i > 0 && j > 0)
        {
          fill(0);  // riempimento di colore nero
          textAlign(CENTER);
          // scrivo nella casella (i, g) il valore tratto dall'array trackPosition[]
          text(trackPosition[count], positionX_Table2 + widthTable3 / 2, positionY_Table2 + i *
heightBox + 15);
        }
        positionX_Table2 = positionX_Table2 + widthTable3;  // aggiornamento coordinata X nella
finestra dello spigolo superiore sinistro della casella
        textAlign(LEFT);
        count++;  // aggiornamento variabile conteggio elementi dell'array strutElementY[]
      }
     positionX_Table2 = positionX_Table1 + 500;  // aggiornamento coordinata X nella finestra dello
spigolo superiore sinistro della prima casella
    }
    fill(0);  // riempimento di colore nero
    // scrittura caselle prima riga seconda, terza e quarta colonna
    text("X [mm]", positionX_Table2 + 50 + widthTable2 / 2 - 20, positionY_Table2 + 15);
    text("Y [mm]", positionX_Table2 + 50 + 1 * widthTable2 + widthTable2 / 2 - 20,
positionY_Table2 + 15);
```

```
    text("Z [mm]", positionX_Table2 + 50 + 2 * widthTable2 + widthTable2 / 2 - 20,
positionY_Table2 + 15);
    // scrittura prima colonna e dalla seconda riga in giù del numero della posizione cliccata
corrispondente
    for(int i = 1; i < nRawTable6; i++)
    {
      text(i, positionX_Table2 + 20, positionY_Table2 + i * heightBox + 15);
    }
  }

  int flag = 0;  //variabile utilizzata per uscire dal ciclo
  int lengthLine = 0;  // variabile che tiene conto della lunghezza dei vari tratti considerati per il
calcolo delle coordinate da inviare ad Arduino
  int sensorX = 0;  // coordinata X da inviare ad Arduino
  int sensorY = 0;  // coordinata Y da inviare ad Arduino
  int sensorZ = Integer.parseInt(positionSensorZ);  // coordinata Z da inviare ad Arduino, dove si
imposta come valore iniziale il valore letto dalla griglia
  // ciclo utilizzato per leggere tutti gli elementi presenti nell'array nLineElementX[] dove sono
salvati i parametri indicanti la lunghezza dei vari tratti con la loro inclinazione
  // si parte da i = 4, poichè è il numero della posizione della casella prima riga e seconda colonna
all'interno dell'array in cui vengono salvati i vari parametri della tabella associata
  // l'incremento di i è di 3 poichè salta alla riga successiva della tabella in cui è posto l'elemento
dell'array utile per il calcolo della coordinata X
  for(int i = 4; i < nLineElementX.length && flag == 0 && elementTable2[i] == 0 &&
elementTable2[i+1] == 0; i += 3)
  {
    // controllo se la posizione cliccata dalla griglia è interna al tratto i-esimo della tabella associata
    if(Integer.parseInt(positionSensorX) <= lengthLine +
Math.round(Integer.parseInt(nLineElementX[i]) * cos(Float.parseFloat(nLineElementX[i + 1]) *
3.14 / 180.0)))
    {
      // calcolo la coordinata di X da inviare ad Arduino come il cateto di un triangolo rettangolo
avente ipotenusa uguale alla lunghezza del tratto meno la coordinata X e angolo compreso uguale
all'angolo di inclinazione del tratto
      sensorX = sensorX + Math.round((Float.parseFloat(positionSensorX) - lengthLine) /
cos(Float.parseFloat(nLineElementX[i + 1]) * 3.14 / 180.0));
      // calcolo la coordinata di Z da inviare ad Arduino come il cateto di un triangolo rettangolo
avente ipotenusa uguale alla lunghezza del tratto meno la coordinata X e angolo opposto uguale
all'angolo di inclinazione del tratto
      sensorZ = sensorZ + Math.round((Float.parseFloat(positionSensorX) - lengthLine) /
cos(Float.parseFloat(nLineElementX[i + 1]) * 3.14 / 180.0) * sin(Float.parseFloat(nLineElementX[i
+ 1]) * 3.14 / 180.0));
      flag = 1;  // aggiornamento variabile che mi permette di uscire dal ciclo
    }
    else
    {
      // calcolo la coordinata di X da inviare ad Arduino come il cateto di un triangolo rettangolo
avente ipotenusa uguale alla lunghezza del tratto e angolo compreso uguale all'angolo di
inclinazione del tratto
      sensorX = sensorX + Integer.parseInt(nLineElementX[i]);
```

```
      // calcolo la coordinata di Z da inviare ad Arduino come il cateto di un triangolo rettangolo avente ipotenusa uguale alla lunghezza del tratto e angolo opposto uguale all'angolo di inclinazione del tratto
      sensorZ = sensorZ + Math.round(Float.parseFloat(nLineElementX[i]) * sin(Float.parseFloat(nLineElementX[i + 1]) * 3.14 / 180.0));
      // aggiornamento della lunghezza lengthLine per il calcolo successivo al tratto i-esimo + 1
      lengthLine = lengthLine + Math.round(Integer.parseInt(nLineElementX[i]) * cos(Float.parseFloat(nLineElementX[i + 1]) * 3.14 / 180.0));
      //lengthLine = lengthLine + Math.round(Float.parseFloat(nLineElementX[i]) * cos(Float.parseFloat(nLineElementX[i + 1]) * 3.14 / 180.0));
    }
   }
   // azzeramento delle variabili flag e lengthLine per l'esecuzione di un ciclo simile per la coordinata Y
   flag = 0;
   lengthLine = 0;
   // ciclo utilizzato per leggere tutti gli elementi presenti nell'array nLineElementY[] dove sono salvati i parametri indicanti la lunghezza dei vari tratti con la loro inclinazione
   // si parte da i = 4, poichè è il numero della posizione della casella prima riga e seconda colonna all'interno dell'array in cui vengono salvati i vari parametri della tabella associata
   // l'incremento di i è di 3 poichè salta alla riga successiva della tabella in cui è posto l'elemento dell'array utile per il calcolo della coordinata Y
   for(int i = 4; i < nLineElementY.length && flag == 0 && elementTable3[i] == 0 && elementTable3[i+1] == 0; i += 3)
   {
    // controllo se la posizione cliccata dalla griglia è interna al tratto i-esimo della tabella associata
    if(Integer.parseInt(positionSensorY) <= lengthLine +Integer.parseInt(nLineElementY[i]))
    {
     // calcolo la coordinata di Y da inviare ad Arduino come il cateto di un triangolo rettangolo avente ipotenusa uguale alla lunghezza del tratto meno la coordinata Y e angolo compreso uguale all'angolo di inclinazione del tratto
     sensorY = sensorY + Math.round((Float.parseFloat(positionSensorY) - lengthLine) / cos(Float.parseFloat(nLineElementY[i + 1]) * 3.14 / 180.0));
     // calcolo la coordinata di Z da inviare ad Arduino come il cateto di un triangolo rettangolo avente ipotenusa uguale alla lunghezza del tratto meno la coordinata Y e angolo opposto uguale all'angolo di inclinazione del tratto
     sensorZ = sensorZ + Math.round((Float.parseFloat(positionSensorY) - lengthLine) / cos(Float.parseFloat(nLineElementY[i + 1]) * 3.14 / 180.0) * sin(Float.parseFloat(nLineElementY[i + 1]) * 3.14 / 180.0));
     flag = 1;  // aggiornamento variabile che mi permette di uscire dal ciclo
    }
    else
    {
     // calcolo la coordinata di Y da inviare ad Arduino come il cateto di un triangolo rettangolo avente ipotenusa uguale alla lunghezza del tratto e angolo compreso uguale all'angolo di inclinazione del tratto
     sensorY = sensorY + Integer.parseInt(nLineElementY[i]);
     // calcolo la coordinata di Z da inviare ad Arduino come il cateto di un triangolo rettangolo avente ipotenusa uguale alla lunghezza del tratto e angolo opposto uguale all'angolo di inclinazione del tratto
     sensorZ = sensorZ + Math.round(Float.parseFloat(nLineElementY[i]) * sin(Float.parseFloat(nLineElementY[i + 1]) * 3.14 / 180.0));
```

```
      // aggiornamento della lunghezza lengthLine per il calcolo successivo al tratto i-esimo + 1
      lengthLine = lengthLine + Math.round(Integer.parseInt(nLineElementY[i]) *
cos(Float.parseFloat(nLineElementY[i + 1]) * 3.14 / 180.0));
    }
  }

  fill(0);
  textAlign(CENTER);
  text(sensorX, 170 + widthBox / 2, positionGridY + heightGrid + 40);
  text(sensorY, 370 + widthBox / 2, positionGridY + heightGrid + 40);
  text(sensorZ, 570 + widthBox / 2, positionGridY + heightGrid + 40);
  textAlign(LEFT);
  fill(0,0,255);
  ellipse(sensorX + positionGridX, positionGridY + heightGrid - sensorY, 8, 8);
}


void serialEvent(Serial myPort)
{
  println("event");
  serial = myPort.readStringUntil('\n');
  if (serial != null) //if the string is not empty, do the the following
  {
    println("serial != null");
    String[] b = split(serial, ',');
    positionSensorX = b[0];
    positionSensorY = b[1];
    positionSensorZ = b[2];
    serial = null;
  }
}

/* Funzione: mousePressed()
  Funzione che si attiva solo quando viene cliccato un punto nella finestra.
  Lo scopo di questa funzione è quello di capire dove il cursore del mouse si posiziona e al momento
del click di cambiare alcuni parametri come ad esempio il colore di una
  casella oppure disegnare un pallino che identifica una posizione all'interno della griglia.
*/
void mousePressed()
{
  int count = 0;  // inizializzazione variabile che conteggia gli elementi di un array
  // ciclo innestato utilizzato per la scansione della tabella principale
  for(int i = 0; i < nRawTable1; i++)
  {
    for(int j = 0; j < nColumnTable1; j++)
    {
      int widthX;  //parametro che comunica la larghezza della casella
      if(i >= nRawTable1 - 2)  // impostazione nuova larghezza della casella per le ultime due righe
dellla tabella principale per non sovrapporsi con i pulsanti associati
        widthX = widthBox - 20;
      else
        widthX = widthBox;
```

```
      // controllo se nel momento in cui è stato premuto il tasto del mouse il cursore era sopra la casella di riga i-esima e colonna g-esima
      // si varia il valore dell'array elementTable[] da 0 a 1 o viceversa per comunicare se è stata attivata o meno la casella in questione
      if(overRect(positionX_Table1 + j * widthBox, positionY_Table1 + i * heightBox, widthX, heightBox) && elementTable1[count] == 0)
        elementTable1[count] = 1;
      else if(overRect(positionX_Table1 + j * widthBox, positionY_Table1 + i * heightBox, widthX, heightBox) && elementTable1[count] == 1)
        elementTable1[count] = 0;
      count++;  // aggiornamento variabile conteggio
    }
  }
  count = 0;  // azzeramento variabile conteggio elementi dell'array
  // ciclo innestato per scansionare lo stato dei 4 pulsanti presenti nella tabella principale
  for(int i = 0; i < 2; i++)
  {
    for(int j = 0; j < 2; j++)
    {
      // controllo se nel momento in cui è stato premuto il tasto del mouse il cursore era sopra la casella di riga i-esima e colonna g-esima
      // si varia il valore dell'array button[] da 0 a 1 o viceversa per comunicare se è stata attivata o meno la casella in questione
      if(overRect(positionX_Table1 + (2 + j) * widthBox - heightBox, positionY_Table1 + (5 + i) * heightBox, heightBox, heightBox) && button[count] == 0)
        button[count] = 1;
      else if(overRect(positionX_Table1 + (2 + j) * widthBox - heightBox, positionY_Table1 + (5 + i) * heightBox, heightBox, heightBox) && button[count] == 1)
        button[count] = 0;
      count++;  // aggiornamento variabile conteggio
    }
  }
  // controllo se il pulsante relativo al numero dei tratti in X è attivo
  if(button[0] == 1)
  {
    widthTable2 = 50;  // larghezza casella prima colonna
    widthTable3 = 120;  // larghezza casella seconda e terza colonna
    positionX_Table2 = positionX_Table1 + 500;  // coordinata X dello spigolo superiore sinistro della prima casella
    positionY_Table2 = positionGridY + heightGrid + 100;  // coordinata Y dello spigolo superiore sinistro della prima casella
    count = 0;  // azzeramento della variabile per il conteggio degli elementi dell'array
    // ciclo innestato per scansionare casella per casella della tabella "Lunghezza e inclinazione tratti lungo X"
    for(int i = 0; i < nRawTable2; i++)
    {
      for(int j = 0; j < nColumnTable2; j++)
      {
        if(j <= 1)
          widthTable2 = 50;
        else
          widthTable2 = 120;
```

```
      if(j == 0)
        widthTable3 = 50;
      else
        widthTable3 = 120;
      // controllo se nel momento in cui è stato premuto il tasto del mouse il cursore era sopra la casella di riga i-esima e colonna g-esima
      // si varia il valore dell'array elementTable2[] da 0 a 1 o viceversa per comunicare se è stata attivata o meno la casella in questione
      if(overRect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3, heightBox) && elementTable2[count] == 0)
        elementTable2[count] = 1;
      else if(overRect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3, heightBox) && elementTable2[count] == 1)
        elementTable2[count] = 0;
      positionX_Table2 = positionX_Table2 + widthTable3;  // aggiornamento coordinata X nella finestra dello spigolo superiore sinistro della casella
      count++;  // aggiornamento variabile conteggio
     }
     positionX_Table2 = positionX_Table1 + 500;  // aggiornamento coordinata X nella finestra dello spigolo superiore sinistro della prima casella
   }
  }
  // controllo se il pulsante relativo al numero dei tratti in Y è attivo
  if(button[1] == 1)
  {
   widthTable2 = 50;  // larghezza casella prima colonna
   widthTable3 = 120;  // larghezza casella seconda e terza colonna
   positionX_Table2 = positionX_Table1 + 500;  // coordinata X dello spigolo superiore sinistro della prima casella
   positionY_Table2 = positionGridY + heightGrid + 100;  // coordinata Y dello spigolo superiore sinistro della prima casella
   count = 0;  // azzeramento della variabile per il conteggio degli elementi dell'array
   // ciclo innestato per scansionare casella per casella della tabella "Lunghezza e inclinazione tratti lungo Y"
   for(int i = 0; i < nRawTable3; i++)
   {
    for(int j = 0; j < nColumnTable3; j++)
    {
     if(j <= 1)
       widthTable2 = 50;
     else
       widthTable2 = 120;
     if(j == 0)
       widthTable3 = 50;
     else
       widthTable3 = 120;
     // controllo se nel momento in cui è stato premuto il tasto del mouse il cursore era sopra la casella di riga i-esima e colonna g-esima
     // si varia il valore dell'array elementTable3[] da 0 a 1 o viceversa per comunicare se è stata attivata o meno la casella in questione
     if(overRect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3, heightBox) && elementTable3[count] == 0)
```

```
        elementTable3[count] = 1;
       else if(overRect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3,
heightBox) && elementTable3[count] == 1)
        elementTable3[count] = 0;
       positionX_Table2 = positionX_Table2 + widthTable3;  // aggiornamento coordinata X nella
finestra dello spigolo superiore sinistro della casella
       count++;  // aggiornamento variabile conteggio
      }
     positionX_Table2 = positionX_Table1 + 500;  // aggiornamento coordinata X nella finestra dello
spigolo superiore sinistro della prima casella
    }
  }
  // controllo se il pulsante relativo al numero dei montanti in X è attivo
  if(button[2] == 1)
  {
   widthTable2 = 50;  // larghezza casella prima colonna
   widthTable3 = 120;  // larghezza casella seconda e terza colonna
   positionX_Table2 = positionX_Table1 + 500;  // coordinata X dello spigolo superiore sinistro
della prima casella
   positionY_Table2 = positionGridY + heightGrid + 100;  // coordinata Y dello spigolo superiore
sinistro della prima casella
   count = 0;  // azzeramento della variabile per il conteggio degli elementi dell'array
   // ciclo innestato per scansionare casella per casella della tabella "Posizione e spessore montanti
lungo X"
   for(int i = 0; i < nRawTable4; i++)
   {
    for(int j = 0; j < nColumnTable4; j++)
    {
     if(j <= 1)
      widthTable2 = 50;
     else
      widthTable2 = 120;
     if(j == 0)
      widthTable3 = 50;
     else
      widthTable3 = 120;
     // controllo se nel momento in cui è stato premuto il tasto del mouse il cursore era sopra la
casella di riga i-esima e colonna g-esima
     // si varia il valore dell'array elementTable4[] da 0 a 1 o viceversa per comunicare se è stata
attivata o meno la casella in questione
     if(overRect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3, heightBox)
&& elementTable4[count] == 0)
      elementTable4[count] = 1;
     else if(overRect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3,
heightBox) && elementTable4[count] == 1)
      elementTable4[count] = 0;
     positionX_Table2 = positionX_Table2 + widthTable3;  // aggiornamento coordinata X nella
finestra dello spigolo superiore sinistro della casella
     count++;  // aggiornamento variabile conteggio
    }
   positionX_Table2 = positionX_Table1 + 500;  // aggiornamento coordinata X nella finestra dello
spigolo superiore sinistro della prima casella
```

```
   }
  }
  // controllo se il pulsante relativo al numero dei montanti in Y è attivo
  if(button[3] == 1)
  {
   widthTable2 = 50;  // larghezza casella prima colonna
   widthTable3 = 120;  // larghezza casella seconda e terza colonna
   positionX_Table2 = positionX_Table1 + 500;  // coordinata X dello spigolo superiore sinistro
della prima casella
   positionY_Table2 = positionGridY + heightGrid + 100;  // coordinata Y dello spigolo superiore
sinistro della prima casella
   count = 0;  // azzeramento della variabile per il conteggio degli elementi dell'array
   // ciclo innestato per scansionare casella per casella della tabella "Posizione e spessore montanti
lungo Y"
   for(int i = 0; i < nRawTable5; i++)
   {
    for(int j = 0; j < nColumnTable5; j++)
    {
     if(j <= 1)
      widthTable2 = 50;
     else
      widthTable2 = 120;
     if(j == 0)
      widthTable3 = 50;
     else
      widthTable3 = 120;
     // controllo se nel momento in cui è stato premuto il tasto del mouse il cursore era sopra la
casella di riga i-esima e colonna g-esima
     // si varia il valore dell'array elementTable5[] da 0 a 1 o viceversa per comunicare se è stata
attivata o meno la casella in questione
     if(overRect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3, heightBox)
&& elementTable5[count] == 0)
      elementTable5[count] = 1;
     else if(overRect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3,
heightBox) && elementTable5[count] == 1)
      elementTable5[count] = 0;
     positionX_Table2 = positionX_Table2 + widthTable3;  // aggiornamento coordinata X nella
finestra dello spigolo superiore sinistro della casella
     count++;  // aggiornamento variabile conteggio
    }
    positionX_Table2 = positionX_Table1 + 500;  // aggiornamento coordinata X nella finestra dello
spigolo superiore sinistro della prima casella
   }
  }

  // controllo se nel momento in cui è stato premuto il tasto del mouse il cursore era sopra il pulsante
posizionato vicino la scritta "Elenco percorso:"
  // si varia il valore dell'array button[4] da 0 a 1 o viceversa per comunicare se è stata attivata o
meno la casella in questione
  if(overRect(840, positionGridY + heightGrid + 50, heightBox, heightBox) && button[4] == 0)
   button[4] = 1;
```

```
  else if(overRect(840, positionGridY + heightGrid + 50, heightBox, heightBox) && button[4] ==
1)
    button[4] = 0;
  // controllo se il pulsante relativo all'elenco percorso è attivo
  if(button[4] == 1)
  {
    widthTable2 = 50;  // larghezza casella prima colonna
    widthTable3 = 120;  // larghezza casella seconda e terza colonna
    nRawTable6 = track + 1;  // numero di righe della tabella uguale al numero di posizioni cliccate
nella griglia + 1 per i titoli delle colonne nella prima riga
    nColumnTable6 = 4;  // numero colonne della tabella
    positionX_Table2 = positionX_Table1 + 500;  // coordinata X dello spigolo superiore sinistro
della prima casella
    positionY_Table2 = positionGridY + heightGrid + 100;  // coordinata Y dello spigolo superiore
sinistro della prima casella
    count = 0;  // azzeramento della variabile per il conteggio degli elementi dell'array
    // ciclo innestato per scansionare casella per casella della tabella "Posizioni percorso da eseguire"
    for(int i = 0; i < nRawTable6; i++)
    {
      for(int j = 0; j < nColumnTable6; j++)
      {
        if(j <= 1)
          widthTable2 = 50;
        else
          widthTable2 = 120;
        if(j == 0)
          widthTable3 = 50;
        else
          widthTable3 = 120;
        // controllo se nel momento in cui è stato premuto il tasto del mouse il cursore era sopra la
casella di riga i-esima e colonna g-esima
        // si varia il valore dell'array elementTable6[] da 0 a 1 o viceversa per comunicare se è stata
attivata o meno la casella in questione
        if(overRect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3, heightBox)
&& elementTable6[count] == 0)
          elementTable6[count] = 1;
        else if(overRect(positionX_Table2, positionY_Table2 + i * heightBox, widthTable3,
heightBox) && elementTable6[count] == 1)
          elementTable6[count] = 0;
        positionX_Table2 = positionX_Table2 + widthTable3;  // aggiornamento coordinata X nella
finestra dello spigolo superiore sinistro della casella
        count++;  // aggiornamento variabile conteggio
      }
      positionX_Table2 = positionX_Table1 + 500;  // aggiornamento coordinata X nella finestra dello
spigolo superiore sinistro della prima casella
    }
  }
}

/* Funzione: keyPressed()
  Funzione che si attiva quando vengono premuti i tasti della tastiera.
```

```
  Questa funzione viene sfruttata per scrivere nell'interfaccia. Essa si attiva ad ogni input di tastiera
dunque per far scrivere in una determinata casella quest'ultima
  dev'essere prima attivata tramite click del mouse sopra di essa. Se è attiva la casella sarà di un
colore più scuro, se disattivata ritornerà ad avere un colore più chiaro.
  ATTENZIONE: dentro ad ogni casella va scritto in modo corretto tutti i numeri prestando
attenzione a non inserire altri caratteri come "INVIO" che potrebbe mandare in errore
  il programma perchè non in grado di inserire il tasto input che valore corrisponde oppure
attenzione a non schiacciare troppo il tasto "BACKSPACE" o cancella perchè si
  può rischiare di tagliare la stringa più della sua lunghezza e il programma va in errore bloccando
l'interfaccia.
*/
void keyPressed()
{
  // ciclo che scorre tutti gli elementi dell'array elementMainTable[] in cui vengono salvati tutti i
dati contenuti all'interno di ogni casella della tabella principale
  for(int i = 0; i < elementMainTable.length; i++)
  {
    // controllo se la casella i-esima della tabella principale è attiva consentendo così la scrittura
all'interno di essa
    if(elementTable1[i] == 1)
    {
      // controllo se viene permuto il tasto BACKSPACE (cancella carattere)
      if(key == BACKSPACE) {
        // controllo se la lunghezza della stringa è maggiore di zero
        if (elementMainTable[i].length() > 0) {
          // cancello l'ultimo carattere della stringa troncando la sua lunghezza di 1
          elementMainTable[i] = elementMainTable[i].substring(0, elementMainTable[i].length()-1);
        }
      }
      // controllo il testo può essere contenuto nella finestra
      else if(textWidth(elementMainTable[i]+key) < width)  {
        elementMainTable[i] = elementMainTable[i]+key;  // scrivo il carattere in input da tastiera
aggiungendolo alla stringa i-esima dell'array
      }
    }
  }
  // controllo se è stata attivata la tabella "Lunghezza e inclinazone tratti in X"
  if(button[0] == 1)
  {
    // ciclo che scorre tutti gli elementi dell'array nLineElementX[] in cui vengono salvati tutti i dati
contenuti all'interno di ogni casella della tabella
    for(int i = 0; i < nLineElementX.length; i++)
    {
      // controllo se la casella i-esima della tabella è attiva consentendo così la scrittura all'interno di
essa
      if(elementTable2[i] == 1)
      {
        // controllo se viene permuto il tasto BACKSPACE (cancella carattere)
        if(key == BACKSPACE) {
          // controllo se la lunghezza della stringa è maggiore di zero
          if (nLineElementX[i].length() > 0) {
            // cancello l'ultimo carattere della stringa troncando la sua lunghezza di 1
```

```
          nLineElementX[i] = nLineElementX[i].substring(0, nLineElementX[i].length()-1);
         }
        }
        // controllo il testo può essere contenuto nella finestra
        else if(textWidth(nLineElementX[i]+key) < width)  {
         nLineElementX[i] = nLineElementX[i]+key;  // scrivo il carattere in input da tastiera
aggiungendolo alla stringa i-esima dell'array
        }
       }
     }
   }
   // controllo se è stata attivata la tabella "Lunghezza e inclinazone tratti in Y"
   if(button[1] == 1)
   {
    // ciclo che scorre tutti gli elementi dell'array nLineElementY[] in cui vengono salvati tutti i dati
contenuti all'interno di ogni casella della tabella
    for(int i = 0; i < nLineElementY.length; i++)
    {
     // controllo se la casella i-esima della tabella è attiva consentendo così la scrittura all'interno di
essa
      if(elementTable3[i] == 1)
      {
       // controllo se viene permuto il tasto BACKSPACE (cancella carattere)
       if(key == BACKSPACE) {
        // controllo se la lunghezza della stringa è maggiore di zero
        if (nLineElementY[i].length() > 0) {
         // cancello l'ultimo carattere della stringa troncando la sua lunghezza di 1
         nLineElementY[i] = nLineElementY[i].substring(0, nLineElementY[i].length()-1);
        }
       }
       // controllo il testo può essere contenuto nella finestra
       else if(textWidth(nLineElementY[i]+key) < width)  {
        nLineElementY[i] = nLineElementY[i]+key;  // scrivo il carattere in input da tastiera
aggiungendolo alla stringa i-esima dell'array
       }
      }
    }
   }
   // controllo se è stata attivata la tabella "Posizione e spessore montanti in X"
   if(button[2] == 1)
   {
    // ciclo che scorre tutti gli elementi dell'array strutElementX[] in cui vengono salvati tutti i dati
contenuti all'interno di ogni casella della tabella
    for(int i = 0; i < strutElementX.length; i++)
    {
     // controllo se la casella i-esima della tabella è attiva consentendo così la scrittura all'interno di
essa
      if(elementTable4[i] == 1)
      {
       // controllo se viene permuto il tasto BACKSPACE (cancella carattere)
       if(key == BACKSPACE) {
        // controllo se la lunghezza della stringa è maggiore di zero
```

```
          if (strutElementX[i].length() > 0) {
            // cancello l'ultimo carattere della stringa troncando la sua lunghezza di 1
            strutElementX[i] = strutElementX[i].substring(0, strutElementX[i].length()-1);
          }
        }
        // controllo il testo può essere contenuto nella finestra
        else if(textWidth(strutElementX[i]+key) < width)  {
          strutElementX[i] = strutElementX[i]+key;  // scrivo il carattere in input da tastiera
aggiungendolo alla stringa i-esima dell'array
        }
      }
    }
  }
  // controllo se è stata attivata la tabella "Posizione e spessore montanti in Y"
  if(button[3] == 1)
  {
    // ciclo che scorre tutti gli elementi dell'array strutElementY[] in cui vengono salvati tutti i dati
contenuti all'interno di ogni casella della tabella
    for(int i = 0; i < strutElementY.length; i++)
    {
      // controllo se la casella i-esima della tabella è attiva consentendo così la scrittura all'interno di
essa
      if(elementTable5[i] == 1)
      {
        // controllo se viene permuto il tasto BACKSPACE (cancella carattere)
        if(key == BACKSPACE) {
          // controllo se la lunghezza della stringa è maggiore di zero
          if (strutElementY[i].length() > 0) {
            // cancello l'ultimo carattere della stringa troncando la sua lunghezza di 1
            strutElementY[i] = strutElementY[i].substring(0, strutElementY[i].length()-1);
          }
        }
        // controllo il testo può essere contenuto nella finestra
        else if(textWidth(strutElementY[i]+key) < width)  {
          strutElementY[i] = strutElementY[i]+key;  // scrivo il carattere in input da tastiera
aggiungendolo alla stringa i-esima dell'array
        }
      }
    }
  }
}


/* Funzione overRect()
  Funzione che accetta in ingresso 4 valori che nello specifico indicano:
   - x: coordinata X del rettangolo nella finestra;
   - y: coordinata Y del rettangolo nella finestra;
   - w: larghezza rettangolo;
   - h: altezza rettangolo.
  La funzione seguente restituisce true se il mouse indica un pixel interno al rettangolo oppure false
se è esterno
*/
```

```
boolean overRect(int x, int y, int w, int h)
{
  if(mouseX < x + w && mouseX > x && mouseY < y + h && mouseY > y)
    return true;
  return false;
}
```

# BIBLIOGRAFIA


1. G. Chitarin, P. Agostinetti, D. Aprile, N. Marconato, and P. Veltri, Improvements of the Magnetic Field Design for SPIDER and MITICA Negative Ion Beam Sources, in “AIP Publishing”, 16/04/2015
2. G. Chitarin, A. Gallo, Design of a structure with 3 linear degrees of freedom for magnetic fields measurements, Technical Note Consorzio RFX, 26/05/2010
3. www.maffucci.it
4. www.arduino.cc
5. https://forum.arduino.cc
6. howtomechatronics.com
7. https://processing.org
8. https://forum.processing.org

# RINGRAZIAMENTI

Desidero ringraziare vivamente il professore, nonché relatore, Giuseppe Chitarin per la sua disponibilità e per l'opportunità che mi ha dato. La sua guida e il suo supporto sono stati essenziali.
Un ringraziamento sentito va anche al tecnico di laboratorio Roberto Losco, che mi ha aiutato e guidato soprattutto nella parte pratica del progetto.

Ringrazio inoltre tutta la mia famiglia per essermi stata vicina in questo percorso a volte tortuoso e per avermi sempre incorraggiato.
Un ultimo ringraziamento va a Marta, il suo sostegno in tutti questi anni è stato per me davvero importante.