

Disegnatore Remoto

Un sistema che permette la creazione di disegni digitali
e la loro stampa su carta



Autore e Creatore: **Marco Belli - IV L**
Docenti: prof. Bergomi, prof. Rusconi, prof. Duse

Premessa

Mi ha sempre affascinato il potere che abbiamo al giorno d'oggi di ottenere ogni tipo di bene o servizio interagendo tramite un semplice click con macchinari molto complessi.

Un classico esempio di questo livello di automatizzazione sono le stampanti. Mai nella storia dell'umanità è stato così facile ottenere una copia esatta di un disegno su un foglio di carta. Questo grazie al progressivo aumento dell'efficienza in sé del processo produttivo, ma anche allo sviluppo di interfacce sempre più intuitive da utilizzare per interagire con il programma.

Ho quindi voluto provare a creare questo tipo di macchina cogliendo nel frattempo l'opportunità di apprendere una vastissima gamma di conoscenze.

Abstract

In questo documento espongo ogni aspetto che ha avuto un ruolo rilevante nel costruire un robot funzionante, capace di muovere un qualsiasi utensile per la scrittura su un foglio di carta secondo le istruzioni degli utenti che interagiscono con un sito internet dedicato.

Inizio descrivendo la porzione puramente meccanica e elettronica del robot, in poche parole l'hardware. In questa sezione si trovano quindi i motori, le componenti meccaniche, le parti stampate in plastica, i circuiti, i microcontrollori e l'uso di software CAD per la rappresentazione digitale dell'intero progetto. Più generalmente, mostro in che modo il robot interpreta le informazioni che riceve tramite comunicazione remota, per riuscire a muovere la penna sul foglio di carta.

Come queste informazioni sono generate e messe a disposizione del microcontrollore del robot è l'argomento del secondo capitolo, ovvero il software. In questa sezione espongo tutto ciò che avviene in sottofondo, per permettere a chiunque di caricare una pagina internet su un computer o dispositivo mobile per interagire con il robot. Non ho solo messo a disposizione dell'utente una serie di comandi per controllare direttamente i motori, ma ho voluto creare un intero ambiente di lavoro, dove è possibile creare, salvare, modificare e stampare disegni digitali nelle diverse pagine interne. In questo modo il funzionamento del robot non è concettualmente diverso da una comune stampante: premendo un bottone l'utente è in grado di comunicare alla macchina di replicare un file digitale su carta.

Indice

1 Come è fatto

1.1 Il robot in sé	3
1.2 Stampante 3D	5
1.3 Modulo ESP8266-12E	5
1.4 Motori passo-passo e DRV8825	6
1.5 Servomotore e convertitore Buck Boost	6
1.6 Server	7
1.7 Sito internet	7

2 Come funziona

2.1 Informazioni generali	8
2.2 Codice server-side	9
2.3 Codice client-side	10
2.4 Creare	12
2.5 Galleria	13
2.6 Software di scrittura, introduzione	15
2.7 Trasmettere	16
2.8 Stampare	17
2.9 Software di scrittura, in dettaglio	19
2.10 Algoritmo del microcontrollore	21
2.11 Driver dei motori	24
2.12 Elettronica	25
2.13 Riempire una figura	26
2.14 Scansionare un'immagine	31
2.15 Calcolo della varianza	36
2.16 Ordine ottimale delle lettere	37

3 Diario

3.1 prime idee	39
3.2 prototipo estivo	40
3.3 uso dell'arduino	42
3.4 server e sito internet	43
3.5 possibili innovazioni	44
3.6 problemi riscontrati	44
3.7 Conclusione	44

Bibliografia	45
--------------	----

Allegati	45
----------	----

Com'è fatto

1.1 Il robot in sé

Per quanto riguarda l'hardware, ci sono diversi requisiti tecnici che la struttura del robot deve soddisfare per risultare in un dispositivo utilizzabile, duraturo e sufficientemente preciso.

Prima di tutto il foglio A4 su cui la penna dovrà scrivere deve restare perfettamente fermo. Può sembrare un punto banale, ma bisogna tenere in considerazione il fatto che la situazione non è analoga alla scrittura a mano a cui siamo tutti abituati. Mentre scriviamo abbiamo sempre una mano che tiene fermo il foglio di carta. Ma questo non è fattibile nel tipo di robot che ho costruito. La penna è l'unico oggetto che viene a contatto con il foglio, quest'ultimo quindi deve essere tenuto fermo con una cornice. Questa è di certo la soluzione più semplice, ma non perfetta. Il foglio non è tenuto in tensione da niente in particolare, ed è quindi potenzialmente soggetto a piegamenti in caso la pressione della penna sia troppo alta. È quindi importante che il foglio stia perfettamente nella cornice, cosa già non facile essendo la cornice costruita a mano, ma deve anche essere integro, ovvero non deve essere mai stato piegato in alcun modo, perché ciò ridurrebbe ampiamente la resistenza che è in grado di mostrare al movimento della penna.

Questa cornice è ciò che costituisce la base del robot. L'ho costruita da un asse di legno poco più grande di un foglio A4, come è visibile dalla foto.



I motori che permettono il movimento della penna sono posizionati in modo tale da coprire l'intera superficie del foglio A4. Il robot è quindi analogo a una macchina CNC con 2 gradi di libertà. In una tale macchina i due assi per il movimento sono convenzionalmente posizionati perpendicolarmente l'uno rispetto all'altro, e paralleli ai margini della superficie che si vuole coprire. Ho però voluto diversificare il mio progetto dalla maggior parte di questi macchinari, che usano indubbiamente il design più affidabile, ma che personalmente ritengo un po' monotono. Il mio robot infatti è in grado di muovere la penna grazie al controllo di coordinate polari, anziché che cartesiane. Un motore rappresenta il centro del sistema di riferimento polare, e allo stesso tempo controlla la lunghezza del raggio, mentre l'angolo è controllato dal secondo motore, posizionato perpendicolarmente al foglio. Queste lunghezze sono modificate grazie a delle viti di controllo, che trasferiscono il movimento rotatorio del motore all'impalcatura che sostiene la penna.

Questo particolare design è stato frutto di un'importante evoluzione che ha avuto il progetto dopo le vacanze estive. Infatti, se comparato al primo prototipo, descritto in dettaglio nel capitolo 3.2, i vantaggi sono evidenti.

La vite di controllo della lunghezza del raggio è collegata alla penna con le 3 aste di ottone, il materiale più leggero disponibile. Queste trasferiscono con molta precisione il movimento del bullone sulla vite di controllo alla penna. Tutte queste componenti in movimento sono fissate grazie a viti e bulloni di dimensione M3 montati su delle parti di plastica personalizzate. La vite di controllo dell'angolo fa invece parte di un meccanismo più semplice, ma anche meno stabile. Muove direttamente le 2 aste di acciaio, affidandosi alla forza di gravità per tenere fermo e in posizione il bullone che si muove.

La penna è sostenuta da due aste di acciaio. Queste devono supportare il peso di una vite di controllo, di 3 aste di ottone più piccole e del meccanismo che muove la penna. In tutto il peso da sostenere è quasi 500g, distribuito su una lunghezza di circa 70cm. Per questo l'acciaio è necessario, e materiali di sostegno come la plastica o il legno non sono delle opzioni da prendere in considerazione.

Il meccanismo che permette alla penna di muoversi grazie al movimento di un servomotore è creato con una stampante 3D. Non supporta solo 2 possibili stati, alzato e abbassato, bensì è possibile regolare la pressione desiderata tra la penna e il foglio. Ciò tramite una piccola molla collegata al motore. È però importante sapere che, non essendo il motore abbastanza preciso, né il movimento della molla lineare, la pressione non è calcolata matematicamente. Lo spettro di pressioni configurabili non è quindi continuo, ma discreto, grazie a numerosi scalini tra il minimo e il massimo che sono stati pre-configurati nel software. La molla che crea la tensione è collegata alla penna, la quale è vincolata alla sola direzione verticale di movimento.



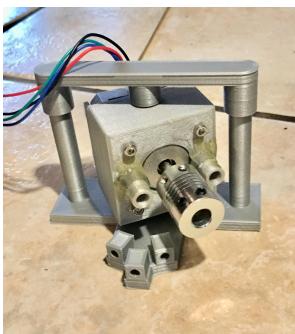
Inoltre il meccanismo che crea tensione sulla penna è smontabile, e permette di sostituire l'oggetto che si vuole utilizzare per la scrittura. Ad esempio potrebbe essere utile voler usare diversi colori o diversi tipi di matite per il disegno.

Un altro punto importante per la precisione del robot è la stabilità dei motori rispetto al foglio di carta. Ovviamente essi devono restare immobili, però devono anche trovarsi nella posizione giusta. Il software calcola il movimento basandosi su una mappa precisa del robot. Lavorando nel sistema di coordinate polari, se ad esempio il motore che controlla il raggio è spostato di solo qualche millimetro dal modello ideale programmato nel software, è visibile ad occhio nudo l'imprecisione che ne risulta. Le lineerette disegnate dal robot in particolare appaiono distorte, ma in compenso i disegni curvi restano praticamente intatti. Ma la precisione al millimetro è molto difficile da ottenere su questo particolare design, e rappresenta quindi uno dei punti deboli più grandi. Questo effetto è comunque mitigato da delle aste che collegano la base al motore del raggio, che oltre a tenerlo fermo forniscono informazioni sul posizionamento corretto.

Una caratteristica che forse contribuisce a questa insufficiente stabilità è la smontabilità. Mentre non è in uso, l'intera struttura può essere smontata per occupare meno spazio. Le estremità della cornice possono essere piegate. Ogni asta e vite di comando può essere conservata a parte, mentre ogni altro componente può essere disposto in modo tale da entrare in uno zaino.

1.2 Stampante 3D

Molte componenti sono state create usando la stampante 3D del liceo grazie all'aiuto del prof. Bergomi. In questo modo è possibile progettare delle parti personalizzate con tutte le caratteristiche desiderate. Sono di plastica le strutture in cui entrano i motori, e che



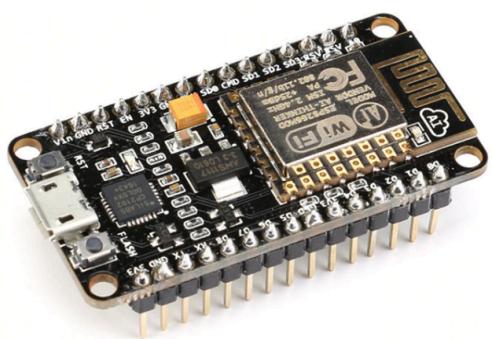
permettono la regolazione della posizione del motore del raggio. Questo tramite una serie di buchi in cui passano le aste di ottone esterne. Inoltre 3 componenti in plastica sono fondamentali per garantire uno scorrimento e movimento liscio delle aste che sostengono la penna. Ma la componente che rende l'uso della stampante 3D indispensabile è sicuramente quella che tiene la penna in posizione, e permette lo spostamento verticale.

Queste parti sono state create tramite il programma Autodesk Fusion 360, che permette di controllare diverse macchine CNC e stampanti 3D. Nelle immagini indicate è possibile vedere più chiaramente il tipo di parti che ho dovuto stampare per questo progetto. Inoltre in allegato ci sono tutti i file delle parti 3D rilevanti al progetto, nel formato "stl" leggibile solo dalle applicazioni CAD come Fusion.



1.3 Modulo ESP8266-12E

Questo modulo è una scheda elettronica che funge da microcontrollore con un modulo Wi-Fi integrato. Il mio progetto è costruito attorno a questo comune modulo Wi-Fi, visto che non sono necessari molti collegamenti elettronici né prestazioni elevate, con esso è possibile controllare i due motori passo-passo e il servomotore. Questo microcontrollore inoltre permette la comunicazione Wi-Fi, che apre moltissime nuove possibilità per il progetto. Il vantaggio più grande che porta la comunicazione remota è di ridurre al minimo le operazioni da svolgere sul microcontrollore. Infatti tutto ciò che svolge il modulo Wi-Fi è aggiornarsi alle informazioni che riceve dal server a cui si collega, e mandare gli appropriati segnali elettronici ai driver dei motori passo-passo e al servomotore.



1.4 Motori passo-passo e DRV8825

Come tutte le macchine CNC, il mio robot necessita di un elevato livello di precisione. E questo dipende non solo dalla stabilità delle componenti del robot, ma anche in maggior parte dal tipo di motori che vengono utilizzati. I motori che indubbiamente svolgono questo compito al meglio sono i motori passo-passo. Grazie al modo in cui vengono controllati, forniscono un altissimo livello di precisione, almeno in teoria.

Grazie a questi motori il robot può muovere la penna con una teorica precisione di:

$$\frac{0.9 \text{ deg}}{1 \text{ passo}} \cdot \frac{1 \text{ riv}}{360 \text{ deg}} \cdot \frac{2 \text{ mm}}{1 \text{ riv}} = 0.005 \frac{\text{mm}}{\text{passo}},$$

che ovviamente però resta sulla carta. Per il controllo dei due gradi di libertà questi motori sono indubbiamente la migliore e unica opzione. Lo svantaggio dei motori passo-passo però sta nel fatto che non basta semplicemente collegarli alla corrente per azionarli. A differenza di un classico motore DC, il motore passo-passo deve ricevere corrente in un determinato ordine tra tutti i suoi quattro fili e non si può semplicemente collegare alla corrente. Ma per semplificare molto il controllo è possibile comprare un “driver” per dei motori passo-passo.

Questi sono piccole schede elettroniche che permettono il controllo dei motori tramite singoli segnali elettronici intermittenti, e fungono inoltre da ponti H per cambiare la direzione di rotazione. Per il mio robot ho usato i driver “DRV8825”, che supportano un sufficiente amperaggio e costano poco più di 1Fr, non sono quindi della massima qualità. In questo modo il microcontrollore manda segnali elettronici a intervalli calcolati per ottenere lo spostamento desiderato della penna.



1.5 Servomotore e convertitore Buck Boost

Il voltaggio operativo del servomotore è 4.8V, mentre il voltaggio fornito dal microcontrollore è al massimo 3.3V, e dopo numerosi esperimenti ho concluso che il segnale PWM creato è troppo debole per controllare il servomotore in modo affidabile. Per questo uso un piccolo convertitore Buck Boost che aumenta il voltaggio di un segnale da 3.3V a 5V.

Questo modulo è importante soprattutto per usare un servomotore più grande, che al segnale PWM di 3.3V del microcontrollore proprio non risponde. Infatti in certe situazioni il servomotore piccolo è troppo debole per applicare sufficiente forza sulla molla collegata alla penna, ed è quindi necessario usare un servomotore con momento torcente più alto. La componente che tiene il servomotore piccolo già mostrata nel capitolo 1.2 può essere sostituita con un altro pezzo simile, progettato per un servomotore grande, fornendo più momento torcente, precisione e affidabilità.



1.6 Server

Finora ho parlato di come tutte le parti del robot contribuiscano al movimento corretto della penna, che è infine l'unico vero obiettivo che il robot deve raggiungere. Ma chi crea e rende disponibili le informazioni sul movimento da seguire?

Per utilizzare la comunicazione remota, c'è bisogno di una richiesta e risposta di informazioni. Il modulo Wi-Fi svolge il ruolo del richiedente, chiamato "client", mentre l'entità che risponde è chiamata "server", perchè appunto serve le informazioni al richiedente. Il controllo del robot è infatti basato su questo scambio di informazioni tra il microcontrollore e il server. Per questo progetto ho creato il server utilizzando NodeJS.

NodeJS è una runtime di JavaScript che permette l'esecuzione di codice in questo linguaggio dalla parte del server. Creare un programma sul computer in questo modo diventa piuttosto semplice. Una volta installato NodeJS, dal terminale si può creare un nuovo programma, che all'avvio esegue un file JavaScript. Ma la vera potenzialità nasce dalle numerosissime librerie pre-installate e create dalla "community", che però in questo contesto vengono chiamate "moduli". Uno di questi moduli per NodeJS è "http-server", che permette, con poche linee di codice, di creare un server sul computer e servire file ai client che ci si connettono.

Altri moduli che vengono importati, non disponibili online, sono scritti da me, come "WritingSoftware" e "PathVariance". Il motivo per cui ho deciso di dividere il programma in più file, scrivendo queste porzioni di codice in moduli indipendenti dal file principale, lo discuto in dettaglio nel capitolo 3.4.

Una volta avviato il programma dal terminale, il codice viene eseguito e il server viene creato. Finché esso viene terminato, qualunque dispositivo con una connessione Wi-Fi può fare richieste http all'indirizzo IP configurato.

Il server quindi non è costantemente online. Per utilizzare il robot è necessario possedere un computer e avviare il server localmente.

1.7 Sito Internet

Usando un controllo remoto tramite il modulo Wi-Fi, l'unico modo per comunicare un'azione dell'utente al microcontrollore è tramite il server. Per interagire con il server, bisogna creare un sito internet, quindi creare un file html, che viene servito al client e aperto nel browser durante il collegamento all'indirizzo IP. Il file html contiene gli elementi del sito internet che si vedono nella finestra del browser. Ci sono altri file che completano la pagina assieme all'html, e questi sono il css, che definisce l'aspetto estetico della pagina html, e il javascript, che definisce le risposte ai diversi eventi che possono accadere mentre la pagina è aperta.

Nel mio progetto ci sono numerose pagine internet che svolgono diverse funzioni, mandando e ricevendo diverse informazioni dal server; tutti questi dettagli verranno trattati nei prossimi capitoli.

Una volta ricevuto e caricato il sito internet, l'utente può interagirci, e il codice javascript in sottofondo risponde alle interazioni, ad esempio trasmettendo le informazioni del movimento della penna al server. In questo modo il server riceve informazioni dall'utente, e quando il modulo Wi-Fi si accende richiede queste informazioni per muovere i motori adeguatamente.

Come funziona

2.1 Informazioni Generali

Lo scopo ultimo del mio progetto è riuscire a creare delle informazioni che rappresentano un percorso su due dimensioni, e poi replicare questo percorso sul foglio di carta con la penna controllata dal modulo Wi-Fi. L'intero sistema è costruito attorno a questo concetto di "percorso", che è per forza creato da un insieme di punti ordinati. Per semplificare la terminologia nel codice, cosa assolutamente necessaria per un progetto di questa magnitudine, utilizzo un'analogia grammaticale per riferirmi a questi insiemi di punti. Due coordinate rappresentano un "punto", un insieme di punti ordinati rappresenta una "lettera", e un insieme di lettere rappresenta una "frase". Le diverse lettere nella frase non sono collegate, quindi mentre il robot sta replicando la frase sul foglio di carta deve alzare e abbassare la penna quando passa da una lettera all'altra. La frase inoltre contiene tutti i punti, ed è l'oggetto che deve essere inviato al microcontrollore. Ogni lettera inoltre possiede una serie di attributi che definiscono come deve essere replicata. Ad esempio l'attributo "press", che indica la pressione che il servomotore deve applicare mentre questa viene scritta, oppure l'attributo "shape" che determina se il percorso che definisce la lettera deve essere trattato come il bordo esterno di una figura da riempire. Nel codice JavaScript la frase è un array di lettere, le lettere sono array di punti e i punti sono oggetti con attributi X e Y. Per semplicità infatti il codice lavora con le coordinate cartesiane dei punti, che vengono solo trasformate in polari prima di essere mandate al microcontrollore.

L'intero progetto è avviato dalla cartella "LocalServer", in essa ci sono tutti i file utilizzati. Il file "server.js" è il codice eseguito all'avvio del programma, la sottocartella "server-files" contiene tutto ciò che riguarda il codice server-side e, analogamente, la sottocartella "Website" contiene tutto ciò che concerne il codice e i file utilizzati client-side. Gli altri file in questa cartella servono per il corretto funzionamento di NodeJS nell'esecuzione del programma, e quindi possono semplicemente essere ignorati.

Premetto che per spiegare con più chiarezza il codice magari non sarò sempre preciso con i termini. Ad esempio, mentre parlo dei comandi che il client può inviare al server, in realtà si tratta di una richiesta http-get a un particolare indirizzo, che dalla parte del server risulta nell'esecuzione di una particolare funzione o algoritmo.

Tutto quello che riguarda il software, come i nomi dei file, i commenti e le variabili, è scritto in lingua inglese. Questo per mantenere un certo livello di continuità nel codice, ed evitare nomi in italiano che suonano semplicemente troppo innaturali. Inoltre ogni documentazione e tutorial che ho seguito per imparare ogni conoscenza che mi è servita era in lingua inglese, e questo ha sicuramente influenzato la mia scelta dei nomi.

Ho deciso di trascurare la spiegazione del codice css, perché nonostante componga una parte importante e molto vasta, contando le linee di codice, del sito internet, non influisce in alcun modo sulla funzionalità del robot. Questo include tutto ciò che riguarda gli elementi html in sé che si trovano nelle pagine internet, come il navigatore estendibile, la galleria, il controllo dell'ingrandimento e spostamento della tela, il feedback visivo dei bottoni e altro.

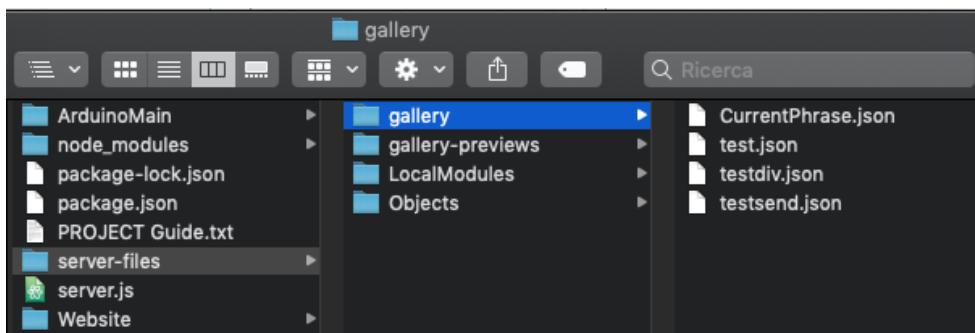
2.2 Codice server-side

All'avvio del server la prima operazione che viene eseguita è l'importazione dei moduli NodeJS. Quelli essenziali sono "http-server", per creare il server locale, e "fs", che permette la scrittura e lettura di file nel computer. Altri moduli che utilizzo sono "express", che facilita la creazione del server, e "body-parser", che facilita la lettura del contenuto delle richieste http-post.

A questo punto il server viene creato, e a qualunque client si colleghi all'indirizzo IP viene servita la pagina "index.html". In realtà l'unica azione che esegue questo file è reindirizzare il client a "HomePage.html", quindi è in sé vuoto, ma è necessario perché express, il modulo NodeJS, cerca il primo file denominato "index.html", proseguendo a servirlo. Questo però è solo un cavillo tecnico che non influenza alcuna funzionalità.

Mentre il programma è in esecuzione, il server potrebbe avere a che fare con più di una frase nel tempo. Le frasi non sono memorizzate sotto forma di variabile, ma sono salvate sotto forma di file "json" nella cartella "server-files". Questo fornisce la possibilità di salvare certe frasi, cosicché anche al riavvio del programma esse non vengano eliminate come tutte le altre informazioni sotto forma di variabile nel codice.

Nella cartella "gallery" si possono trovare questi file che fanno riferimento alle frasi salvate in precedenza. Aprendo questi file è inoltre possibile leggere direttamente tutte le informazioni che definiscono le frasi.

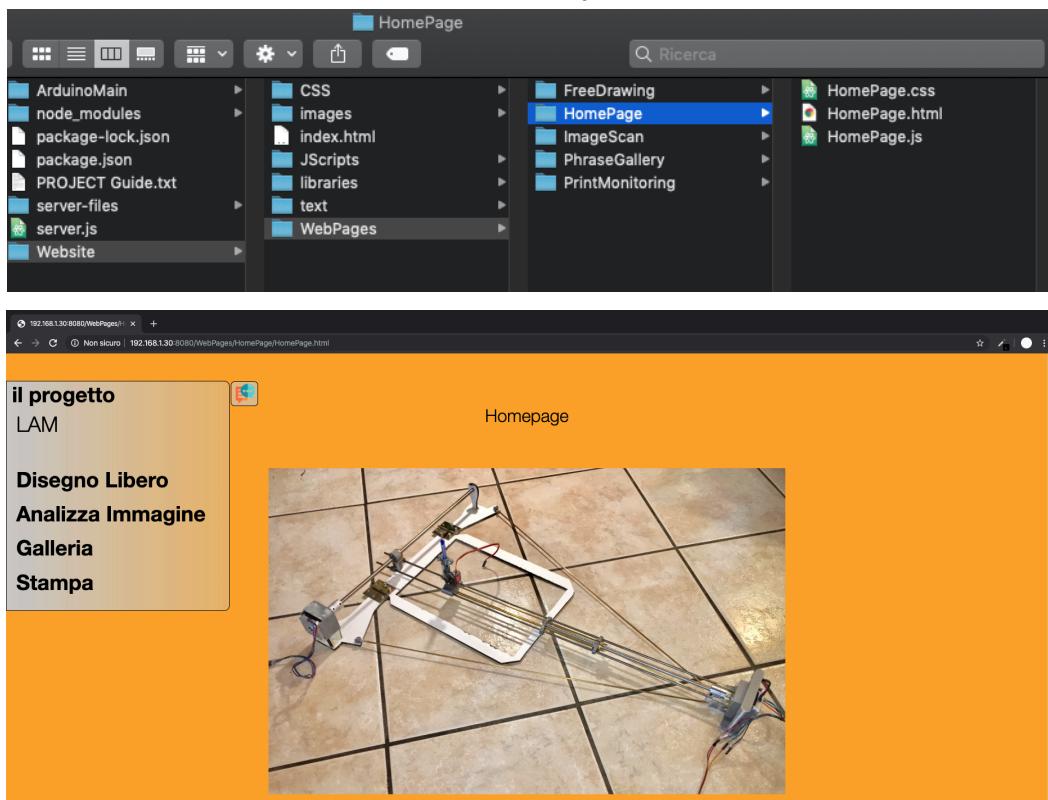


È importante capire il concetto di "frase corrente", ovvero una variabile chiamata "CurrentPhrase" che rappresenta la frase attualmente in uso. Appena avviato il programma, viene letto il file "CurrentPhrase.json", e l'oggetto che ne risulta è assegnato alla variabile. Ogni volta che l'utente della pagina web modifica in qualche modo l'insieme di punti, nel server viene modificata la variabile "CurrentPhrase". È quindi possibile pensare a questa variabile come una frase temporanea, non ancora salvata da nessuna parte. E infatti solo quando il server riceve il comando "SaveCurrent", crea un file "json" con un particolare nome, che contiene la frase corrente, salvandola quindi permanentemente. Analogamente, quando riceve il comando "EraseCurrent", la variabile "CurrentPhrase" viene reinizializzata e le viene assegnato il valore del file "BlankPhrase.json", che si trova nella cartella "Objects". Questo è il modo in cui il server interagisce con i client, ovvero quando riceve una richiesta http-get a un particolare indirizzo, esegue le operazioni che corrispondono al comando desiderato.

2.3 Codice client-side

Nel momento in cui si ricerca un URL nel browser, il client fa una richiesta all'indirizzo IP digitato, e aspetta una risposta dal server. Nel caso del mio progetto viene servito il file "HomePage.html", che corrisponde alla pagina iniziale. Nella cartella "Website" si trovano tutti i file che potrebbero essere utilizzati dalle diverse pagine, e nella sottocartella "WebPages" si trovano tutte le pagine.

Si vede in questa cartella che "HomePage" è solo una delle numerose pagine web che ho creato, tra le quali si può navigare premendo sui diversi link nella pagina iniziale. Nelle cartelle corrispondenti si trovano i relativi file html, js e css.



Ho usato principalmente 2 librerie per il codice javascript client-side, "jQuery" e "p5". jQuery facilita molto il controllo degli elementi html tramite javascript, ma non è indispensabile. p5 invece è il cuore della programmazione javascript client-side. Con p5 è molto semplice creare un elemento html chiamato "canvas", sul quale è possibile eseguire il codice "setup" e "draw". Al caricamento del canvas, la funzione setup è eseguita e successivamente la funzione draw viene chiamata in loop.

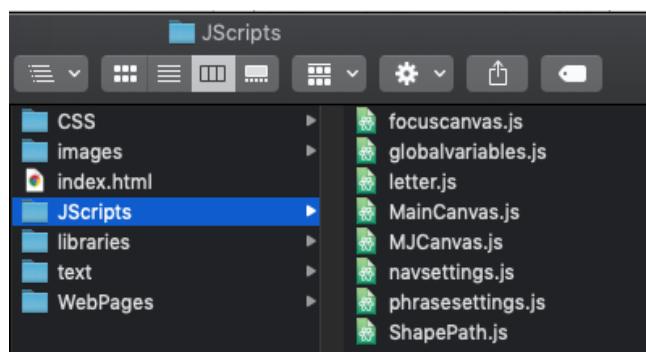
Questa funzionalità è principalmente usata per creare certi tipi di animazione. Ad esempio le funzioni più utili sono: "background" per colorare l'intero canvas, "ellipse" per creare un'ellisse a coordinate e grandezza configurabile, "fill" per impostare il colore della prossima figura creata, "line" per creare una linea con due punti. Inoltre in p5 è integrata l'interazione tra gli utenti e il canvas, con funzioni e variabili come "mousePressed", "mouseReleased", "mouseX", "mouseY".

Un dettaglio importante riguardante p5 nelle pagine web è il fatto che non lo utilizzo in modalità “globale”, ma nella cosiddetta “instance mode”. In sintesi, usare la modalità globale vuol dire che nel codice JavaScript il canvas e tutte le funzioni di p5 sono variabili globali, non c’è quindi bisogno di fare un riferimento esplicito all’oggetto p5, mentre in “instance mode”, questo non è il caso. Bisogna creare prima un oggetto p5 con un nome per il contesto appena creato, come “ctx”, e all’interno della funzione per crearlo scrivere il codice, facendo riferimento al contesto per usare le funzioni p5 in particolare:

```
var p5object = new p5(function(ctx){ ctx.ellipse() });
```

Questo è il motivo per cui il codice è pieno di riferimenti all’oggetto “ctx”, e spesso nelle funzioni una delle variabili è ctx, perché è necessario fare riferimento al contesto del canvas che si sta controllando. È necessario usare questa modalità se si vuole creare più di un canvas sulla stessa pagina.

Spesso ci sono molte caratteristiche comuni attraverso diverse pagine. Ad esempio per quanto riguarda il CSS: il navigatore estendibile, la messa in scala automatica del canvas e attributi generali per mantenere un livello di continuità grafica tra ogni pagina. Ma anche molte funzioni JavaScript che sono usate dappertutto e non solo su una pagina specifica, come: manipolazioni della frase corrente sul server, interazione con il navigatore espandibile, l’oggetto “lettera”. Per questo ci sono le cartelle CSS e JScripts, che contengono numerosi file utili a tutte le pagine. Queste cartelle infatti contengono codici che possono essere considerati librerie, in quanto vengono inclusi nelle pagine che ne hanno bisogno. E questo aiuta sia con una struttura più comprensibile del codice, ma soprattutto per l’espandibilità.



Infatti creare una nuova pagina è relativamente semplice perché la maggior parte delle funzionalità potenzialmente utili si trovano in queste librerie, che sono solo da importare.

Ad esempio, il file “phrasesettings.js” contiene tutte le funzioni per comunicare al server una modifica della frase corrente o di una frase salvata. La funzione “AddLetter” invia al server, tramite una richiesta http-post, l’ultima lettera della variabile “phrase”, e il server risponde aggiornando la frase corrente; la funzione “ErasePhrase” invece segnala al server, tramite una richiesta http-get, di inizializzare la frase corrente.

Un altro esempio di file nella cartella “JScripts” è “MainCanvas.js”. In esso sono già programmate le fondamenta del canvas principale su cui p5 lavora in “instance mode”. Per creare un canvas basta importare il file, e definire le quattro funzioni globali in esso chiamate: ModeVariables(ctx), ModeSetup(ctx), ModeDraw(ctx), ModeFunctions(ctx).

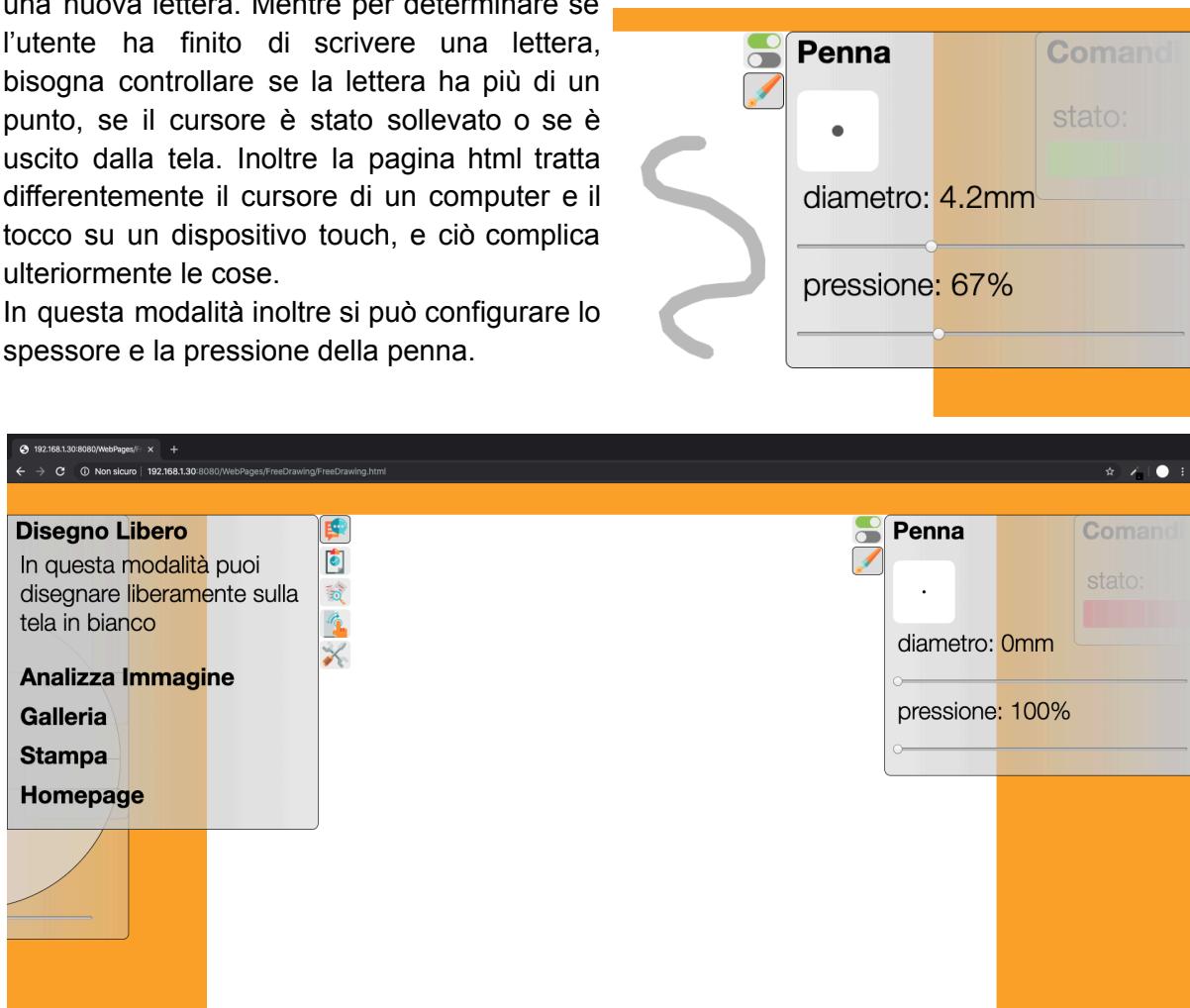
Queste funzioni devono essere create nel contesto globale, cosicché vengano successivamente chiamate da “MainCanvas.js”. Sono l’unica caratteristica che differenzia ogni canvas nelle diverse pagine che ho creato, e sono quindi presenti nelle cartelle della particolare pagina nel file “Mode.js”. L’oggetto p5 “MainCanvas” è infatti uguale per tutte le pagine, e basta importarlo dalla cartella JScripts.

2.4 Creare

La pagina iniziale è in realtà piuttosto semplice, e non possiede funzionalità. Per davvero creare una nuova frase, o modificare quelle salvate, bisogna caricare una pagina che possieda questa funzionalità.

Ad esempio la pagina “FreeDrawing”, è un semplice strumento per disegnare. Trascinando il mouse, o il dito su un dispositivo touch, sulla tela bianca, il codice salva in tempo reale i punti che definiscono il percorso tracciato. Tutto è di nuovo costruito attorno alla variabile “phrase”, che nel caso del codice client-side è un array di oggetti “letter”. Le lettere possiedono diversi attributi, tra cui due array che contengono le coordinate “X” e “Y” dei punti che fanno parte di quella specifica lettera. Nella funzione draw, il codice controlla sistematicamente il movimento del cursore, e se la sua posizione cambia rispetto al punto precedente, un nuovo punto è creato e aggiunto alla lettera corrente. Il meccanismo in sé è semplice, ciò che aggiunge molta complessità invece è riuscire a trattare correttamente i casi limite. Ad esempio, per creare una lettera con un solo punto, bisogna fare un doppio click, altrimenti ogni volta che si schiacciano i bottoni nel navigatore estendibile si creerebbe una nuova lettera. Mentre per determinare se l’utente ha finito di scrivere una lettera, bisogna controllare se la lettera ha più di un punto, se il cursore è stato sollevato o se è uscito dalla tela. Inoltre la pagina html tratta differentemente il cursore di un computer e il tocco su un dispositivo touch, e ciò complica ulteriormente le cose.

In questa modalità inoltre si può configurare lo spessore e la pressione della penna.



2.5 Galleria

Una volta creata una frase, basta digitare un nome e salvarla. Quando si schiaccia l’apposito bottone, il client esegue la funzione “SaveCurrent”, mandando al server un comando per

salvare la frase corrente nella galleria. La galleria è una sottocartella di “server-files”, le frasi salvate vengono depositate qui sotto forma di file json.

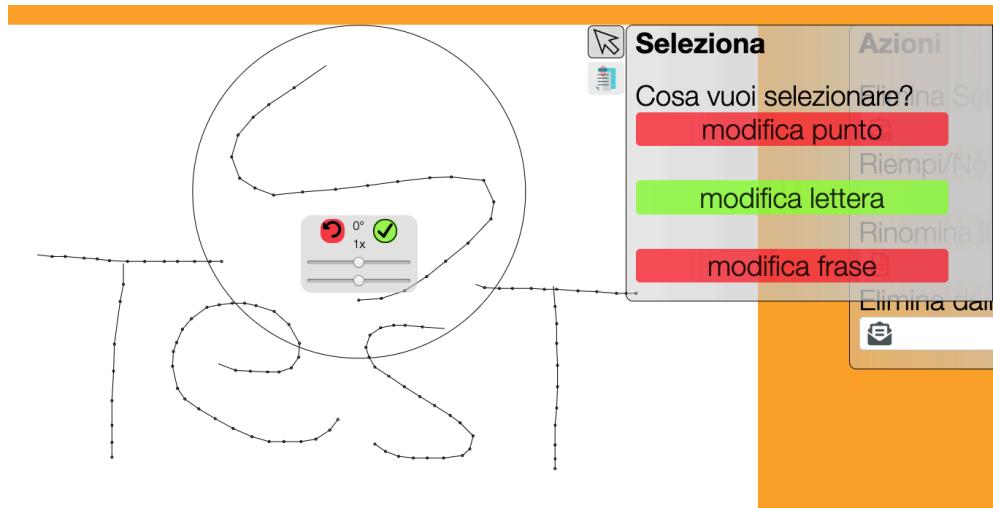


Quando si salva la frase, si crea e si trasmette al server un’anteprima della frase sulla tela. La funzione “SaveCurrent” crea un’immagine della tela, la codifica in base 64 sotto forma di stringa di testo e la invia al server tramite una richiesta http-post. Il server salva questa stringa di testo nella cartella “gallery-previews”, cosicché sia successivamente disponibile.

Questa anteprima è utile per creare una galleria, dove è possibile esplorare le frasi che sono state salvate. Questa galleria in miniatura si trova in una sezione del navigatore estendibile sinistro, e contiene le anteprime con tutte le informazioni relative alle frasi, per rendere la ricerca molto più facile. Basta premere una di queste anteprime per caricare sulla tela la frase desiderata. Ciò è possibile grazie alla funzione “LoadPhrase”, che richiede al server la frase con un particolare nome. La galleria in miniatura è aggiornata quando una frase viene salvata, modificata o eliminata, così da aggiornare anche le anteprime.



Però una vera e propria galleria dovrebbe avere un po' più di funzionalità, per questo un'altra pagina web che ho creato è "PhraseGallery". In questa pagina è possibile modificare le frasi esistenti. Ci sono 3 modalità: punto, lettera e frase. Con la modalità punto attivata, basta premere un punto per spostarlo o eliminarlo. È inoltre possibile eliminare, spostare, ruotare e rimpicciolire o ingrandire la lettera selezionata in modalità lettera o, analogamente, l'intera frase in modalità frase. Il browser tiene sotto forma di variabile l'intera cronologia dei cambiamenti applicati, cosicché se il bottone verde per salvare non è ancora stato premuto si possa tornare indietro.



Il bottone “elimina dalla galleria” risulta nell’eliminazione permanente della frase dalla galleria. Il server elimina i file json della frase e dell’anteprima dalle cartelle. Non è da confondere con “elimina selezionato” mentre è attivata la modalità frase, perché questa azione risulta solo nell’inizializzazione della frase a una frase vuota, mantenendo quindi i file nelle cartelle.

Un’altra opzione è cambiare il livello di pressione e attivare o disattivare il “riempimento” per una certa lettera. Una lettera normale, quindi solo un percorso per la penna, di default non è da riempire. Ma è possibile creare una lettera, che non rappresenta solo un percorso, ma una figura della quale tutta la superficie deve essere coperta dalla penna. Quindi, se una lettera è da riempire, quando il server la manderà al modulo Wi-Fi per stamparla, calcolerà automaticamente un percorso per la penna che riempie di colore l’intera figura. Come questo sia possibile viene approfondito nel capitolo 2.13

2.6 Software di Scrittura, introduzione

Uno dei moduli per NodeJS che ho creato è “WritingSoftware”. L’oggetto JavaScript che risulta dall’importazione di questo modulo viene assegnato alla variabile “WS”, e contiene tutte le informazioni e funzioni per gestire la creazione e la stampa in tempo reale di una frase. Per fare ciò però le informazioni contenute in una frase vengono trasformate più volte, attraverso le numerose funzioni presenti nel software di scrittura.

Il valore delle X e Y presenti nelle frasi corrisponde alla distanza in millimetri nelle 2 dimensioni. Questi valori vengono convertiti in una nuova unità di misura, i “passi”, ovvero quanti passi del motore di distanza da un punto d’origine. Per questa conversione bisogna prima definire certe informazioni tecniche del robot, come la posizione dei motori, il passo delle viti di controllo e le diverse direzioni algebriche positive.

La costante “StepsPerMM” indica quanti passi del motore corrispondono allo spostamento di un millimetro del bullone sulle viti di controllo, e vale $200 \frac{\text{passo}}{\text{mm}}$. Guardando in orizzontale il foglio dall’alto, il vertice in alto a sinistra è l’origine delle coordinate cartesiane, con X positiva verso destra e Y positiva verso il basso. Il centro delle coordinate polari invece è nel punto $PivotX = 297 + 380 \text{ mm}$, $PivotY = 210/2 \text{ mm}$, che è anche la posizione del motore R, con direzione positiva centripeta del raggio e senso antiorario positivo per l’angolo. Il motore A si trova in basso a sinistra, con la vite di controllo parallela all’asse Y e distante $L = 712 \text{ mm}$ dal motore R.

Il valore delle X e delle Y deve quindi essere trasformato in raggio e angolo, e successivamente in passi:

$$\text{cartesiane} \Rightarrow \text{polari} \Rightarrow \text{passi}$$

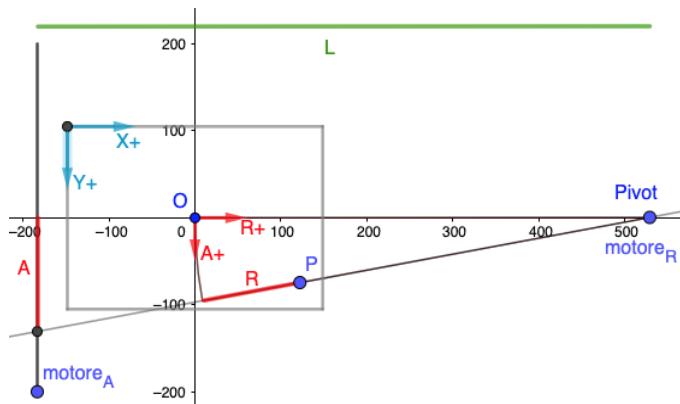
Con R e A che indicano le coordinate di questo nuovo sistema di riferimento, possiamo definire arbitrariamente l’origine nel centro del foglio:

$$O_X = 297/2, O_Y = 210/2 \Rightarrow O_{\text{raggio}} = 528.5, O_{\text{angolo}} = 0 \Rightarrow O_R = 0, O_A = 0$$

Possiamo ricavare la formula per la conversione delle coordinate cartesiane di un qualsiasi punto P in passi, con il teorema di Pitagora per il raggio e un’identità tra triangoli simili per l’angolo:

$$P_{\text{raggio}} = \sqrt{(PivotX - P_X)^2 + (PivotY - P_Y)^2} \quad P_R = (O_{\text{raggio}} - P_{\text{raggio}}) \cdot StepsPerMM$$

$$P_A = \frac{(P_Y - PivotY)}{(PivotX - P_X)} \cdot L \cdot StepsPerMM$$



Con tutte queste informazioni il software di scrittura può convertire con una singola funzione le coordinate. Ad esempio, con $P_X = 285$, $P_Y = 195$, le corrispondenti coordinate in passi del punto P sono $P_R = 25260$, $P_A = 32694$.

Il software di scrittura ha come scopo la creazione di “bersagli”, ovvero oggetti JavaScript con diverse informazioni utili per definire lo spostamento della penna da parte del microcontrollore. Queste informazioni infatti, per il modulo Wi-Fi che le riceve, sono degli obiettivi che deve raggiungere controllando i motori passo-passo e il servomotore, passando quindi da un punto iniziale a uno finale. Le informazioni fondamentali di un bersaglio sono le coordinate di arrivo, chiamate “Rtarg” e “Atarg”, che appunto definiscono verso quali coordinate il microcontrollore deve muovere la penna. Altri attributi sono le coordinate iniziali, il cambiamento delle coordinate dal punto iniziale al punto finale, se la penna deve alzarsi, il tempo che il microcontrollore dovrebbe impiegare a raggiungere il bersaglio, la velocità dei motori in passi al secondo e l’intervallo di tempo in microsecondi tra ogni passo. Tutte queste informazioni sono in unità passi, ma anche in unità cartesiane. Inoltre ci sono gli attributi “Label”, “Letter” e “Point” che danno semplicemente un’etichetta al bersaglio.

La funzione “Setup” nell’oggetto “WS” prende come variabile una frase, e, quando chiamata, calcola tutte le possibili informazioni per i bersagli che saranno potenzialmente creati. Quindi, nel momento che si chiama Setup, nessun bersaglio viene ancora creato, ma vengono solo inizializzate tutte le variabili e calcolata la nuova variabile “Phrase” all’interno di WS.

Nuovi bersagli sono creati solo quando è chiamata la funzione “PushTarget”. Questa funzione calcola tutti gli attributi sopraelencati di un bersaglio, e ritorna un nuovo oggetto, che viene aggiunto alla fine dell’array “TargetQueue”, ovvero coda di bersagli. Questo array all’interno di WS contiene quindi tutti i bersagli che devono ancora essere raggiunti. Infatti ciò che viene inviato al modulo Wi-Fi è il primo bersaglio della coda, che viene successivamente eliminato quando il microcontrollore l’ha raggiunto. In questo modo, mano a mano che il modulo Wi-Fi muove il robot e richiede nuovi bersagli, la coda si accorcia finché non è completamente vuota, condizione che segnala il termine del processo di stampa, ovvero quando non ci sono più bersagli da raggiungere.

2.7 Trasmettere



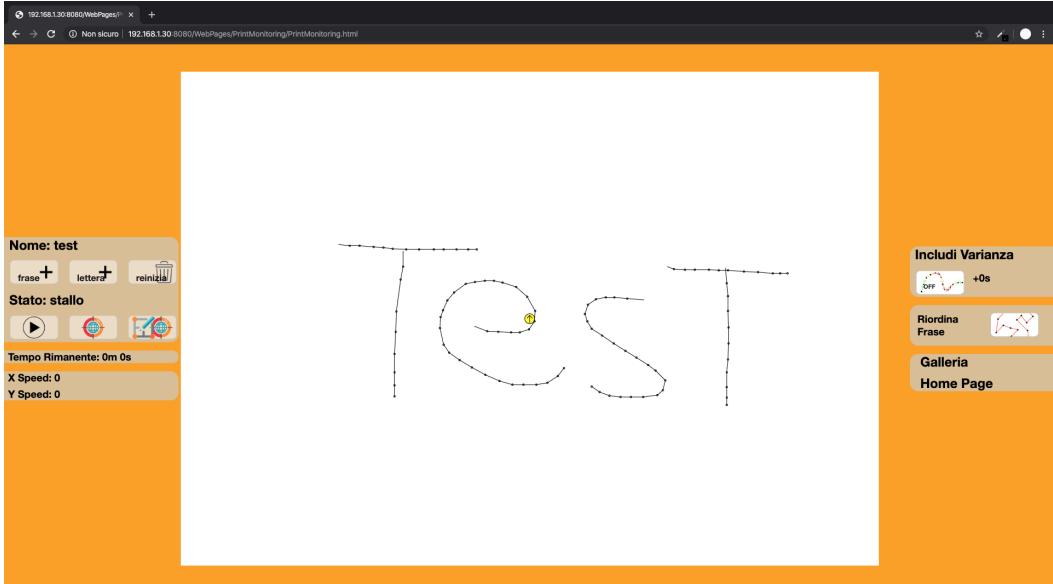
Nel programma il concetto “trasmettere” si riferisce alla preparazione di una particolare frase per essere inviata. Quando da una qualsiasi pagina internet viene premuto il pulsante azzurro invia, la frase non viene direttamente inviata al modulo Wi-Fi. Quando il server riceve questo comando, semplicemente salva la frase corrente in un nuovo file chiamato “TransmissionPhrase.json” all’interno della cartella “Objects”.

Oltre a salvare, il server chiama la funzione del software di scrittura “Setup” con la frase corrente come argomento. Quindi “trasmettere” vuol dire preparare il software di scrittura alla creazione di bersagli da mandare.

Per questo la frase corrente e la frase trasmessa sono due cose diverse, nonostante possano corrispondere allo stesso insieme di punti ordinati; la prima è solo una frase temporanea modificata dall’utente nel sito internet, mentre la seconda viene utilizzata per creare nuovi bersagli da inviare al modulo Wi-Fi.

2.8 Stampare

Una volta trasmessa la frase desiderata, si può iniziare il processo di stampa. Per attivare il robot e iniziare a stampare bisogna raggiungere la pagina internet “Stampa”, nel codice chiamata “PrintMonitoring”.



La funzionalità principale di questa pagina è comandare al server di creare nuovi bersagli, e aggiungerli alla coda. Ad esempio con il bottone “aggiungi frase”, vengono creati bersagli per ogni singolo punto, mentre con “aggiungi lettera” solo per i punti della lettera selezionata. Una volta aggiunti tutti i bersagli desiderati, che sono visibili sulla tela bianca, si può procedere ad attivare il robot. Premendo il bottone “play”, il server cambia lo stato del software di scrittura; alla variabile “IsPrinting” all’interno di WS viene assegnato il valore “true”. Adesso la prossima volta che il modulo Wi-Fi si aggiorna alle informazioni presenti sul server, riceverà in risposta un nuovo bersaglio, che viene rimosso dalla coda.

Sulla pagina internet è possibile seguire in tempo reale i progressi del robot, quindi osservare la sua attuale posizione, assieme ai bersagli ancora in coda che aspettano di essere raggiunti.

Altre funzionalità sono la messa in pausa della stampa, l’inizializzazione forzata della coda di bersagli e il ritorno al centro del foglio. Ma anche attivare certe opzioni che influenzano il modo in cui vengono creati nuovi bersagli, come la diminuzione della velocità in modo proporzionale alla curvatura del percorso, discusso più in dettaglio nel capitolo 2.15.

Il modo in cui la pagina internet aggiorna le informazioni in tempo reale è piuttosto complesso. Entrambi i client, quindi l’utente e il modulo Wi-Fi, possono solo ricevere e mandare informazioni al server, ma non comunicare direttamente tra di loro. Fare costantemente richieste a intervalli di tempo regolari è un uso molto inefficiente ed elevato della rete. Per questo la pagina si aggiorna solo sotto particolari condizioni.

Prima di tutto, con il termine “aggiornarsi” intendo l’esecuzione della funzione “UpdatePoint” all’interno del file JavaScript della pagina. Questa funzione richiede al server l’attuale bersaglio del microcontrollore. In risposta riceve le etichette del bersaglio, le coordinate cartesiane iniziali e finali, le velocità, il tempo per raggiungere il bersaglio e il tempo totale rimanente per l’intera coda.

La pagina fa una previsione sul prossimo istante in cui riaggiornarsi con le informazioni ricevute nell’aggiornamento precedente. Assumendo che dopo il tempo necessario per

raggiungere il nuovo bersaglio, il modulo Wi-Fi ne abbia richiesto uno nuovo, se la pagina si aggiorna dopo lo stesso intervallo di tempo dovrebbe anch'essa ricevere in risposta il nuovo obiettivo. Quindi, ricapitolando, una volta raggiunto un bersaglio la pagina aspetta un opportuno intervallo di tempo prima del prossimo aggiornamento, così da non ricevere più volte le stesse informazioni. In pratica, però, anche in questo modo è possibile che il microcontrollore accumuli nel tempo dei ritardi, e la pagina potrebbe ricevere più volte lo stesso bersaglio, per questo nel codice ho introdotto la variabile booleana "new", che consente l'aggiornamento delle informazioni solo se il bersaglio ricevuto è nuovo.

Per questa ragione la posizione della penna in continuo movimento è un'illusione. Dopo un aggiornamento, il cerchio giallo semplicemente si muove alla stessa velocità della penna secondo le informazioni ricevute dal server, ma non riflette direttamente la posizione di essa.

2.9 Software di Scrittura, in dettaglio

Riassumendo i concetti già spiegati precedentemente, l'oggetto WS contiene tutte le informazioni e funzioni necessarie per la creazione e invio di bersagli al microcontrollore.

Con la funzione “Setup”, tutte le informazioni vengono calcolate e depositate in un altro grande oggetto chiamato “Phrase” all'interno di WS. Prima di tutto vengono copiate le coordinate cartesiane e create le coordinate in passi della frase nell'argomento della funzione. Poi, con le coordinate in passi, è possibile calcolare tutto ciò che riguarda la velocità, il tempo e l'intervallo tra ogni passo.

All'avvio del programma, questa funzione viene chiamata con la frase letta nel file “TransmissionPhrase.json”, così il software di scrittura è già dall'inizio pronto con l'ultima frase trasmessa.

WS.PushTarget(x, y, isup, point, filler, letter, label)

La funzione “PushTarget” è fondamentale perché permette di creare un nuovo bersaglio. Nonostante il numero di argomenti richiesti per questa funzione possa sembrare esagerato, sono necessari perché ognuno influenza in modo diverso come il microcontrollore debba comportarsi mentre cerca di raggiungere l'obiettivo. Tutte le informazioni calcolate in questa funzione sono depositate nella variabile “trg”, che viene aggiunta alla coda di bersagli.

Per calcolare un nuovo bersaglio, molti valori dipendono dal bersaglio precedente, per questo è necessaria una variabile “LastTarget” nel WS. Ad esempio, per calcolare un cambiamento di spazio dal punto iniziale a quello finale, bisogna conoscere le coordinate del bersaglio, ma anche le coordinate del punto di partenza, ovvero del bersaglio precedente.

Gli attributi “point” e “letter” non influiscono sul bersaglio in sé, ma forniscono informazioni utili ad esempio per seguire a che punto della frase corrisponde un bersaglio. L'attributo “label” serve invece a etichettare il bersaglio con una certa tipologia, ad esempio se equivale a 0 il bersaglio è considerato normale, ovvero che fa parte della frase, altri valori possono invece indicare ad esempio un'ascesa forzata della penna o un ritorno all'origine. Gli argomenti “x” e “y” sono le coordinate cartesiane del bersaglio, “isup” indica lo stato in cui deve trovarsi la penna prima di iniziare a spostarsi verso il bersaglio.

Questa funzione è molto simile a “Setup”: prima vengono copiate le coordinate cartesiane e calcolate quelle in passi, poi vengono calcolati i valori delle velocità con la funzione “CalcSpeeds”. Nell'oggetto WS sono presenti le costanti “MaxSpeed”; queste indicano la velocità massima in passi al secondo che i motori possono svolgere. Per raggiungere un bersaglio, un motore sicuramente si dovrà muovere più velocemente dell'altro, quindi si può assumere che un motore giri a velocità massima, mentre l'altro a una velocità compresa tra 0 e la velocità massima. Inoltre è da notare che i valori degli spostamenti e delle velocità sono algebrici, tengono quindi conto della direzione con il segno. Prima di calcolare le velocità il codice scopre quale motore deve andare a velocità massima e quale più lento, comparando i tempi in cui entrambi riuscirebbero a completare il cambiamento di posizione a velocità massima.

Conoscendo la velocità, si può ora anche ricavare il tempo ideale impiegato per raggiungere il bersaglio. Queste informazioni sono tutte utili e restano nella variabile, ma solo un attributo è in realtà fondamentale al microcontrollore per muovere i motori correttamente. Questo è l'intervallo di tempo tra i segnali digitali da mandare ai driver dei motori, calcolato tramite la velocità e il livello di microstepping configurato sui driver:

$$Stepdt = \frac{10^6}{|velocità| \cdot microstepping \cdot 2}$$

Il termine 10^6 rappresenta la quantità di microsecondi; il valore assoluto della velocità serve poiché l'intervallo di tempo deve avere un valore positivo, moltiplicato per un mezzo perché a ogni intervallo lo stato digitale viene invertito, e deve essere invertito due volte per risultare in un passo del motore.

Inoltre, durante questi calcoli bisogna fare attenzione ai casi limite dove la velocità, il tempo o il cambiamento di coordinate sono di valore zero, perché risulterebbero in operazioni impossibili, problemi facilmente risolvibili con un paio di dichiarazioni "if".

Quando tutte le informazioni sul bersaglio sono calcolate, esso viene aggiunto alla coda, e la variabile "LastTarget" viene aggiornata.

Sopra PushTarget sono costruite molte altre funzioni per aggiungere velocemente un intero insieme di punti, che si occupano di inserire i valori giusti per ogni argomento. Ad esempio la funzione PushAscend aggiunge un bersaglio che rappresenta solo il sollevamento della penna, e non richiede alcun argomento.

Oltre la coda di bersagli, WS contiene altre variabili booleane modificate in tempo reale, che rappresentano diversi stati del processo di stampa, come "IsPrinting" e "CurrTargCIntSent". Queste due in particolare servono alla comunicazione con la pagina internet per monitorare il processo di stampa in tempo reale. IsPrinting è controllata dall'utente con il bottone apposito, e diventa "falsa" automaticamente quando la coda di bersagli diventa vuota. Mentre quando la pagina richiede il bersaglio corrente la variabile "CurrTargSent" diventa "vera", e torna "falsa" solo quando il modulo Wi-Fi richiede un nuovo bersaglio. Questo è necessario per rispondere alla richiesta della pagina solo se il bersaglio corrente ancora non è stato mandato, per non inviare doppi. Con "bersaglio corrente" mi riferisco all'oggetto che il microcontrollore sta usando come obiettivo al momento, e corrisponde al primo elemento dell'oggetto "TargetQueue", ovvero il primo bersaglio in coda.

L'ultima funzione importante che richiede una spiegazione è "TargetQueueSetSize". Semplicemente, quando chiamata, analizza la coda di bersagli, e ritorna il numero di bersagli consecutivi che possono essere considerati dello stesso gruppo. Ad esempio, se la coda è lunga 50, ma partendo dal primo solo 20 bersagli sono della stessa lettera, questa funzione ritorna 20. Ci sono diverse condizioni analizzate per determinare se due bersagli fanno parte dello stesso gruppo, come l'appartenenza a una stessa lettera. Inoltre il gruppo ha una grandezza massima definita da "MaxSetSize". Questa funzione è usata per decidere quanti bersagli alla volta mandare al modulo Wi-Fi, un aspetto spiegato più in dettaglio nel capitolo 2.10.

2.10 Algoritmo del microcontrollore

Il microcontrollore, per comunicare con il server e controllare i motori, segue un algoritmo piuttosto complicato. Inizio spiegando lo sketch di Arduino caricato sul modulo ESP8266. La funzione `setup` configura le connessioni elettriche e connette il modulo Wi-Fi a una rete. Dopodiché il resto è tutto svolto nella funzione `loop`.

Ci sono due variabili booleane principali: “`IsPrinting`” e “`TargetReached`”. Nel contesto globale ci sono tutti gli attributi che definiscono il bersaglio corrente, come le coordinate da raggiungere e le velocità. `TargetReached` semplicemente indica se il bersaglio corrente è stato raggiunto o, contrariamente, se bisogna muovere la penna per raggiungerlo. In caso `TargetReached` sia “falso”, l'unica funzione che svolge il microcontrollore è “`GoTarget`”. Questa funzione utilizza le informazioni del bersaglio corrente per muovere i motori correttamente ogni volta che è chiamata, ovvero nel loop ad ogni intervallo di tempo. Su come il microcontrollore mandi i segnali giusti ai driver dei motori passo-passo andrà più in dettaglio nel capitolo 2.11. `GoTarget` viene chiamata fino a quando le coordinate della penna coincidono con il bersaglio, quindi quando la variabile `TargetReached` diventa “vera”. Una volta raggiunto il bersaglio, la funzione `GoTarget` non verrà chiamata finché un nuovo bersaglio non è ricevuto dal server.

La funzione “`ArdQueueUpdate`” richiede al server un insieme di bersagli, cosicché il microcontrollore possa raggiungerli uno ad uno. In questa funzione entra in gioco la libreria per Arduino “`ArduinoJson.h`”, utile perché permette di creare oggetti json da delle stringhe di testo, visto che tramite le richieste http non si possono mandare variabili, ma solo caratteri. La risposta del server consiste in un array di bersagli. La variabile “`totsize`” indica la grandezza di questo array, mentre “`remaining`” quanti bersagli dell'array sono ancora da raggiungere. Quando un nuovo insieme di bersagli è ricevuto, alla variabile “`remaining`” viene assegnato il valore `totsize`, mentre a `current` il valore -1, perché a questo punto ancora nessun bersaglio del nuovo array dovrebbe esser stato raggiunto. Ogni volta che la variabile `TargetReached` diventa “vera”, il bersaglio corrente viene riassegnato al prossimo oggetto di questo array, diminuendo di uno il valore di `remaining` e aumentando di uno il valore di `current`.

La funzione “`ArdFastUpdate`” può essere considerata un aggiornamento veloce dello stato del processo di stampa. Quando chiamata, il modulo Wi-Fi fa una richiesta al server, dal quale riceve un numero da zero a due come risposta. Se la risposta è zero vuol dire che la stampa è inattiva, quindi la variabile `IsPrinting` diventa “falsa”, mentre se equivale a uno la stampa è attiva, e `IsPrinting` diventa di conseguenza “vera”. se `IsPrinting` è “falsa”, una volta raggiunto il bersaglio corrente, quest'ultimo non viene aggiornato e la variabile `TargetReached` di conseguenza rimane accesa. Questo permette all'utente di mettere in pausa la stampa, avendo la possibilità di controllare questa variabile che disattiva l'aggiornamento del bersaglio corrente.

Se però il microcontrollore ha in memoria una coda di bersagli, e l'utente non volesse aspettare che vengano tutti raggiunti prima di modificarla, è necessario un modo per chiamare la funzione “`ArdQueueUpdate`” prima del previsto, ovvero prima dell'esaurimento dei bersagli.

Qui entra in gioco la terza possibile risposta alla richiesta “`ArdFastUpdate`”. Se la risposta alla richiesta ha valore 2, il microcontrollore richiede nuovamente un nuovo array di bersagli e aggiorna il valore di `IsPrinting`.

Una domanda legittima che può sorgere da questo modo di inviare bersagli è perché non inviare semplicemente un bersaglio alla volta. Teoricamente ciò risolverebbe molti problemi e semplificherebbe il codice, ma dal lato pratico nascono dei problemi. Le informazioni che definiscono un bersaglio sono molte, e le richieste http e la deserializzazione json richiedono tempo. Se a ogni punto della frase la penna dovesse fermarsi anche solo quasi un terzo di secondo perché il microcontrollore sta processando le informazioni ricevute, il movimento del robot diventerebbe visibilmente discontinuo e creerebbe imprecisioni nel tratto della penna. Per questo ArdFastUpdate, che viene chiamata tra un punto e l'altro, richiede solamente un valore numerico, rendendo l'attesa praticamente invisibile. In questo modo il grosso delle informazioni viene inviato nei momenti in cui il robot può permettersi di ritardare un attimo, permettendo comunque di comunicare con il robot mentre disegna, ad esempio mettendolo in pausa e osservando la posizione in tempo reale.

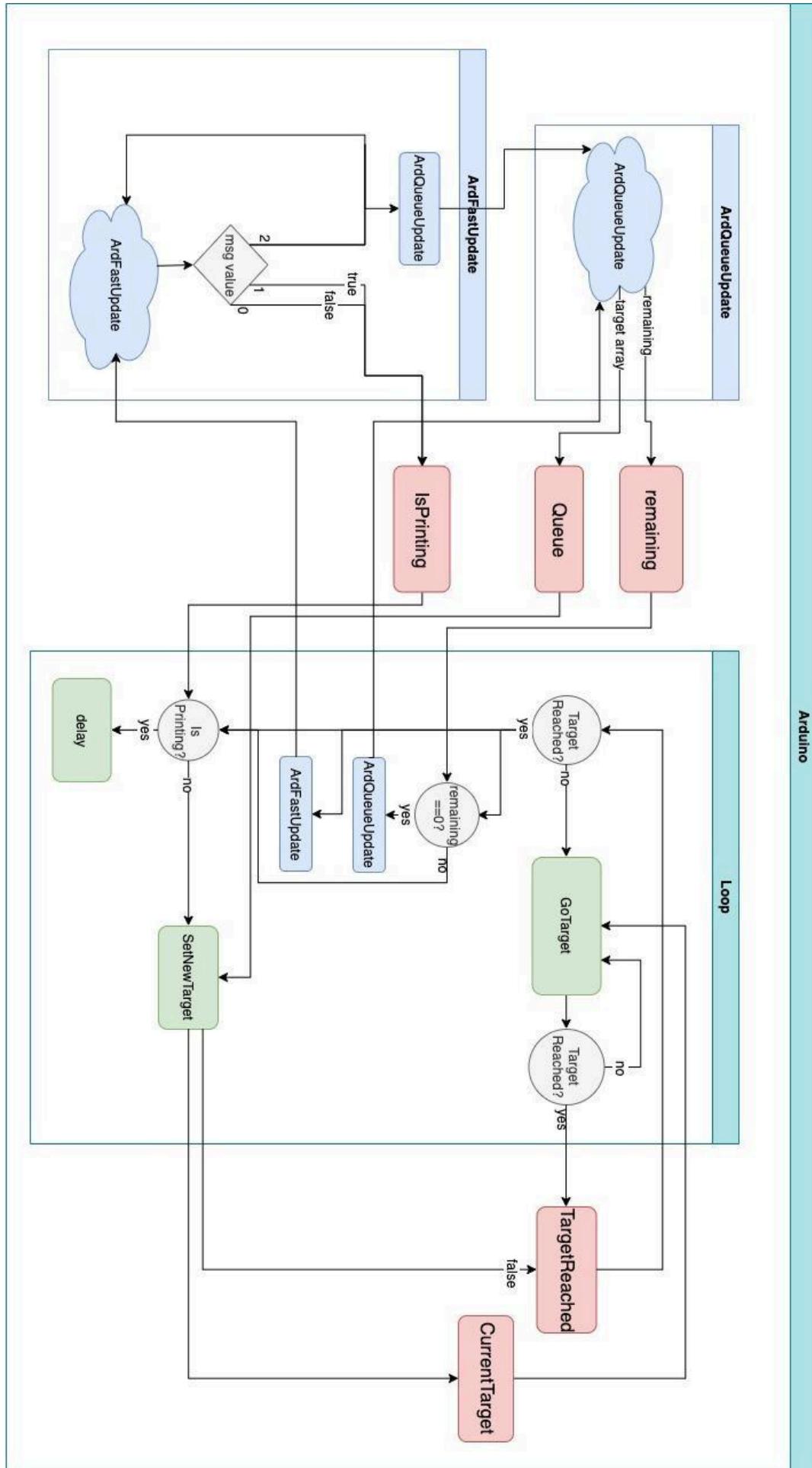
Per velocizzare, e migliorare, ulteriormente il processo di stampa è possibile disattivare questo aggiornamento che avviene tra ogni bersaglio, e usare il robot in modalità veloce, o “FastMode”, configurabile sempre dalla pagina di stampa. Con questa modalità attivata il modulo Wi-Fi comunica con il server solo quando ha terminato il gruppo di bersagli, perdendo quindi la possibilità di mettere in pausa a qualsiasi punto e seguire la posizione in tempo reale sulla pagina.

Gli insiemi di punti mandati non sono grandezze prestabilite, ma selezionati cercando questi punti di discontinuità dove il robot può permettersi di ritardare e perdere tempo a processare interi array di bersagli, come ad esempio al termine di una lettera. Questa scelta degli insiemi di punti da mandare è operata dalla funzione del software di scrittura “TargetQueueSetSize”, discussa nel capitolo 2.9.

Per quanto riguarda il codice server-side, l'unica cosa importante da capire è in che modo il server risponde alle 2 richieste del modulo Wi-Fi.

Quando riceve la richiesta per un nuovo insieme di bersagli, il server crea array per tutti gli attributi rilevanti, e per ogni bersaglio aggiunge un nuovo valore a ogni array. Con un ciclo “for” aggiunge i primi bersagli della coda del WS, quelli che si trovano nell'insieme definito da TargetQueueSetSize. Se però IsPrinting è “falsa”, solo un oggetto con attributo “totsize” con valore zero viene mandato in risposta.

Per la richiesta ArdFastUpdate il valore da mandare in risposta è determinato dalle variabili booleane. Leggendo il valore di IsPrinting si può determinare se rispondere zero o uno, in più se la variabile “ArdReupdate” è accesa la risposta consiste nel valore due. Inoltre, quando il server riceve questa richiesta, capisce che bisogna aggiornare le informazioni sul bersaglio corrente, e modificare in modo opportuno diverse variabili. Ad esempio controlla se il bersaglio corrente sia stato precedentemente mandato al modulo Wi-Fi, nel qual caso lo elimina dalla coda, mentre se la coda di bersagli si svuota, assegna “falso” alla variabile IsPrinting.



2.11 Driver dei motori passo-passo

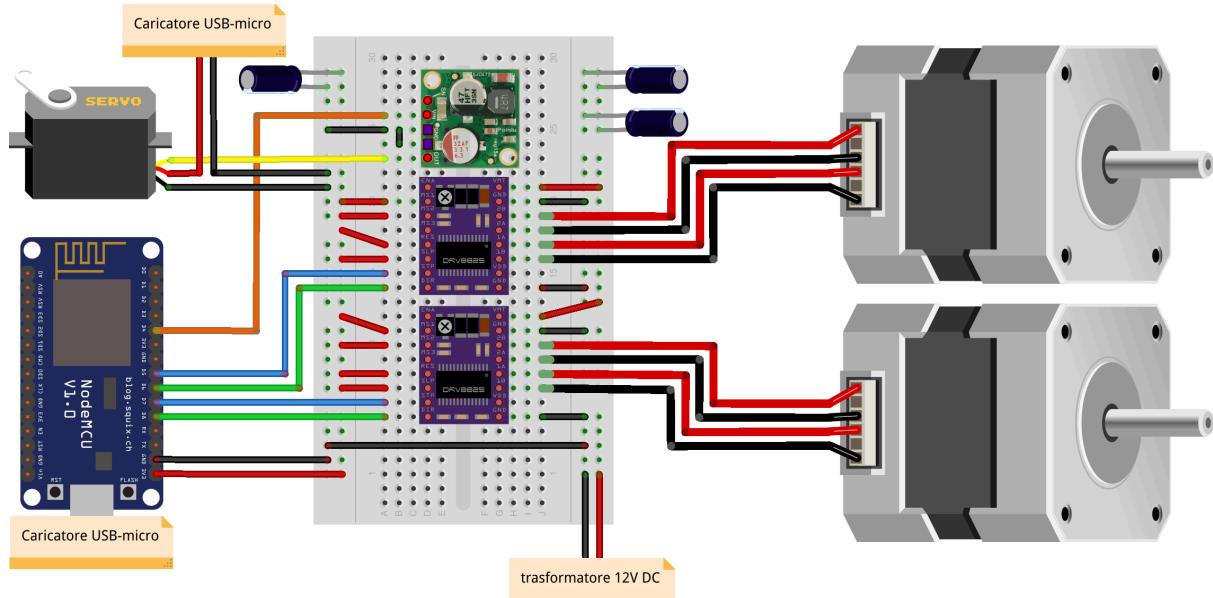
Il microcontrollore, quando deve muovere i motori passo-passo, manda impulsi elettrici ai driver DRV8825. Sapendo che quando il segnale digitale passa da spento ad acceso il driver muove di un passo il motore, è possibile far girare i motori alla velocità desiderata e a un particolare livello di microstepping, calcolando l'intervallo di tempo corretto tra ogni impulso, operazione già svolta nel capitolo 2.9.

Questo compito è affidato alla funzione GoTarget, che deve ruotare entrambi i motori alla velocità giusta. Innanzitutto una condizione fondamentale è che questo procedimento debba essere asincrono rispetto al resto del codice. Vuol dire che mentre altre funzioni vengono chiamate, questa funzione viene eseguita in sottofondo dalla funzione loop. Quindi ogni tipo di funzione “delay” non funzionerebbe, in quanto il resto del codice dovrebbe aspettare la completa esecuzione di questa funzione prima di essere raggiunto, rendendo la funzione sincrona.

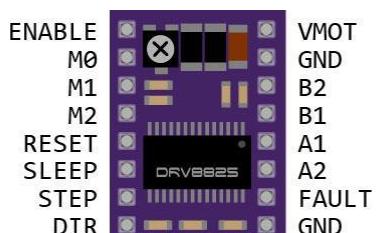
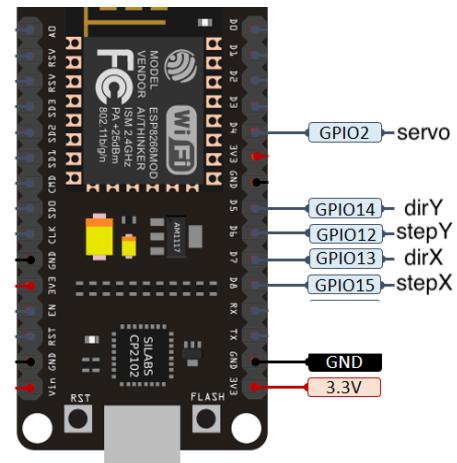
Il modo in cui l'intervallo di tempo è rispettato è con l'uso di un timer. GoTarget conta i microsecondi e tiene in memoria il tempo di partenza, e non appena è passato un tempo sufficiente, una porzione di codice viene eseguita e il tempo di partenza viene resettato al tempo corrente. In questa porzione di codice semplicemente viene invertito lo stato digitale della connessione elettronica al driver. Oltre all'impulso elettrico, si tiene conto delle coordinate attuali della penna, aumentando o diminuendo le coordinate a ogni passo, tenendo anche in considerazione il livello di microstepping.

La direzione del motore invece è configurata solo quando il bersaglio corrente viene aggiornato, guardando il segno del cambiamento delle coordinate dal punto iniziale a quello finale.

2.12 Elettronica



Il robot ha dei circuiti elettrici piuttosto semplici. Il modulo ESP8266 controlla i due driver dei motori passo-passo attraverso 4 connessioni digitali, e il servomotore con una connessione analogica. I cavi verdi veicolano gli impulsi dei passi che i motori R e A dovrebbero fare, mentre i cavi blu controllano i ponti H nei driver, cambiando direzione dei motori. Il microcontrollore fornisce anche 3.3 Volt al circuito logico dei driver, attivando i driver dai collegamenti "SLEEP" e "RESET" e controllando il livello di microstepping, in questo caso configurato su 8 attivando i collegamenti "M0" e "M1". I motori passo-passo sono collegati ai driver tramite i 4 collegamenti dedicati, e usano la corrente fornita dal trasformatore a 12 Volt attaccato a una qualunque presa. Il servomotore ha un voltaggio operativo di 4.8 Volt, quindi non gli fornisco corrente collegandolo al microcontrollore direttamente perché quest'ultimo potrebbe danneggiarsi. Riceve corrente da un comune caricabatterie USB-micro a 5 Volt, mentre il segnale PWM proveniente dal microcontrollore viene trasformato da 3.3 a 5 Volt tramite un convertitore Buck Boost prima di arrivare al collegamento giallo del servomotore. Anche il modulo ESP8266 è alimentato via cavo USB-micro da un caricabatterie per telefoni.



2.13 Riempire una figura

Fino ad ora mi sono sempre riferito all'insieme di punti da stampare come un semplice percorso, concettualmente di una solo dimensione, in quanto idealmente la matita non traccia un'area ma solo una linea. Con la funzionalità descritta in questo sottocapitolo è possibile trattare questo percorso non come l'oggetto finale da stampare, ma come bordo esterno di una figura da riempire interamente di un certo colore. Il software distingue tra queste due modalità di interpretare l'insieme di punti grazie all'attributo "shape" che ogni lettera possiede. Questa variabile booleana, se vera, fa eseguire al software di scrittura un algoritmo molto complesso per trovare ogni singolo punto all'interno del percorso che definisce la lettera.

Questo avviene durante la funzione setup del software di scrittura, quando una nuova frase viene trasmessa e deve essere preparata per la stampa. Se infatti la lettera è una figura da riempire, la funzione "PushLetter" aggiunge alla coda di bersagli non il solito percorso che definisce la lettera, ma il percorso che ritorna la funzione "Fill".

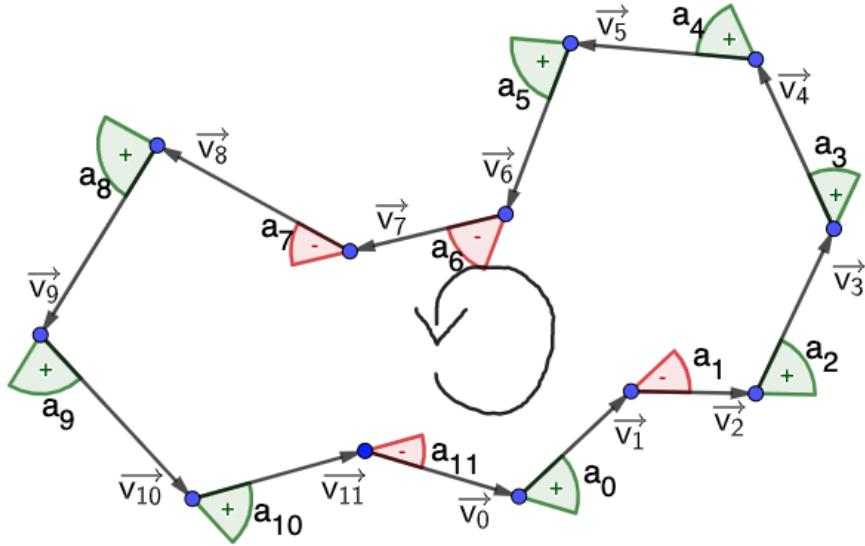
La funzione "Fill" si trova nell'oggetto "SP", importato nel file del software di scrittura da un altro modulo locale che ho scritto chiamato "ShapeFill". In questo modulo si trovano tutte le funzioni necessarie per determinare il percorso che riempie un'intera figura, dato solamente come argomento il bordo di quest'ultima.

Inoltre all'interno dell'oggetto lettera è possibile definire la variabile "holes", ovvero buchi, che ha influenza solo se la variabile "shape" è vera, e quindi se la lettera è una figura da riempire. Questa variabile è un array di percorsi, ognuno con il proprio array X e Y che definiscono i punti del percorso specifico. Questi percorsi sono chiamati "buchi" perché definiscono le regioni di spazio all'interno della figura che non devono essere riempite. L'oggetto "SP" infatti, nel calcolare il percorso che riempie la figura, prende in considerazione questi buchi, evitando i punti al loro interno mentre esegue la funzione "Fill". Ricapitolando, se la lettera ha l'attributo "shape" vero, alla coda di bersagli viene aggiunto un percorso che riempie la figura, evitando i buchi definiti nell'attributo "holes" della lettera.

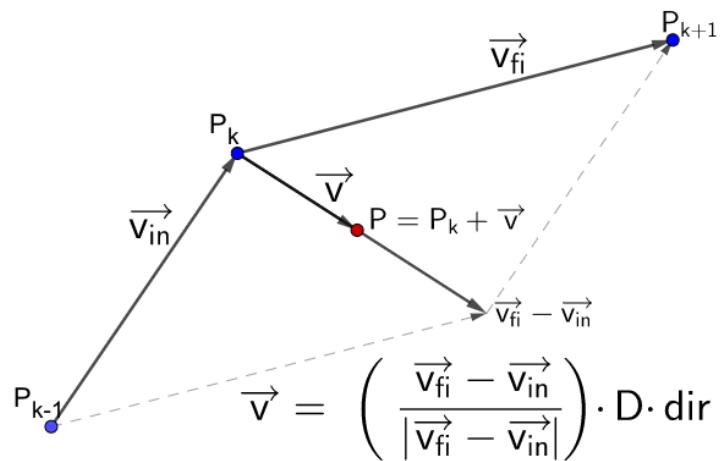
Prima di tutto l'oggetto SP richiede un setup iniziale, che inizializza diverse variabili nell'oggetto stesso per poter utilizzare la funzione "Fill". Ad esempio vengono copiati nell'oggetto il bordo e i buchi della figura, e l'array "pxls" viene ridefinito. Questo array contiene inizialmente il valore 0 per ogni singolo pixel della tela, così mentre la figura viene riempita i pixel corrispondenti ai punti aggiunti al percorso vengono marcati come letti, ridefinendo il valore nell'array a 1. Il software sarà in questo modo in grado di evitare di aggiungere gli stessi punti due volte, evitando i pixel con valore 1.

Un'altra importante variabile definita nel setup è “lefty”, un valore booleano che indica la direzione che segue il bordo mentre forma la figura, se esegue un giro completo virando verso destra o sinistra, rispettivamente in senso orario o antiorario. Viene calcolata guardando il segno della somma algebrica degli angoli formati dai vettori con la retta tangente per ogni punto.

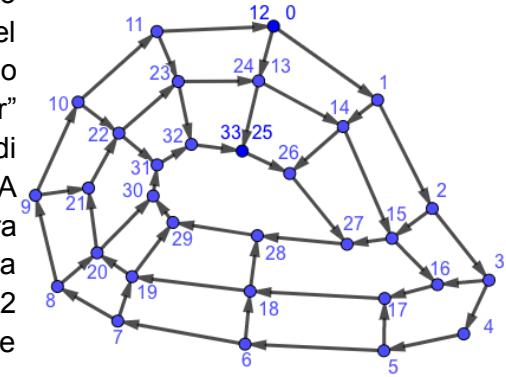
$$a_i = \arccos\left(\frac{\vec{v}_i \cdot \vec{v}_{i+1}}{|\vec{v}_i| \cdot |\vec{v}_{i+1}|}\right) \cdot \text{sgn}(\det(\vec{v}_i, \vec{v}_{i+1})) \quad \text{lefty} = \left(\sum_i a_i > 0\right)$$



Le coordinate dei punti all'interno della figura sono calcolate prendendo in considerazione i vettori che arrivano e partono dal “parente” dello specifico punto. Sottraendo il vettore iniziale a quello finale, si ottiene un vettore in direzione centripeta rispetto alla figura. Quest'ultimo viene poi normalizzato e scalato di un fattore D , che idealmente rappresenta la larghezza del tratto creato sul foglio di carta, ovvero il diametro della matita o penna. Inoltre, per determinare da che parte posizionare il punto, è necessario sapere la direzione del percorso che si sta usando. L'esempio riportato nella figura mostra che il punto P si trova dalla parte giusta se il percorso è in senso orario, ovvero se l'attributo “lefty” è falso, e quindi la costante “dir” dovrebbe avere valore “1”. Mentre se il percorso andasse in senso antiorario, il software inverte il vettore di spostamento interno calcolato, assegnando “-1” alla costante.



La funzione “Fill” cerca i punti da aggiungere all’oggetto “fillers”, ovvero l’insieme di punti che riempiono la figura. Tutto funziona grazie a un ciclo “while” che, partendo dal bordo, a ogni iterazione trova tramite la funzione “inner” il percorso di punti concentrico rispetto a quello trovato nell’iterazione precedente. Questo percorso concentrico è creato aggiungendo per ogni punto del percorso parente il punto determinato con il metodo sopracitato. Questo processo continua finché “inner” non ha più punti nuovi da aggiungere, quindi ritornando come immagine un percorso vuoto. A questo punto termina il ciclo “while” e tutta la figura dovrebbe essere stata riempita. Ad esempio in questa figura si vede come, partendo dal bordo lungo 12 punti, “Fill” ritorna un percorso di 33 punti che riempie la figura concentricamente.



Questa però è una figura semplice e “rotonda”, che può essere riempita con un solo percorso concentrico senza passare dagli stessi punti più volte, possibile perché il percorso P di n punti soddisfa la seguente identità:

$$\text{con } P = \{0, 1, 2, \dots, n - 1\} \text{ e } f(x) = \text{dist}(P_k, P_x)$$

$\forall k \in P \ \exists M \in \mathbb{N}$ tale che:

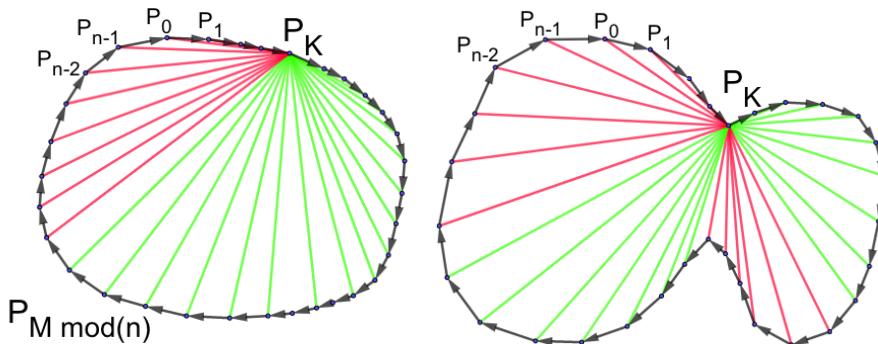
$f(x)$ è monotona crescente per $x \in \{j \bmod(n) \mid k \leq j \leq M\}$

$f(x)$ è monotona decrescente per $x \in \{j \bmod(n) \mid M \leq j \leq k + n\}$

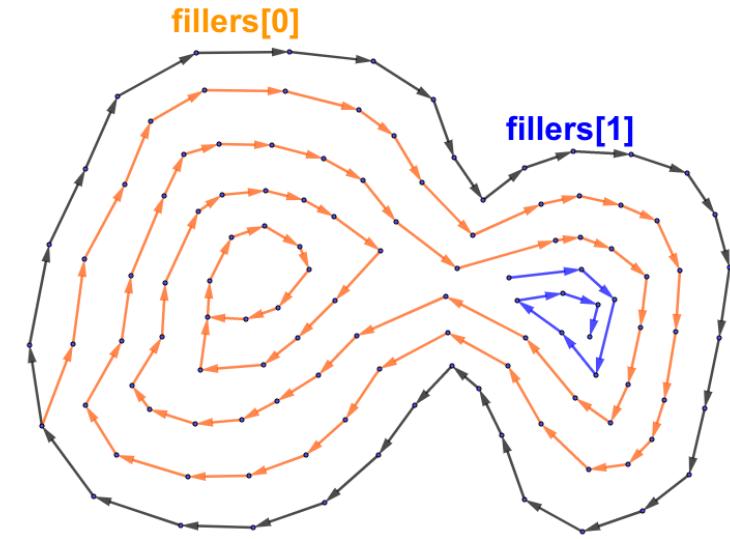
Ovvero: per ogni punto K , attraversando il percorso la distanza tra il punto stesso e gli altri punti cresce costantemente fino a un certo punto con indice $M \bmod(n)$, dopodichè decresce fino a tornare al punto di partenza, e quindi possiede un solo punto “estremo”. Se questo vale, la figura è “convessa”, come nell’esempio che mostro nella prima immagine.

Il valore M è sempre maggiore di K , ma $M \bmod(n)$ indica in ogni caso un punto sul percorso che può anche venire prima di K . In questo modo posso usare l’identità sopradefinita anche per i casi dove, nell’ordine del percorso, il punto estremo viene prima del punto K .

Ma è possibile che una figura non soddisfi queste condizioni, e il bordo quindi non possieda un valore M accettabile per un qualsiasi punto K . Nella seconda immagine infatti mostro un tale esempio, con il colore dei segmenti che rappresenta il crescere o decrescere di $f(x)$. Nel secondo caso non è possibile trovare un unico percorso concentrico, ma bisogna dividere i punti opportunamente per creare percorsi che di nuovo soddisfino l’identità.



In questi casi vengono trovati normalmente i primi strati interni finché una parte della figura viene chiusa fuori, in quanto non è possibile arrivarci senza attraversare punti già aggiunti in precedenza nel percorso. Dopo questa divisione, il percorso continua l'esplorazione delle regioni accessibili, mentre in un secondo momento, ricordando il punto in cui il primo percorso è stato tagliato, viene creato un secondo percorso che riempie le regioni inaccessibili. Per questo in realtà l'oggetto che ritorna la funzione "Fill" è un array di percorsi, ovvero di "fillers". In questo esempio si vede come il primo "filler", colorato in arancione, viene tagliato nel punto dove la figura si restringe, e continua riempiendo solo la regione a sinistra. Quando quest'ultimo ha terminato, viene aggiunto un altro percorso all'array "fillers" che riprende nel punto in cui il percorso precedente è stato tagliato.



Per evitare i buchi, e allo stesso tempo mantenere un certo livello di ordine nel percorso, sfrutto invece un meccanismo relativamente semplice. Mentre la funzione "inner" crea uno strato interno allo strato precedente, se a un certo punto si trova troppo vicino a un buco aggiunge direttamente tutti i punti che compongono il bordo di quest'ultimo, per poi continuare normalmente dal punto interrotto.

Perché questo funzioni è fondamentale che tutti i buchi abbiano il bordo che va in direzione inversa al bordo esterno della figura.

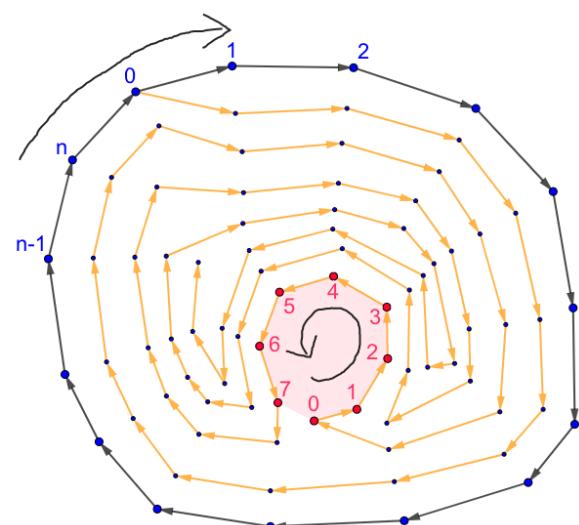
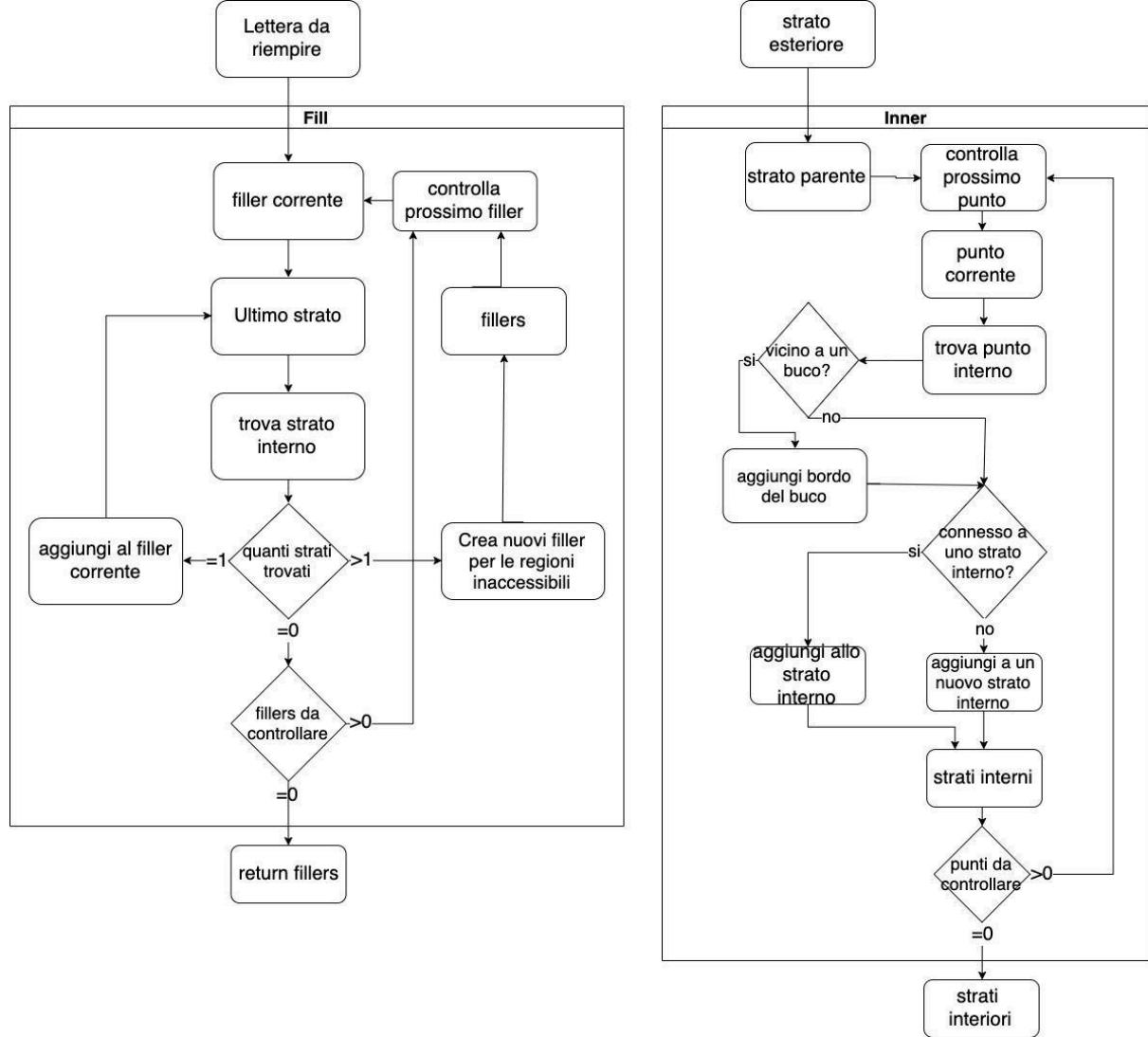
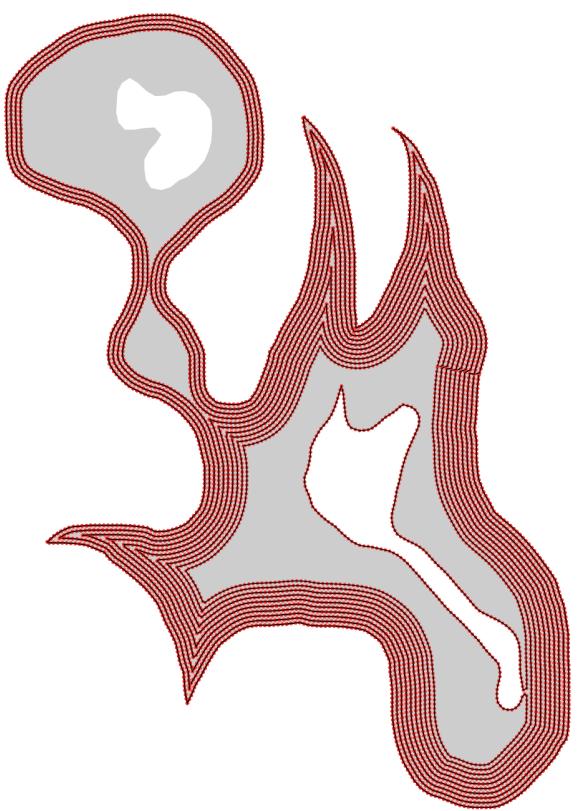
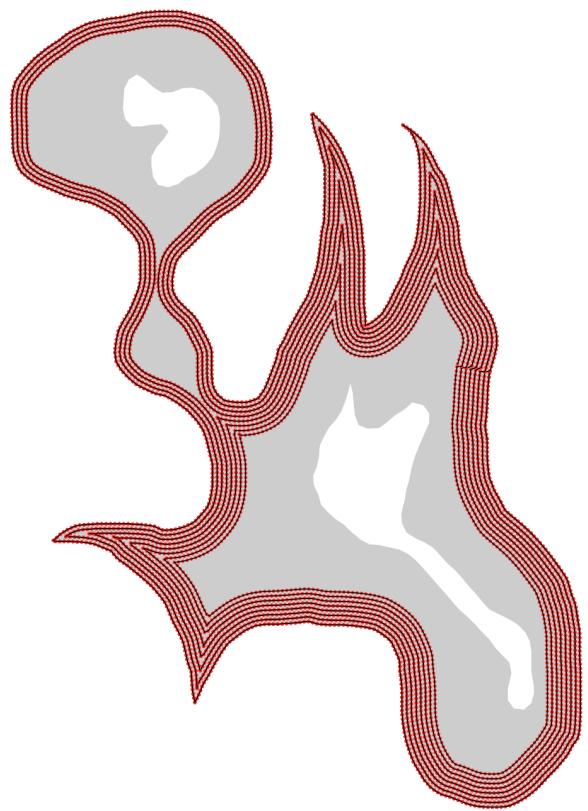
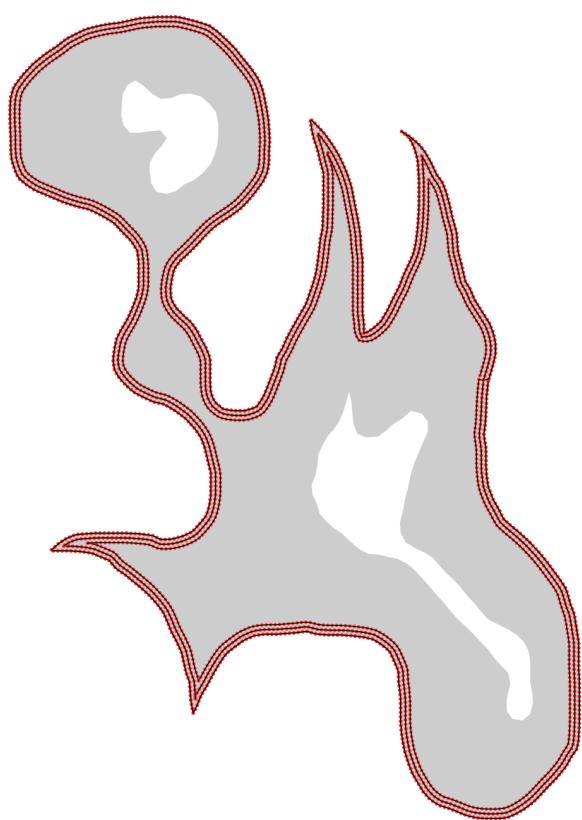
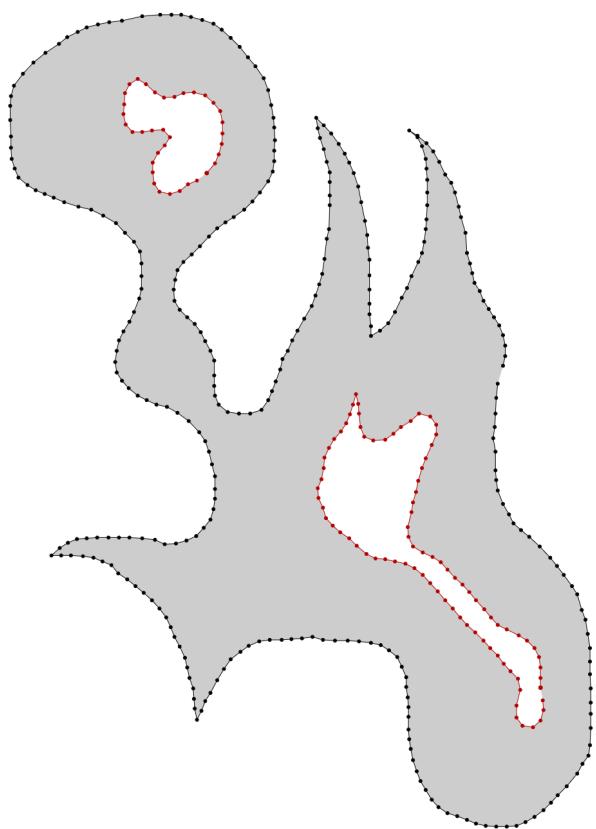
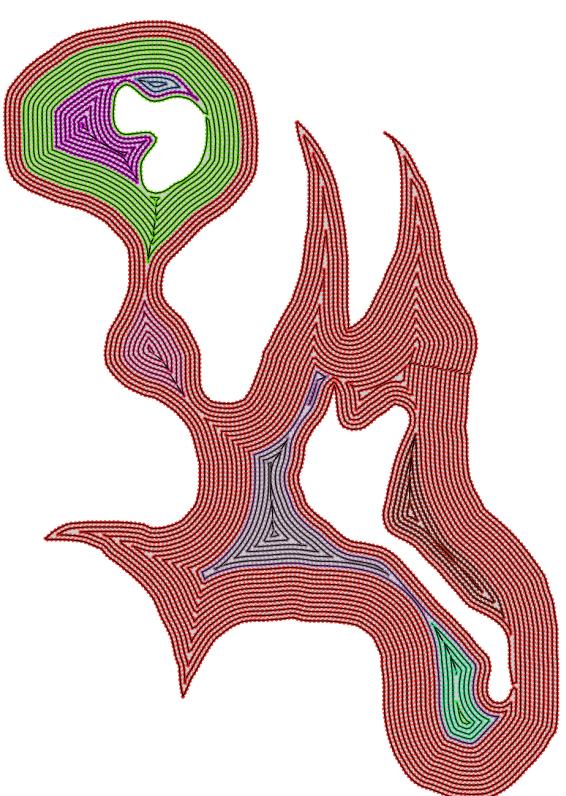
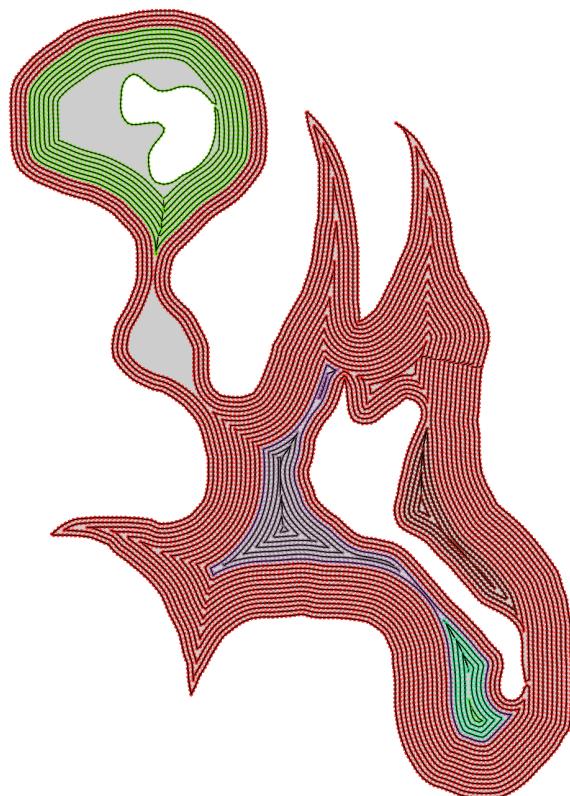
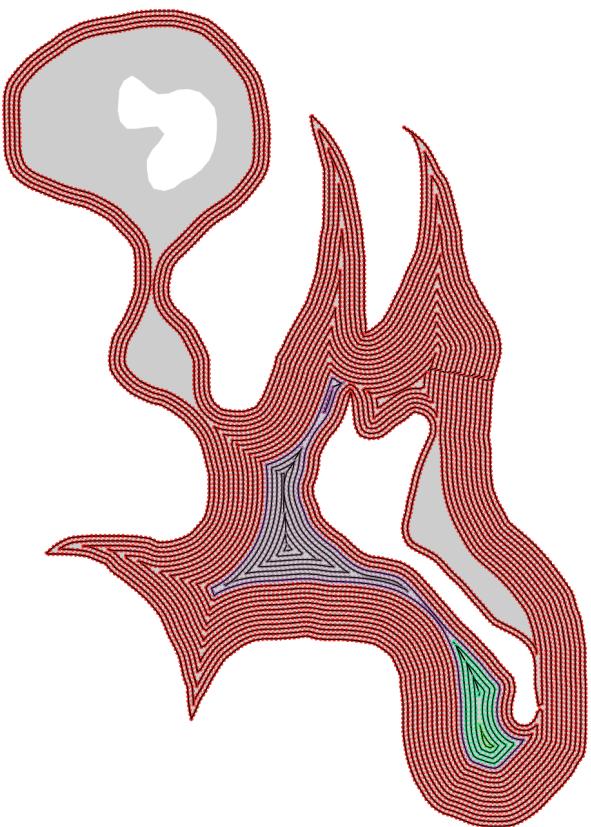
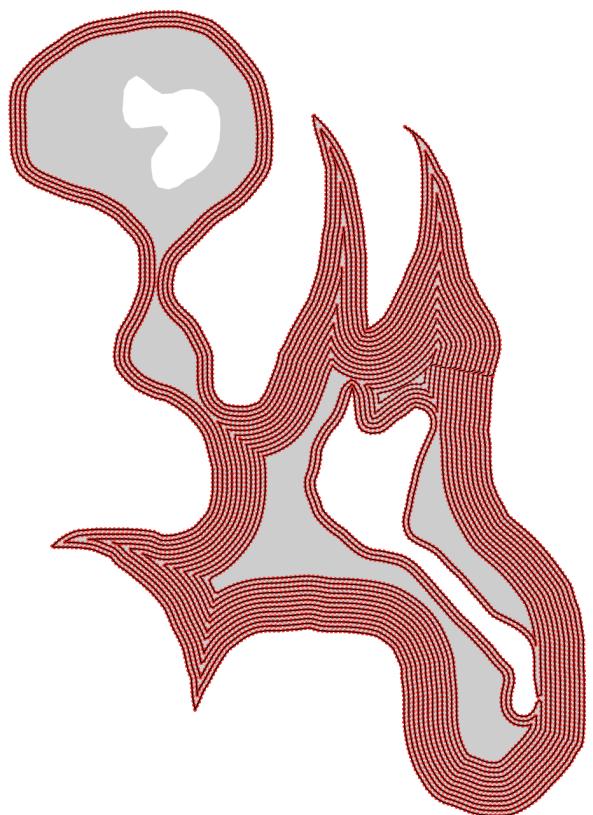


Diagramma di flusso delle funzioni “Fill” e “Inner”:



Nelle prossime pagine è presentata un'applicazione pratica di questa funzionalità: la figura definita soltanto dai percorsi del bordo esterno e dei buchi viene riempita. Ogni colore corrisponde a un oggetto dell'array “fillers” creato dalla funzione “Fill”, ovvero un percorso che riempie una porzione della figura.





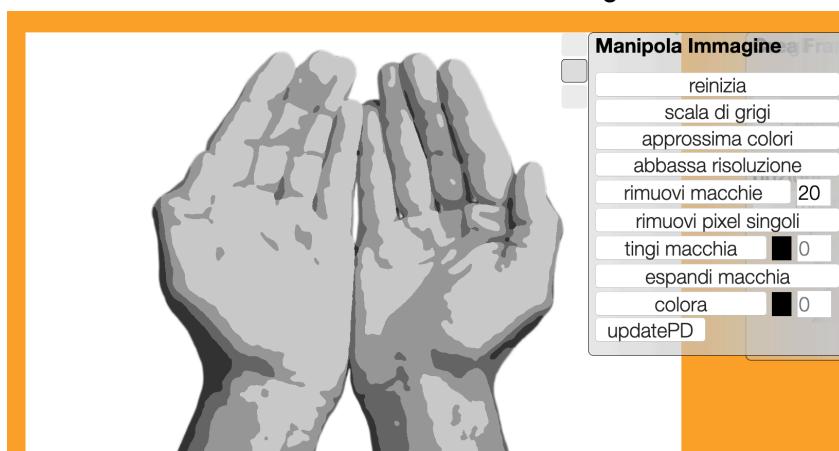
2.14 Scansionare un'immagine

Il disegno libero sulla tela bianca non è l'unico modo per creare nuovi file da stampare. La pagina "ImageScan" mette a disposizione tutti i controlli necessari per trasformare le informazioni contenute nell'array di pixel in percorsi di punti ordinati che descrivono i bordi delle diverse figure che compongono l'immagine scelta. Dalla pagina internet è possibile caricare un'immagine nel browser, o immettere l'url di un'immagine su internet nell'apposita casella di testo per iniziare a lavorare. Tutti questo processo è svolto nel browser client-side, quindi non è necessaria alcuna comunicazione con il server, al quale viene solo inviato il prodotto finito della scansione.



Prima di parlare della scansione in sé, apro una parentesi sulla manipolazione dei pixel dell'immagine. Un'immagine qualsiasi è troppo imprevedibile e piena di discontinuità per riconoscerci all'interno dei confini precisi per le figure. Sarebbe estremamente difficile sviluppare un algoritmo che tiene automaticamente conto di tutte le variabili, quindi ho ritenuto necessario facilitare il processo con dei controlli manuali. Per questo se la scansione di una parte dell'immagine non va come previsto, è sempre possibile correggere qualche pixel e riprovare.

L'utente può manipolare i pixel dell'immagine tramite i bottoni nel secondo navigatore estendibile destro. Questo serve a preparare l'immagine a essere davvero analizzata in cerca di figure da riempire. Tutto quello che compone un'immagine infatti è soltanto un insieme di figure da riempire di diversi colori, per questo è solo possibile distinguere al massimo 6 gradazioni di grigio e non si può pretendere dal software di lavorare con una scala rgb classica. Approssimando ogni pixel a una delle 6 gradazioni, l'intera immagine perde certamente la maggior parte dei dettagli, ma in compenso diventa composta da semplici figure di colori che, in certa misura, rendono l'immagine ancora riconoscibile.



Il bottone “scala di grigi” trasforma ogni pixel singolarmente da rgb a grigio basandosi sulla formula per calcolare la luminosità, ovvero:

$$lum = 0.3 \cdot \text{rosso} + 0.59 \cdot \text{verde} + 0.11 \cdot \text{blu}.$$

Il valore trovato, che va da 0 a 255, viene approssimato a solo 6 gradazioni attraverso la funzione “approssima colori”. A questo punto ogni pixel possiede solo uno dei 6 valori: 0, 51, 102, 153, 204, 255.

Successivamente la funzione “abbassa risoluzione” rimuove ulteriori dettagli, prendendo ogni singolo pixel in relazione ai suoi vicini entro un determinato raggio. Se un pixel di un certo colore non ha abbastanza vicini dello stesso colore, questo verrà inizializzato al colore più comune nelle vicinanze. Questo passo è necessario per “arrotondare” i bordi delle figure, cosicché questi siano molto più evidenti e distinguibili dal software che successivamente termina la scansione.

Il software lavora molto con il concetto di “macchia”, oggetti nel codice riferiti come “blob”. Con la funzione “UpdateBlobPxls”, nell’array “blobpxls” all’indice per ogni punto viene salvata una copia delle coordinate di tutti i punti dello stesso colore e collegati al punto stesso. In questo modo per sapere tutti i punti facenti parte di una macchia, sapendo solo un punto nella macchia, basta accedere con l’indice corrispondente all’array. L’array viene creato tramite un semplice algoritmo ricorsivo: partendo da un punto, aggiunge i punti vicini dello stesso colore che non sono già stati aggiunti, e ripete per questi nuovi punti. In questo modo l’insieme che definisce la macchia cresce fino a coprire un’intera figura nell’immagine. Il bottone “rimuovi macchie” rimuove gli insiemi di pixel meno grandi del valore impostato sulla destra del bottone.

Se necessario, è anche possibile selezionare una macchia ed espanderla o colorarla di un colore specifico, oppure direttamente modificare i singoli pixel disegnando con il cursore.

Con queste funzioni a disposizione si possono correggere i punti dell’immagine che causerebbero problemi durante la scansione e stampa di quest’ultima. Ad esempio, se il robot dovesse abbassare e alzare la penna per coprire solo qualche pixel, il processo sarebbe interminabile e il risultato caotico. Per questo molti dettagli vengono eliminati, semplicemente il prodotto finito è più riconoscibile e veloce da stampare. Inoltre con ogni manipolazione dell’immagine è probabile che molti errori e pixel “d’interferenza” vengano creati nel processo, e bisogna correggerli al costo della risoluzione.



Una volta che l'immagine è pronta, il software è in grado di riconoscere i bordi delle diverse macchie e creare dei percorsi che definiscono le figure che compongono l'immagine intera. Una volta trovati i bordi e i buchi, ovvero ciò che definisce una figura come discusso nel capitolo precedente, si possono aggiungere a una nuova lettera della frase corrente, cosicché quest'ultima possa in seguito essere salvata, trasmessa e stampata.

Essendo l'algoritmo per analizzare l'immagine e trovare le figure che la compongono molto complicato, l'ho scritto su un file a parte chiamato "PathDetector.js". La pagina internet include automaticamente questo file dalla cartella "JSscripts", mettendo a disposizione l'oggetto "PD", che contiene tutte le funzioni necessarie per trovare i bordi e i buchi delle figure nell'immagine.

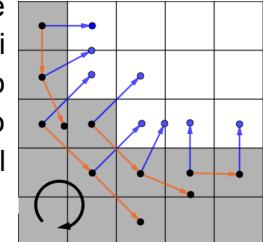
Quando l'immagine è pronta per essere scansionata, l'array di pixel viene passato all'interno dell'oggetto "PD", che ne fa una copia al suo interno e inizializza tutte le variabili per prepararsi a una nuova scansione. L'array di pixel non è bidimensionale, per accedere a uno specifico punto dello schermo bisogna quindi calcolare il suo indice all'interno della lunga sequenza.

L'array originale dell'immagine è in formato rgba, quindi ogni pixel ha bisogno di quattro variabili. Ad esempio, per trovare il valore del colore verde del pixel con coordinate $x = 10$, $y = 20$, bisogna cercare nell'array all'indice $[(10 + 20 \cdot h) \cdot 4 + 2]$, dove h è l'altezza in pixel dell'immagine. Durante il setup iniziale all'interno di PD l'array è semplificato, prima di tutto dando un solo valore a ogni pixel, e questo valore va solo da 0 a 5, indicando a quale delle 6 gradazioni di grigio appartiene il pixel. A questo punto è possibile chiamare la funzione "DiscoverShape" all'interno di PD, che ritorna un oggetto con i percorsi del bordo esterno e di tutti i buchi della figura che è stata selezionata sullo schermo. Per mostrare come questo sia possibile devo prima spiegare un paio di concetti.

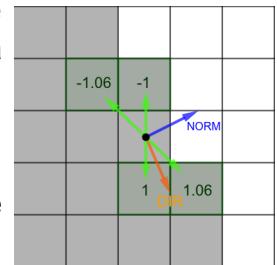
L'obiettivo è riuscire a trovare in maniera ordinata i punti che stanno sul confine tra 2 figure con colori diversi. La funzione "IsBorder" ritorna un valore booleano che verifica se le coordinate immesse come argomento appartengono a uno di questi punti di confine. Per essere tale, un pixel deve essere a contatto con almeno un pixel di colore diverso. Nell'immagine si vede il risultato di IsBorder per i diversi punti della figura nera.

0	0	1		
0	1	1		
0	1			
0	1	1	1	
0	0	0	1	

La funzione "BorderVect" ritorna un vettore che indica la direzione in senso orario del confine, dal punto di vista di un particolare punto. Facendo la media aritmetica delle coordinate dei punti esterni, ovvero di colore diverso, e trovando il vettore che va dal punto di partenza a questo punto esterno medio, si trova il vettore normale rispetto al confine. Ruotando quest'ultimo in senso orario di 90 gradi invece si trova il vettore idealmente tangente al confine, che punta verso la direzione, in senso orario, del bordo. Nell'esempio i vettori blu indicano verso il punto esterno medio, mentre ruotati di 90 gradi sono tangenti al confine, e fanno percorrere il bordo della figura in senso orario.



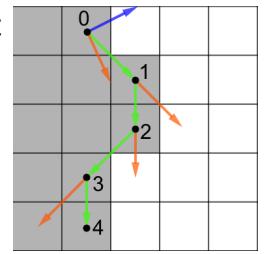
Partendo da un punto di confine casuale, posso quindi trovare la direzione da prendere attraverso "BorderVect". Con delle coordinate di partenza e una direzione, il software cerca il prossimo punto da aggiungere al percorso con la funzione "NextBorderPoint". Tutti i punti di confine vicini sono candidati a essere aggiunti come prossimi, ma solo quello che meglio approssima la direzione da prendere verrà selezionato. Per eseguire questa selezione, viene massimizzato il prodotto scalare tra il vettore direzione e il vettore normalizzato tra il punto d'origine e il punto candidato. L'esempio illustra il vettore normale al bordo in blu, il vettore direzionale creato da "BorderVect" in arancione e tutti i potenziali prossimi punti del bordo con il valore del prodotto scalare in verde. In questo caso il punto in basso a destra con vettore normalizzato V massimizza il prodotto scalare e quindi verrà selezionato.



$$NORM = \begin{bmatrix} 1 \\ -\frac{1}{2} \end{bmatrix} \quad DIR = \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix} \quad V = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \quad DIR \cdot V = \frac{3}{4}\sqrt{2}$$

Questo processo continua in loop, chiamando sempre NextBorderPoint con il vettore dal punto precedente al punto appena trovato come direzione.

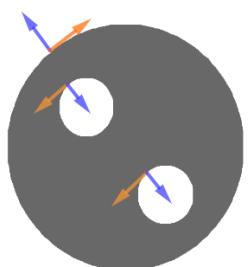
In questo esempio si parte dal punto di confine casuale in alto con $NORM = (1, -0.5)$ $DIR = (0.5, 1)$ selezionando di conseguenza il punto in basso a destra con $V = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ da essere aggiunto al percorso.



La seconda volta che NextBorderPoint è chiamata verrà usato $DIR = (1, 1)$ perché è il vettore di spostamento dal punto precedente, ovvero dal punto 0 al punto 1, e come risultato verrà selezionato $V = (0, 1)$, quindi il punto in basso. Analogamente, nel punto 2 vale $DIR = (0, 1)$ $V = (-1, 1)$.

Una precisazione importante è che mentre questo processo va avanti, i punti trovati e aggiunti al percorso vengono segnalati come letti in modo da non passare dagli stessi punti più volte. Infatti il loop finisce quando non ci sono più punti di confine disponibili per andare avanti, quindi nel momento in cui si torna al punto di partenza, dove il bordo è stato marcato. I punti sono segnati modificando il valore del loro colore all'interno dell'array di pixel. In questo modo non saranno più classificati come punti di confine della figura, in quanto saranno di colore diverso.

Come già spiegato nel capitolo precedente, è fondamentale che i bordi dei buchi di una figura vadano in verso opposto rispetto al bordo esterno di quest'ultima, altrimenti il riempimento non funzionerebbe correttamente. Questa condizione rende in realtà le cose molto più facili quando le figure vengono create scansionando un'immagine. Sapendo che i percorsi creati lungo i bordi vanno nella direzione del vettore normale al bordo ruotato di 90 gradi in senso orario, possiamo assumere che qualsiasi bordo esterno trovato con questo processo vada anch'esso in senso orario, mentre il bordo di qualsiasi buco sempre al contrario. Il riempimento delle figure si basa su questa funzionalità appunto perché in questo modo non bisogna mai neanche verificare in che direzione vanno i percorsi trovati dalla scansione.



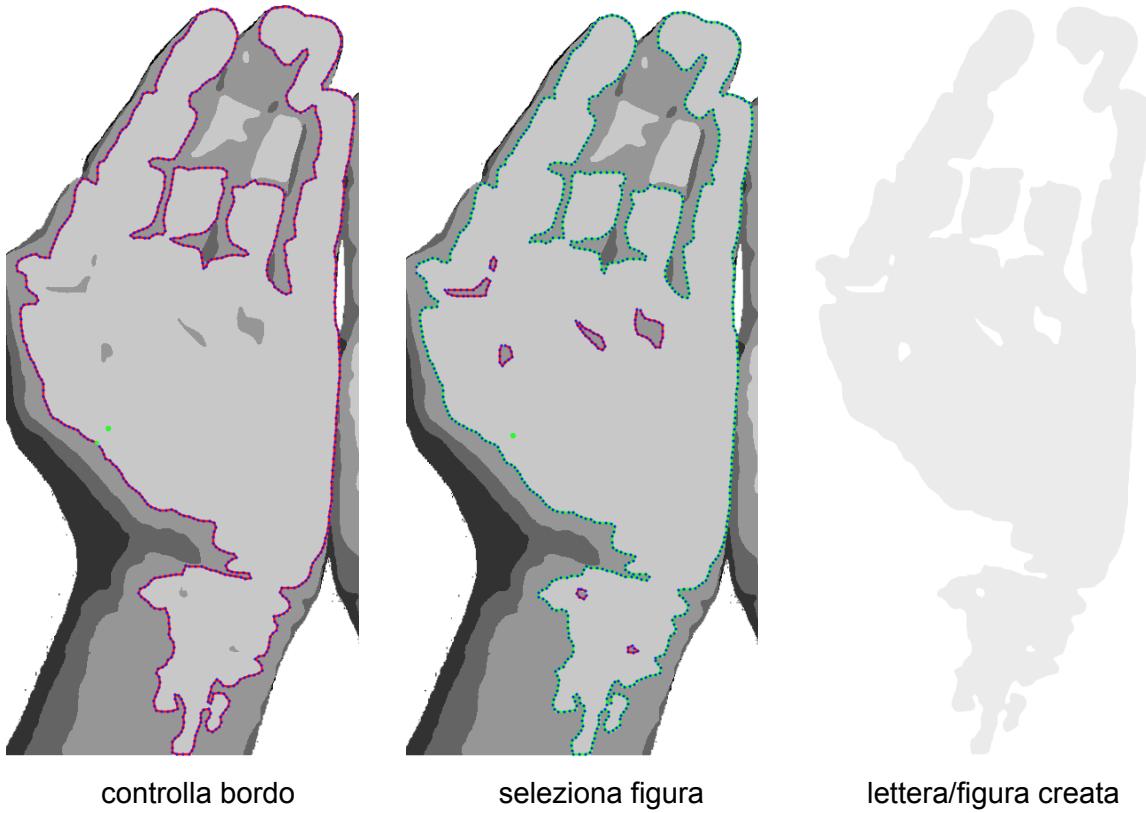
La funzione che si occupa di questo loop è “BorderPixelPath”, che crea il percorso di tutti i pixel che definiscono un bordo della figura. Una volta, finito i pixel marcati come letti vengono inizializzati al loro valore originale.

Tutte queste componenti dell'algoritmo sono messe assieme nella funzione “DiscoverPath”, che semplicemente, prendendo come argomento le coordinate di un punto, trova il punto di confine più vicino e, usando le funzioni sopracitate, ritorna i punti del bordo, occupandosi anche di altri dettagli che non ho menzionato.

Dalla pagina internet è possibile visualizzare il percorso trovato dal software selezionando un punto sull'immagine e premendo “controlla bordo” nel terzo navigatore estendibile destro. Il punto verde sull'immagine indica il punto selezionato con il mouse, mentre il bordo trovato è colorato di rosso, e va in senso orario partendo dal punto verde più piccolo.

Questa però è solo una funzionalità per controllare rapidamente che il riconoscimento funzioni correttamente per quel bordo. La funzione più completa per creare una vera e propria lettera da aggiungere alla frase corrente che si trova nel server è “DiscoverShape”. Premendo “seleziona figura” quest'ultima controlla tutti i punti della figura selezionata, e per ogni nuovo punto di confine chiama DiscoverPath, trovando ogni bordo nella figura e distinguendo il bordo esterno dai bordi dei buchi all'interno della figura. Inoltre rende visibile il bordo esterno in verde e i buchi in rosso. Premendo invece “aggiungi figura” viene creata una nuova lettera e, successivamente, aggiunta alla frase come figura da riempire con i suoi possibili buchi e il suo bordo esterno, trovati dai percorsi calcolati precedentemente con DiscoverShape.





2.15 Calcolo della varianza

Per qualsiasi forma una lettera possa assumere, le velocità con cui si raggiunge un bersaglio dipendono esclusivamente dal tragitto tra due punti. Con questa opzione invece le cose cambiano. Includendo la varianza nel calcolo delle velocità, più un bersaglio si trova in un punto "spigoloso", o molto "ripido", più la velocità verrà diminuita.

Per calcolare questa "ripidità" ho usato le formule nel campo della matematica della variabili aleatorie e delle loro distribuzioni.

Usando questa analogia, dato un percorso P con n punti, definisco la variabile aleatoria X :

$$X \in \{v_j \mid P_j \in P\}$$

Ovvero: ogni punto P_j del percorso P possiede un proprio attributo di velocità v_j , e la variabile aleatoria X può assumere tutti i valori di queste velocità.

L'esponenziale dell'opposto della distanza tra il punto k e j invece rappresenta il valore della distribuzione di probabilità $f(X)$ relativa allo specifico punto k :

$$p_k(X = v_j) = f_k(v_j) =: \exp(-\text{dist}(P_k, P_j)) \cdot D_k, \quad D_k = \left(\sum_{j \in P} \exp(-\text{dist}(P_k, P_j)) \right)^{-1}$$

La distribuzione di probabilità ha un pedice perché è diversa per ogni punto k del percorso. La funzione è scelta arbitrariamente, è semplicemente un termine che decresce con la distanza, cosicché nella sommatoria la velocità di punti più distanti abbia peso minore rispetto ai punti più vicini. Il fattore D scala la funzione in modo che diventi una distribuzione. Adesso possiamo calcolare il valore atteso di X nel punto k :

$$E_k(X) = \sum_{j \in P} v_j \cdot f_k(v_j) = \sum_{j \in P} v_j \cdot \exp(-\text{dist}(P_k, P_j)) \cdot D_k =: v_{Kmedia}$$

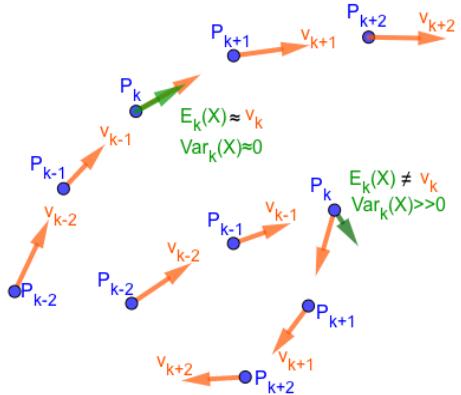
Il valore atteso nel punto k rappresenta la media delle velocità dei punti circostanti, considerando però il fatto che più un punto è vicino al punto k , più la sua velocità avrà

influenza sulla media, infatti la media è diversa per ogni punto e il valore atteso ha un pedice che lo rende relativo a ogni punto k .

Ciò che serve a questa funzionalità però è il discostamento del valore della velocità dalla velocità media, ovvero la varianza, calcolata come segue:

$$Var_k(X) = E_k((X - E_k(X))^2) = \sum_{j \in P} (v_j - v_{Kmedia})^2 \cdot exp(-dist(P_k, P_j)) \cdot D_k$$

Ad esempio nell'immagine il vettore verde rappresenta la velocità media nel punto k , quindi $E_k(X)$. Avendo la lettera in basso a destra un punto "spigoloso", e di conseguenza un'alta varianza, la velocità sarà diminuita molto. Mentre la lettera in alto è molto "liscia", con una varianza praticamente nulla perché la velocità di ogni punto è molto simile alla velocità media.

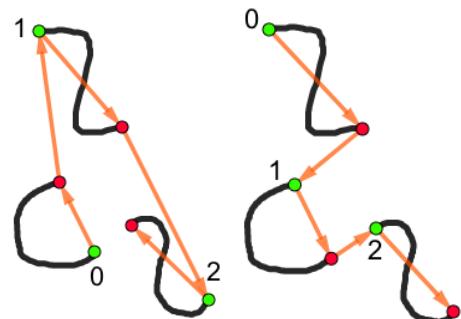


2.16 Ordine ottimale delle lettere

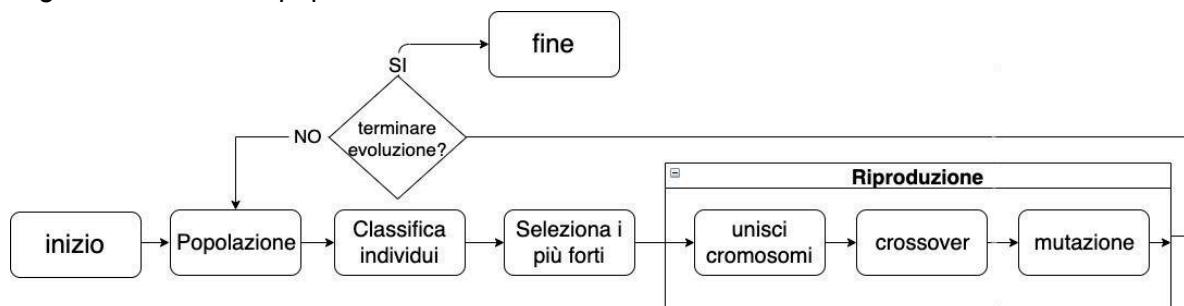
Una volta creata una frase, un fattore che potrebbe ridurre di molto il tempo totale necessario per stamparla è l'ordine in cui vengono eseguite le lettere. Questo accade nella funzione setup del software di scrittura, ovvero quando una nuova frase è preparata per essere stampata. Non si tratta solo di cambiare gli indici in cui appaiono le lettere nell'array che definisce l'intera frase, ma anche di potenzialmente invertire la direzione con cui una lettera è scritta, quindi partendo dall'ultimo punto e finendo nel primo. Nelle immagini si vede un esempio di questo tipo di riordinamento: partendo da una situazione caotica e inefficiente il software è in grado di trovare la strada più veloce conoscendo i primi e gli ultimi punti delle lettere, rispettivamente in Verde e in rosso.

Mentre cercavo di creare un algoritmo efficiente per trovare ogni singola possibile combinazione e selezionare la migliore, ho scoperto il popolare problema informatico denominato "traveling salesman", ovvero il calcolo della strada più corta che collega con un preciso ordine un insieme di città, cosicché un uomo d'affari le possa visitare tutte esattamente una volta.

Il mio caso è un'applicazione particolare di questo problema, un po più complicata perché ogni città ha un punto di partenza e di arrivo, che possono anche essere scambiati. Il problema nasce quando le città da visitare non sono soltanto una decina ma molte di più, in quanto il numero di combinazioni cresce con il fattoriale del numero delle città. Per di più nel mio caso il numero di combinazioni è moltiplicato per un fattore esponenziale a causa dei 2 possibili versi in cui percorrere ogni città, risultando in un totale di $2^n \cdot n!$ dove n è il numero di città. Questa è senz'altro una funzione che cresce molto velocemente, non è quindi realistico controllare ogni singola combinazione, per questo ho optato per un approccio differente.



L'ordine ottimale viene trovato attraverso un algoritmo genetico, ovvero un metodo di risolvere problemi di ottimizzazione basandosi sul modo in cui l'evoluzione migliora i geni degli individui di una popolazione.



Nel caso del problema “traveling salesman”, e per estensione anche nella mia situazione, un individuo è definito dall’ordine in cui percorrere le città, quindi da un insieme di geni che indicano ognuno a che indice dovrebbe appartenere una lettera. La forza di un individuo è classificata in base all’ inverso del tempo necessario a completare il percorso nel particolare ordine. Quindi le combinazioni più efficienti sono rappresentate dagli individui più forti che la popolazione ha da offrire. Questi individui migliori vengono selezionati per riprodursi, passando i loro geni ai nuovi individui che costituiranno la generazione successiva. Andando avanti in questo modo il sistema trova nel tempo i geni migliori, non solo selezionandoli dal patrimonio genetico iniziale, ma anche creandone nuovi attraverso le mutazioni. Le mutazioni sono infatti fondamentali, perché permettono a lungo termine di sperimentare con nuovi geni, che in caso si manifestino vantaggiosi vengono mantenuti all’interno del patrimonio genetico.

L’unico aspetto negativo è ovviamente il fatto che la soluzione trovata con questo metodo non è per forza la migliore, ma solo molto buona. Riducendo la potenza di calcolo necessaria si perde la certezza matematica che garantiva l’approccio con l’applicazione della forza bruta. Ma in pratica l’algoritmo funziona molto bene e fornisce sempre l’ordine ottimale in cui attraversare le città, o nel mio caso le lettere.

L’intero algoritmo si trova su un modulo NodeJS locale chiamato “OptimalPath.js”. In esso si trovano tutte le funzioni per classificare gli individui, selezionare un insieme di parenti, mescolare il genoma di due individui per crearne uno nuovo, mutare e altro. Una volta importato il modulo nel file principale “server.js” nella variabile “OP”, basta immettere le posizioni d’inizio e fine di tutte le lettere in una funzione per ottenere la combinazione più efficiente. Il software di scrittura da questo punto in poi userà questa nuova frase riordinata al posto di quella trasmessa. Le informazioni sulla nuova combinazione vengono mantenute in una variabile, in una sorta di dizionario che fa corrispondere l’indice di una lettera nella vecchia combinazione al suo indice nella combinazione ottimale.

Diario

3.1 Prime idee

Dall'inizio sono restato con l'intenzione di costruire un robot che disegnasse utilizzando le coordinate polari invece che cartesiane, infatti questa è stata la caratteristica che più definiva il mio progetto.

Pensavo di costruire un robot di dimensioni molto più piccole, con la penna supportata da un'asta centrale, ruotata direttamente da un motore. Un altro motore invece avrebbe dovuto controllare l'inclinazione della penna, modificando il raggio. Ma questo sarebbe stato un design semplicemente troppo problematico.

Non ho impiegato molto tempo per concettualizzare in modo generale la versione finale del robot. Qualche settimana prima della fine dell'anno scolastico mi sono quindi messo all'opera per creare un modello 3D. Con l'applicazione CAD Autodesk Fusion 360 ho creato in due settimane quello che avrebbe costituito il prototipo costruito durante l'estate.

Fusion 360 ha avuto un ruolo fondamentale nel mio progetto: avere un modello esatto da seguire durante la costruzione è un grande aiuto, per una chiara mappa della posizione di ogni componente, ma soprattutto per un modo intuitivo di visualizzare il funzionamento del prototipo.

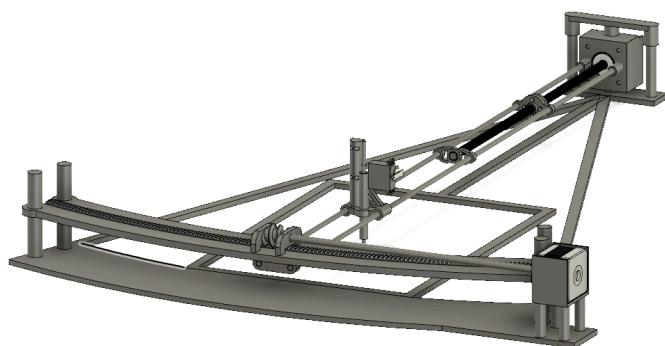
All'inizio la curva di apprendimento è piuttosto ripida, essendo comunque un'applicazione dedicata a un uso professionale e industriale, ma grazie a un'ampia disponibilità di risorse online sono riuscito a creare un intero modello computerizzato del prototipo.

Prima del termine della terza liceo e durante l'estate il prof. Bergomi mi ha stampato tutte le parti in plastica di cui avevo bisogno, e entro agosto sono riuscito a costruire un prototipo funzionante.

Oltre alle componenti stampate in plastica, avevo anche progettato delle parti in formato svg, per ritagliare delle figure in plexiglas con una tagliatrice laser.

Per quanto riguarda le componenti, i motori passo-passo li ho presi in prestito dalla riserva della scuola, assieme a un paio di fili.

Prima di partire in vacanza, quindi a fine giugno, ho ordinato dalla Cina i driver dei motori passo-passo e una vite di controllo, mentre le aste di metallo che sostengono la penna e le parti laterali del primo prototipo le ho comprate al Jumbo di Grancia.



3.2 Prototipo estivo



Il prototipo estivo era composto da uno scheletro stampato in plastica, ma concettualmente è molto simile alla versione finale. Un motore controlla la lunghezza del raggio e l'altro muove le aste di acciaio, cambiando l'angolo.

Questo prototipo però presenta molti problemi:

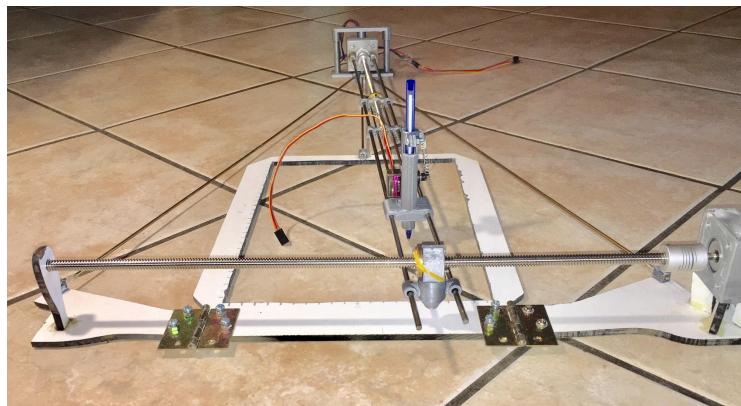
1. Il meccanismo che permette alla penna di muoversi è instabile, e il servomotore resta fissato solo grazie ad un paio di graffette. Inoltre non permette la pressione configurabile, ma solo uno stato binario della penna.
2. Le parti in plastica sono grandi al massimo 210 mm, ciò rende necessario l'uso di molte componenti, che devono essere fissate assieme tramite un paio di viti, e complessivamente rende l'intera struttura meno stabile e difficile da smontare.
3. La rotaia che supporta le aste di acciaio e controlla l'angolo deve essere di un materiale molto duro. Per questo le parti in plexiglas che il prof. Bergomi mi ha fabbricato con la tagliatrice laser sono state utilizzate solo in parte, in quanto non riuscivano a sostenere il peso dell'impalcatura. Ho quindi dovuto ritagliare manualmente un asse di legno, che però ha reso l'intera struttura meno precisa e meno stabile, a causa di occasionali intoppi in sezioni dove il legno non era regolare.

Durante l'estate ho anche avuto un paio di problemi con l'elettronica che mi serviva per il progetto. I driver dei motori passo-passo continuavano a danneggiarsi, probabilmente per via di un'intensità di corrente troppo elevata, quindi continuavo a provarne di nuovi, finché ho trovato i drv8825, che per adesso sembrano reggere. Anche l'Arduino ha smesso di funzionare, ne ho quindi ordinato un clone, assieme a due moduli Wi-Fi, dei quali solo uno funzionava bene. A questo punto ho scartato un'idea interessante nata poco tempo prima di dedicare un modulo Wi-Fi a ogni motore, creando un algoritmo sul server che tiene i motori in sincronia, aggiungendo ulteriore complessità al progetto. Anche i servomotori ho scoperto non essere molto affidabili. Quello dello Starter Kit si è rotto d'estate, mentre due ordinati successivamente non mostravano neanche segni di vita. Inoltre ho scoperto che un regolare servomotore non ha abbastanza momento torcente per applicare una forza sufficiente sulla molla, sarebbe quindi impossibile calcare troppo con una regolare matita senza usare un servomotore più grande.

Per risolvere i problemi dell'impalcatura in legno, in agosto e settembre ho ricostruito tutto ciò che componeva la base del robot, scartato il meccanismo della rotaia, progettato un nuovo meccanismo per il movimento della penna e per il controllo dell'angolo con il secondo motore.

La versione finale ha una base costituita da un unico grande componente, che è anche molto più resistente della plastica. Le estremità del robot sono pieghevoli grazie a 2 cardini di una porta.

Essendo la vite di controllo precisa e facile da utilizzare, ho deciso di usarne una anche per il controllo dell'angolo, come visibile in figura. È sostenuta dal motore stesso e da un altro componente di legno, incastrato solidamente nella base.



3.3 Uso dell'Arduino

In principio non mi era venuto in mente di usare un modulo Wi-Fi, e credevo di essere limitato all'uso dell'Arduino. Infatti a giugno e luglio ho ideato un programma per il controllo dei motori tramite Arduino.

L'Arduino fungeva da microcontrollore, e tramite un collegamento a una porta seriale del computer richiedeva bersagli al programma che avevo scritto.

Ho usato la popolare applicazione Processing, in linguaggio Java, per creare un programma sul computer che permetesse all'utente di disegnare e mandare all'Arduino, tramite la comunicazione seriale, le informazioni sui punti da raggiungere. Questa modalità di controllo però, nonostante la sua semplicità e comodità, limita ampiamente le potenzialità di un tale robot. Prima di tutto l'interfaccia per l'utente è primitiva e difficile da rendere intuitiva, a differenza di una pagina html. Inoltre permette di mandare informazioni al microcontrollore solo tramite un cavo collegato a un computer.

Il software in sé non è cambiato molto, perché la libreria p5 che uso nel codice client-side è un'evoluzione di Processing, con le stesse funzioni "setup" e "draw". L'unico cambiamento sta nel modo di comunicare le informazioni al microcontrollore.

Per queste ragioni la comunicazione remota tramite una pagina internet mi ha affascinato subito come idea, e ho subito ordinato un paio di moduli Wi-Fi dalla Cina a inizio agosto.

All'inizio pensavo di richiedere le informazioni al server tramite il modulo Wi-Fi e trasmetterle, tramite comunicazione seriale o I2C, all'Arduino, che poi avrebbe trasmesso gli impulsi ai driver dei motori. Ma, scoprendo che il modulo esp8266 aveva abbastanza collegamenti disponibili per il mio progetto, ho deciso di scartare completamente l'Arduino.

Come già accennato precedentemente, il modo in cui il microcontrollore manda impulsi ai driver dei motori passo-passo è asincrono, ma non è sempre stato così. Il problema di un algoritmo sincrono si è presentato mentre programmavo la comunicazione seriale tra Arduino e Processing. Conoscendo l'intervallo di tempo tra un impulso e l'altro, ho iniziato semplicemente usando un ciclo for con la funzione dell'Arduino "delay". Ma, visto che il delay per un motore influenza tutto il codice, cambia anche la velocità finale del secondo motore. Inoltre, mentre questo ciclo di delay viene eseguito, non si possono svolgere funzioni finché il bersaglio è raggiunto. Quindi ho pensato a questo algoritmo più evoluto e complicato per controllare i due motori in modo asincrono tramite due cronometri programmati nel codice.

3.4 Server e sito internet

La vera sfida che ho incontrato durante l'estate, ad agosto, è stata imparare a gestire la comunicazione tra un server e un client, non conoscendo assolutamente niente sull'argomento. I linguaggi di programmazione html e javascript invece non credo siano particolarmente complicati per iniziare, e ho quindi deciso di basare il mio progetto su questi. Il server infatti l'ho creato usando NodeJS, che permette di eseguire codice JavaScript sul computer, evitando quindi di usare altri linguaggi più popolari per il codice server-side come Python.

Un'alternativa a NodeJS sarebbe trovare un host che mantiene il server online. A differenza del server creato localmente, però, creare, leggere e manipolare i file è molto più complicato perché richiede anche l'uso di un database online come SQL. Complessivamente quindi credo che l'approccio che ho usato io sia molto più semplice, nonché gratuito.

Quando ho cominciato a scrivere il codice, ho rapidamente realizzato che per un progetto di questa magnitudine bisogna essere molto più ordinati, e pensare in anticipo all'impostazione generale del programma.

Ad esempio, dopo aver scritto una pagina internet rudimentale per il disegno libero sulla tela bianca, ho usato la libreria p5 in modalità globale, non contando che forse in futuro avrei avuto bisogno di inserire due canvas nella stessa pagina. E quando questo bisogno si è manifestato, per aggiungere un joystick per muovere e ingrandire la tela, ho dovuto riscrivere tutto il codice p5 in "instance mode".

Inoltre, avendo moltissime idee su come espandere in futuro il sito, ho deciso di separare le porzioni di codice che possono restare utili da quelle che sono rilevanti solo a una pagina in particolare. Per questo moltissime funzioni si trovano nella cartella JScripts, e non nel file js di una pagina in particolare. Questa struttura modulare permette facilmente di creare nuove pagine con nuove funzionalità, usando sempre gli stessi blocchi di partenza. Un altro grande esempio di questo aspetto è il codice per creare il navigatore laterale estendibile. Questo elemento viene creato in ogni pagina con poche linee di codice, perché è già tutto presente sotto forma di libreria nelle cartelle CSS e JScripts.

Anche per il codice server-side ho svolto questo compito di organizzazione in diversi file delle diverse funzioni, creando dei moduli che vengono poi importati dal file principale.

3.5 Possibili Innovazioni

Il settore del progetto che credo abbia più bisogno di innovazioni è la disponibilità di strumenti per creare nuove frasi. Per adesso si può creare un nuovo percorso per la penna tramite il disegno libero sulla tela bianca e la scansione di un'immagine.

Un'idea un po' basica è un software per la scrittura di testo. Sarebbe comodo poter scrivere un testo sulla pagina internet, e il codice aggiunge automaticamente per ogni carattere un percorso pre-configurato nel software alla frase corrente.

Un'altra idea, forse più interessante, è quella dell'uso di un'intelligenza artificiale per analizzare le immagini, invece di creare artificialmente un algoritmo molto complicato. Ma il problema con questo approccio è che avrei bisogno di un vasto insieme di immagini già scansionate per l'allenamento di tale intelligenza artificiale.

3.6 Problemi riscontrati e persistenti

- Per controllare i motori con un'intensità di corrente adeguata, bisogna regolare i potenziometri sui moduli drv8825, che ho scoperto è sempre meglio controllare prima di usare il robot, perché molte volte devono essere ricalibrati. Ad esempio capita spesso che mentre i due motori si muovono, a un certo punto si bloccano a causa di questo problema, costringendomi a regolare un po' i potenziometri finché i motori ricominciano a muoversi normalmente.
- Provando a calcare troppo la matita o penna si crea una forza d'attrito troppo elevata tra il foglio e la punta. La matita, facendo fatica a muoversi sul foglio mentre i motori continuano a girare, al posto di restare verticale si storta verso la direzione dello spostamento, in quanto la aste di acciaio non sono perfettamente rigide, ma un po' flessibili.
- L'occasionale intoppo del meccanismo per il controllo dell'angolo tramite la vite di controllo. Il bullone sulla vite non è fissato verticalmente, e quindi è libero di dondolare. Mentre muove le aste di acciaio, se queste non scorrono nei due buchi liberamente causano lo spostamento in avanti o indietro del bullone, bloccando il sistema.

Conclusione

Prima di iniziare il lavoro di maturità non avrei mai creduto possibile creare un ambiente che permette a chiunque di semplicemente “stampare” un file digitale premendo un paio di bottoni su un sito internet. È facile pensare alla stampante di casa come una scatola chiusa che, magicamente, reagisce ai comandi del computer producendo dal nulla il foglio desiderato. Questi macchinari non cadono dal cielo; come sono costruiti in fabbrica, possono anche essere costruiti a casa. Aprendo la scatola, si impara che con le istruzioni e le componenti giuste, si può costruire in salotto. Ad esempio questo è il caso se si vuole comprare una stampante 3D. Non arriva come prodotto finito, ma con tutte le componenti e le istruzioni per costruzione e uso. È praticamente solo una costruzione Lego molto complicata. E questo è ciò che credo di aver dimostrato a me stesso, e a chi sta leggendo, con il robot che ho costruito.

Bibliografia

- Motori passo-passo nel dettaglio: <https://www.youtube.com/watch?v=0qwrnUeSpYQ>
 - Trasformatori: <https://www.youtube.com/watch?v=lT19dg73nKU>
 - Modulo ESP8266: <https://www.youtube.com/watch?v=NEo1WsT5T7s>
 - Driver motori passo-passo: <https://www.youtube.com/watch?v=5CmjB4WF5XA>
-
- Canale youtube di Daniel Shiffman:
<https://www.youtube.com/user/shiffman/videos?app=desktop>
 - Server con NodeJS, comunicazione server-client e progettazione di una API:
https://www.youtube.com/playlist?list=PLRqwX-V7Uu6YxDKpFzf_2D84p0cyk4T7X
 - Pagine internet e p5js: <https://www.youtube.com/watch?v=8j0UDiN7my4>
 - Ordine ottimale delle lettere attraverso un algoritmo genetico ispirato da:
https://www.youtube.com/watch?v=M3KTWnTrU_c

Allegati

- Tutti i file di cui parlo li lascio in allegato in una cartella Google Drive raggiungibile tramite questo link
<https://docs.google.com/document/d/1yufAzvm-YmBF5FJW-CDQHOr3H1Efvr5cywDJvZXItzl/edit?usp=sharing>
- Ho creato un breve video dimostrativo che mostra quasi tutte le funzionalità del robot
<https://youtu.be/YZUVihPjlJg>