UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

Corso di Laurea in Informatica

PolyRec.js: una libreria JavaScript per il riconoscimento di gesture

PolyRec.js: a JavaScript library for gesture recognition

Relatori:

Prof.
Gennaro COSTAGLIOLA

Dott.
Mattia DE ROSA

Candidato:

Francesco Pio Covino
Mat. 0512105735

ANNO ACCADEMICO 2020/2021

Dedicata a chi
ci ha sempre creduto.

Ringraziamenti

Prima di procedere con la trattazione, vorrei dedicare qualche riga a tutti coloro che mi sono stati vicini in questo percorso di crescita personale e professionale.

Un sentito grazie al mio relatore il prof. Costagliola per la sua disponibilità e tempestività ad ogni mia richiesta. Un enorme ringraziamento al dott. Mattia De Rosa per avermi fornito ogni materiale utile alla stesura dell'elaborato, consigliato ed aiutato nel progetto ogni qual volto ne avessi bisogno.

Fondamentale è stato il supporto morale dei miei genitori, della frizzante Gerardina, di Benedetto, di Alfredo e di Angelica, mia fan numero uno, senza i quali non sarei mai riuscito ad arrivare fin qui. Grazie, soprattutto nei momenti di sconforto.

Ringrazio i miei colleghi e i miei amici per essermi stati accanto in questo periodo così intenso e per gioire, insieme a me, dei traguardi raggiunti.

Abstract

La navigazione nelle pagine web, così come l'interazione con le stesse su dispositivi touchscreen, ha incluso nel tempo l'uso di gesture. La facilità d'utilizzo e le molteplici associazioni possibili possono rendere le gesture un mezzo efficace per navigare attraverso un sito web o per effettuare operazioni sugli elementi della pagina. Il lavoro di tesi si basa su PolyRec, un riconoscitore di gesture, precedentemente sviluppato in Java, adatto anche per la prototipazione rapida di applicazioni basate su gesture. Il riconoscimento di una gesture avviene in 3 fasi sequenziali: in primis ogni gesture viene approssimato ad una polilinea, le due polilinee in esame vengono allineate per trarne un uguale numero di segmenti ed infine avviene il calcolo della distanza sommando il contributo di ciascuna coppia di segmenti. PolyRec.js è un riconoscitore, che si basa sui medesimi concetti della versione Java, più compatto, adattato per l'utilizzo lato Client senza necessità di moduli aggiuntivi o di un esecuzione lato server. Il riconoscitore è compresso in un unico file .min.js per facilitarne la portabilità e renderlo fruibile in diversi contesti e per diversi scopi.

INDICE

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

CAPITOLO 1

INTRODUZIONE

L'interazione utente con i moderni calcolatori si è evoluta nel tempo, in diversi contesti, verso l'uso delle gesture. Con la diffusione del touch screen e la crescita esponenziale degli smartphone come prodotti di semplice fruizione, è nato il bisogno di dover interagire in modo veloce con i programmi applicativi. Il mercato tecnologico è stato scosso dalla quasi completa sostituzione delle tecniche di interazione standard con approcci di tipo *"touch"* e *"motion sensing"*. Recenti studi sul futuro dell'interazione degli esseri umani con la tecnologia sostengono, senza ombra di dubbio, che *"le gesture e il controllo del movimento diventeranno vitali nei prossimi anni per alcune forme di comunicazione tra le persone e le macchine, aprendo la strada ad approcci innovativi e sempre più rapidi"* [1].

La tesi presenta PolyRec.js, una libreria JavaScript per il riconoscimento di gesture a 2 dimensioni. PolyRec [2], riconoscitore di gesture, di cui è già presente una versione Java [3], è adatto per la prototipazione rapida di applicazioni basate su gesture. Il lavoro di tesi è il risultato di un processo di adattamento del riconoscitore ad un linguaggio lato Client quale JavaScript, per essere utilizzato rapidamente e in modo semplice nella propria applicazione WEB. Essa presenta dapprima le motivazioni che hanno portato alla stesura,

si concentra sul funzionamento dell'algoritmo, sul lavoro di implementazione svolto, sul testing fino alla presentazione di un possibile utilizzo con una demo.

1.1 Motivazioni

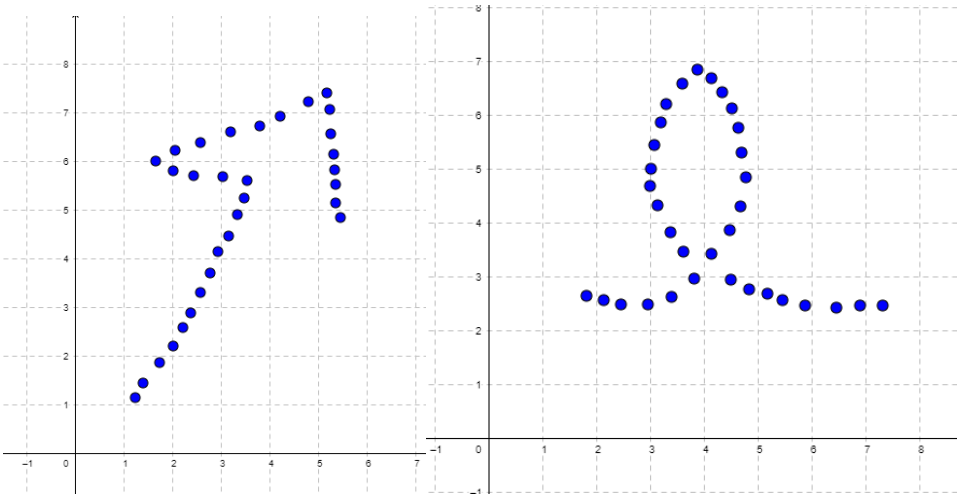
La tesi nasce con l'obiettivo di rendere disponibile una libreria JavaScript equivalente, di poche righe di codice e di facile utilizzo, per il riconoscimento di gesture a 2 dimensioni. I punti a favore sono vari:

- l'esecuzione del riconoscitore avviene lato Client senza necessità di un Server che ne esegua il codice.
- l'integrazione consiste nella sola inclusione della libreria come file JavaScript esteso o minimizzato.
- non necessita di particolari software per essere eseguito e testato, il browser di sistema è sufficiente per provare il riconoscitore.
- è utilizzabile anche lato server, tramite Node.js, nella progettazione di applicazioni di rete scalabili.
- può essere utilizzata per i scopi più disparati e per diversi tipi di piattaforme, essendo JavaScript un linguaggio che si presta a una moltitudine di progetti.
- può essere eseguito su un qualsiasi dispositivo su cui è installato un browser.

1.2 Gesture

Le gesture o gesti fanno parte del linguaggio del corpo, rappresentano l'aspetto più studiato e conosciuto della comunicazione non verbale. Nel campo dell'Informatica sono una combinazione di movimenti e click del dispositivo di puntamento, sia esso un mouse, una penna o le dita della mano che vengono riconosciuti dal software come specifici comandi. L'uso primario delle gesture

è quello di semplificare l'uso di funzioni comuni, richiamare shortcut o come nel caso degli smartphone, interagire con il sistema operativo.
Nella seguente tesi una gesture è rappresentata tramite una lista ordinata di punti. La lista di punti riassume informazioni quali lunghezza, rotazione, orientamento e il significato che viene attribuito alla gesture stessa. Un'ordine differente della stessa lista di punti può indicare una gesture completamente diversa, due esempi di definizione di una gesture mediante punti in figura 1.1.



(a) Freccia (b) Pigtail

Figura 1.1: Gesture rappresentate su un piano cartesiano a due dimensioni

Storicamente la prima gesture ideata per mouse è stata il drag and drop, introdotto dalla Apple per i computer Macintosh.

1.3 Gesture Recognition

La gesture recognition indica un tipo di tecnologia informatica e del linguaggio che ha l'obiettivo di interpretare i gesti umani attraverso algoritmi matematici. I gesti possono essere originati da qualsiasi posizione o movimento del corpo, ma comunemente provengono dalla mano. I processi informatici di riconoscimento dei gesti sono progettati per migliorare l'interazione uomo-computer e possono verificarsi in diversi modi, ad esempio attraverso l'uso di schermi tattili, una fotocamera, dispositivi periferici o sensori di

movimento per tracciare i movimenti dell'utente e tradurli in tempo reale. L'interesse verso le gesture è salito esponenzialmente nel tempo grazie anche alla nascita del touch screen. Nel campo delle gesture a tre dimensioni molto si deve ai sistemi di motion tracking e dispositivi quali il Microsoft Kinect [4] e il Leap Motion Controller [5].

1.3.1 Possibili applicazioni

Le possibili molteplici applicazioni della gesture recognition coprono i campi d'interesse più disparati. Tale tecnologia può, per esempio, consentire a più utenti di interagire con lo stesso dispositivo senza il timore di diffondere germi, molto utile in campo medicale. Possibili altre applicazioni [6]:

- **Riconoscimento del linguaggio dei segni**: software di riconoscimento dei gesti che possono trascrivere i simboli rappresentati attraverso il linguaggio dei segni in testo.
- **Robotica applicata all'assistenza sociale**: utilizzando sensori appropriati (accelerometri e giroscopi) applicati al corpo di un paziente e leggendo i valori generati da tali sensori è possibile aiutarne la riabilitazione.
- **Indicazioni direzionali attraverso il puntamento**: determinare in che direzione una persona sta puntando è utile per identificare il contesto delle dichiarazioni o delle istruzioni. Di particolare interesse nel campo della robotica.
- **Controllo attraverso gesti facciali**: controllare un computer attraverso i gesti facciali è una utile applicazione del riconoscimento dei gesti per gli utenti che possono non essere fisicamente in grado di utilizzare il mouse o la tastiera.
- **Interfacce alternative per il computer**.

- **Tecnologia di gioco immersiva**: controllare le interazioni all'interno dei videogiochi per cercare di rendere l'esperienza di gioco più realistica e coinvolgente.
- **Controller virtuali**: come un meccanismo di controllo alternativo.
- **Controllo remoto**: per controllare, attraverso il movimento della mano, vari dispositivi in remoto.

1.4 Organizzazione della Tesi

La seguente sezione descrive in breve come è stata strutturata la stesura del lavoro di tesi. Nel capitolo due vengono descritti i lavori correlati al progetto di tesi, le varie alternative nel campo del riconoscimento di gesture. Nel terzo capitolo viene descritto il funzionamento del riconoscitore PolyRec [2]. Nel capitolo quattro vengono descritte tutte le fasi di implementazione della libreria, dalle scelte di progettazione fino all'effettivo sviluppo. Il quinto capitolo è dedicato allo sviluppo della demo *PolyCoding*. L'ultimo capitolo, il sesto, è infine dedicato alle considerazioni conclusive riguardanti lo sviluppo della tesi.

CAPITOLO 2

LAVORI CORRELATI

Il seguente capitolo è dedicato alla descrizione di lavori di ricerca e algoritmi precedentemente sviluppati nel campo della gesture recognition. Il lavoro di tesi, essendo principalmente un porting, si basa sul riconoscitore esistente PolyRec, descritto in dettaglio nel Capitolo 3. Negli anni sono nati molteplici strumenti per sostenere lo sviluppo di applicazioni basate sui gesti. Le soluzioni proposte sono varie, da singoli algoritmi per il riconoscimento ad interi ambienti che supportano l'utente in tutte le fasi dell'utilizzo. I lavori correlati sono elencati dal meno recente al più recente:

- **DTW (Dinamic Time Warping) [7]**: Riconoscitore applicato al problema del riconoscimento della scrittura corsiva. I parametri utilizzati nell'abbinamento sono derivati da sequenze temporali di dati in coordinate X, Y di parole scritte a mano su una tavoletta elettronica. Principale vantaggio è che combina la segmentazione delle lettere e il riconoscimento in un'unica operazione, valutando il riconoscimento su tutte le possibili combinazioni.

- **Rubine [8]**: riconoscitore di gesti allenabile a singolo tratto.

- **$family**: un'insieme di riconoscitori di cui alcuni sono i seguenti:

$1 [9]: il $1 Unistroke Recognizer è un riconoscitore per gesture 2D a singola linea, progettato per la prototipazione rapida di interfacce utente basate su gesti. $1 è un classificatore che usa la tecnica del *"nearest-neighbor"* con una funzione per il calcolo della distanza euclidea durante il confronto. Nasce per consentire ai programmatori alle prime armi di incorporare i gesti nei loro prototipi di interfaccia utente.

$N [10] : il $N Multistroke Recognizer è il riconoscitore dell'insieme che riconosce gesture 2D multi-linea. Confronta tutti i possibili ordini e direzioni, il che significa che è possibile creare e definire tratti multipli utilizzando qualsiasi ordine e direzione, a condizione che si inizi da uno dei due punti finali di ciascun tratto.

$P [11]: il $P Point-Cloud Recognizer è un riconoscitore per gesture sia multi-linea che a singola linea. Tratta ogni gesture non come punti ordinati ma come una "nuvola" di punti non ordinati. $P cerca una soluzione al classico problema dell'assegnazione tra due grafi bipartiti per trovare il miglior matching.

$Q [12]: un rapido riconoscitore a tratto unico point-cloud per dispositivi mobili, indossabili e incorporati con risorse di calcolo ridotte. $Q funziona fino a 142 volte più velocemente del suo predecessore $P.

$P+ [13]: utilizzato per il riconoscimento dei gesti per migliorare l'accessibilità del touch screen per le persone ipovedenti.

- **Quick$** [14]: è un'estensione di Dollar Recognizer progettata per migliorare l'efficienza del riconoscimento. Quick$ utilizza il clustering gerarchico insieme alla ricerca per rami per un riconoscimento più efficiente.

- **1¢** [15]: un semplice ed efficiente riconoscitore per gesture tracciate a mano. Tecnica resa possibile da una rappresentazione unidimensionale delle gesture senza necessità di pre-elaborazioni.

- **Jackknife** [16]: riconoscitore usabile per la prototipazione rapida generale di gesti dinamici che possono essere addestrati con pochi campioni, lavorare con dati continui e ottenere un'elevata precisione indipendentemente dalla modalità.

- **FTL** [17]: riconoscitore che utilizza anch'esso la tecnica del *"nearest-neighbor"*, calcola la similarità tra una gesture in input e un dataset di gesture. Per evitare una costosa fase di pre-elaborazione, normalizzazione e ricampionamento, ogni gesture viene definita sotto forma di vettori e il calcolo della similarità avviene tramite la *Local Shape Distance* tra i vettori.

- **Protractor** [18]: riconoscitore che utilizza l'approccio del *"nearest neighbor"*, riconosce una gesture sconosciuta in base alla similarità con ciascuna delle gesture note. Similarità calcolata tramite la minima distanza angolare tra le gesture in esame.

- **Quill** [19]: software Java nato per aiutare i progettisti di interfacce utente basate su penna, può consentire quindi l'uso e la definizione di gesture nei propri progetti. Le funzionalità supportate includono la possibilità di aggiungere gesture, riconoscere, modificare e organizzare set di gesture.

- **GestMan [20]**: strumento web-based per la gestione e l'acquisizione di gesture 2D/3D. Oltre a consentire agli utenti di raccogliere gesture per creare i propri set, supporta la condivisione e la gestione delle stesse. Fornisce inoltre funzioni di elaborazione (ad es. rotazione, traslazione, ridimensionamento), analisi e riconoscimento. I set di gesti possono essere poi esportati in formato JSON.

- **PolyRec GDT [21]**: uno strumento che consente la definizione di set di gesture per applicazioni mobili. Supporta l'intero processo di progettazione del set, dalla raccolta delle gesture all'analisi e all'esportazione del set creato per l'uso in un'applicazione mobile. Utilizza il riconoscitore di gesti PolyRec, pur supportando l'uso di

altri riconoscitori. Le caratteristiche principali includono la capacità di raccogliere le gesture direttamente sui dispositivi mobili, di rilevare le somiglianze tra le gesture per aiutare il progettista a rilevare le ambiguità nella fase di progettazione e la possibilità di selezionare automaticamente i gesti più rappresentativi per ogni classe di gesture tramite tecniche di clustering.

CAPITOLO 3

POLYREC

Il capitolo seguente illustra il funzionamento del riconoscitore pre-esistente PolyRec [2] in tutte le sue fasi, spiegando nel dettaglio gli algoritmi utilizzati e le decisioni prese.

Polyrec utilizza l'approccio del *nearest neighbor*, la ricerca del vicino più vicino, come forma di ricerca di prossimità che consiste nel trovare il punto in un dato insieme che è più vicino (o più simile) a un punto dato. Un buon vantaggio è che il riconoscitore richiede solo un piccolo numero di esempi per ogni classe di gesture e può funzionare in tempo reale su qualsiasi dispositivo. La similarità tra due gesture viene calcolata con una procedura divisa in tre passi sequenziali:

1. **Pre-elaborazione**: ogni gesture viene approssimata ad una polilinea per estrarne i movimenti principali.

2. **Allineamento**: le due polilinee vengono allineate per ottenere un pari numero di segmenti.

3. **Esecuzione**: la distanza, in termini di somiglianza tra polilinee, viene calcolata sommando i contributi di ogni coppia di segmenti.

Durante la fase di pre-elaborazione, da una gesture vengono estratti una piccola serie di punti dominanti. Ogni gesture viene quindi rappresentata come una **polilinea**: un insieme finito e totalmente ordinato di segmenti che identificano i principali movimenti della gesture. Ogni qual volta si confrontano due gesture si cerca la migliore corrispondenza punto a punto, le due polilinee vengono ridotte al medesimo numero di segmenti. A questo proposito, durante il campionamento iniziale, vengono aggiunti all'occorrenza nuovi punti ad una o ad entrambe le gesture.

Per trovare la distanza tra due gesture, i segmenti di ogni polilinea vengono trattati come **vettori** composti da una coppia di valori indicanti intensità ed angolazione. I vettori vengono quindi confrontati con quelli dell'altra polilinea per il calcolo dell'effettiva distanza.

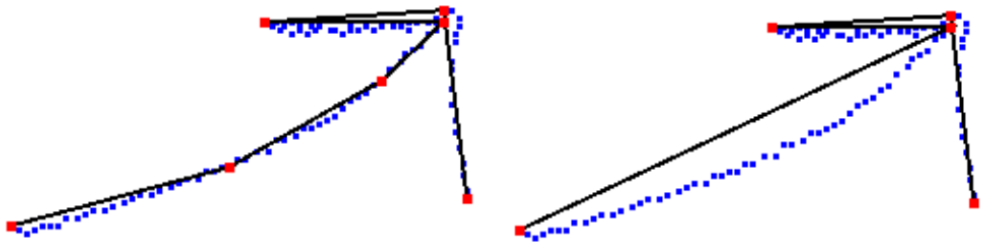
3.1 Pre-elaborazione

Prima di qualsiasi elaborazione, una gesture di input contiene i punti campionati dal dispositivo, che sono in numero variabile e di solito si ottengono catturando la posizione del puntatore (dito o penna o mouse) a intervalli di tempo regolari. I punti campionati in questa fase sono i cosiddetti **punti dominanti**, ossia i punti dove la gesture mostra una curvatura maggiore. Il risultato è una linea spezzata, quella che viene definita *polilinea*.

3.1.1 Elicitazione e riconoscimento

La gesture in input può contenere troppi punti, considerabili inutili ai fini del riconoscimento, motivo per cui avviene una riduzione del numero di punti con il riduttore di Douglas-Peucker [22]. Lo scopo è quello di ridurre il numero di punti, rimuovendo quelli non necessari e conservando quelli che sono sufficienti a rappresentare la gesture. Inizialmente vengono selezionati gli estremi di ogni curva, la polilinea derivante sarà quindi formata dai segmenti che connettono gli estremi. Nel passo successivo, preso ogni segmento, l'algoritmo divide la singola curva in due nel punto più lontano dal segmento e

lo aggiunge alla polilinea risultante. Ricorsivamente viene effettuata la stessa operazione sulle due parti della curva. La curva viene interrotta solo se il punto selezionato è ad una distanza maggiore rispetto ad una certa soglia di tolleranza.
Nella figura 3.1a viene mostrato un esempio dell'applicazione del riduttore di Douglas-Peucker su una gesture rappresentante una freccia. Per ottenere una polilinea con minori punti estremi viene aggiunta una ulteriore fase di fusione. Ad ogni iterazione, il punto con il più piccolo cambio di direzione viene rimosso. Il processo termina quando il più piccolo cambio di direzione supera una soglia minima θ. La figura 3.1b mostra il risultato della fusione: in due iterazioni consecutive, due punti campionati vengono rimossi dall'asta della freccia.



(a) Riduzione Douglas-Peucker (b) Fusione delle linee

Figura 3.1: I punti campionati (evidenziati in rosso) su una freccia [2]

Nel caso peggiore la complessità del riduttore Douglas-Peucker [22] è $O(N^2)$ con N rappresentante il numero di punti iniziali della gesture in input. La fase di pre-elaborazione viene eseguita una sola volta su ogni template, in modo tale da non appesantire eccessivamente la procedura di riconoscimento.

3.2 Allineamento

L'allineamento è la fase in cui due polilinee vengono approssimate allo stesso numero di segmenti per consentire un confronto alla pari. Nella precedente fase di pre-elaborazione, la polilinea è composta da un numero variabile di segmenti che collegano i punti dominanti della gesture. In questa

fase viene stabilita una **mappatura** tra i punti campionati della gesture in input e della gesture del template. La mappatura si basa sulla somiglianza tra due vertici delle rispettive polilinee, somiglianza che si determina mediante due parametri:

- $pos(p)$: la posizione del punto all'interno della gesture, calcolata come rapporto tra la lunghezza fino a p e la lunghezza totale.
- $angle(p)$: la differenza di direzione tra i due segmenti della polilinea che si incontrano al vertice p.

In alcuni casi è possibile che uno o più vertici di una polilinea non abbiano una corrispondenza sull'altra polilinea e viceversa. Per ogni vertice non mappato viene aggiunto, nell'altra polilinea, un vertice aggiuntivo.

3.2.1 Needleman-Wunsch

Per effettuare l'allineamento, PolyRec utilizza una versione modificata dell'algoritmo di Needleman-Wunsch [23], un algoritmo utilizzato, in software per la Bio-Informatica, per l'allineamento di stringhe di nucleotidi o proteine. L'algoritmo calcola l'allineamento ottimale tra due stringhe basandosi su uno schema di punteggi. Vengono valutati i possibili allineamenti tramite una **funzione di similarità** e viene scelto il valore risultante più simile. L'allineamento risultante consta di due stringhe di uguale lunghezza, dove due lettere corrispondenti sono uguali (caso in cui si ha un *match*) o diverse (caso in cui si ha un *mismatch*). Nell'ultimo caso, una lettera viene allineata con uno spazio vuoto (*gap*) nell'altra stringa o viceversa.

3.2.2 Variante utilizzata

Nel caso di PolyRec la stessa idea è utilizzata per allineare i punti di due gesture. In particolare, l'obiettivo è associare due vertici delle polilinee in esame solo se sono abbastanza simili (*match*) oppure nel caso contrario (*mismatch*), campionare un nuovo punto per una di esse.

Date due polilinee P con vertici $p_1, p_2, ..., p_n$ e Q con vertici $q_1, q_2, ..., q_m$ si definisce la funzione di similarità ricorsivamente come segue:

- Caso base :

  $F_{i,0} = i\, x\, g \;\; for\, i = 0, ..., n$

  $F_{j,0} = j\, x\, g \;\; for\, i = 0, ..., m$

- Caso ricorsivo

$$F_{i,j} = \max \begin{cases} F_{i-1,j-1} + similarity(p_i, q_j), & Match(p_i, p_j) \\ F_{i,j-1} + g, & Insert(P, q_j) \\ F_{i-1,j} + g, & Insert(Q, p_i) \end{cases} \tag{3.1}$$

  $for\ i\ =\ 1, ..., n;\ j\ = 1, ..., m$

dove la funzione $similarity(p_i, q_j)$ calcola l'indice di similarità tra p_i e q_j. Il valore g indica la *gap penalty*, ossia il costo da aggiungere in caso di inserimento in P o in Q. La similarità è calcolata con la seguente formula:

$$similarity(p_i, q_j)\ =\ 1 - [b\ *\ pDiff(i.j)\ +\ (1 - b)\ *\ aDiff(i, j)] \tag{3.2}$$

dove:

$$pDiff(i, j)\ =\ |pos(p_i)\ -\ pos(q_j)| \tag{3.3}$$

$$aDiff(i, j)\ =\ \frac{|angle(p_i)\ -\ angle(q_j)|}{2\pi} \tag{3.4}$$

e b è una costante che determina il bilanciamento del peso delle due caratteristiche (pos e angle). L'allineamento, formato da coppie di vertici mappati, viene determinato trovando il *percorso di corrispondenza massima* nella matrice costruita da F (per una descrizione più dettagliata [23]): partendo dalla cella in alto a sinistra si scende verso la cella in basso a destra, calcolando il valore di ogni cella con l'equazione 3.1 durante il percorso. In particolare:

- $Match(p_i, q_j)$: se il vertice p_i di P e il vertice q_j di Q coincidono, allora la coppia (p_i, q_j) viene aggiunta all'allineamento.

- $Insert(P, q_j)$: un nuovo vertice mappato con q_j viene aggiunto a P e la coppia $(-, q_j)$ viene inserita nell'allineamento.

- $Insert(Q, p_i)$: un nuovo vertice mappato con p_i viene aggiunto a Q e la coppia $(p_i, -)$ viene inserita nell'allineamento.

L'inserimento di un vertice in una polilinea avviene tra i vertici più vicini che hanno una corrispondenza nell'altra polilinea.
Ad esempio, dall'allineamento $...(p'_m, q''_m)(-, q_j)(p''_m, q''_m)...$ si può determinare che un nuovo vertice p mappato con q_j deve essere aggiunto in P fra i vertici p'_m e p''_m.
La posizione ideale in cui aggiungere il nuovo vertice campionato p nella sub-gesture delimitata da p'_m e p''_m è stabilita in proporzione rispetto alla posizione di q_j nella sub-gesture da q'_m a q''_m. Viene quindi campionato il punto che più corrisponde alla posizione calcolata. La formula generale utilizzata per il calcolo della posizione è la seguente:

$$pos(p) = pos(p'_m) + \frac{length(q'_m, q_j) * length(p'_m, p''_m)}{length(q'_m, q''_m)} \tag{3.5}$$

Un esempio di allineamento di polilinee è mostrato in figura 3.2 e 3.3. Le due figure mostrano il risultato dell'allineamento su due gesture ridotte rispettivamente a due polilinee P e Q.
P e Q contengono rispettivamente 8 e 9 vertici, l'allineamento risultante dall'algoritmo di Needleman-Wunsch è, dato l'esempio, il seguente:
$(p_1, q_1)\,(p2, -)\,(q_3, q_2)\,(-, q_3)\,(p_4, q_4)\,(p_5, q_5)\,(p_6, q_6)\,(-, q_7)\,(p_7, q_8)\,(p_8, q_9)$ e q_7 viene inserito in P e il corrispondente di p_2 è aggiunto in Q.

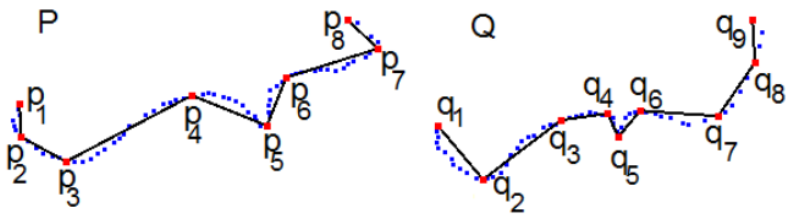


Figura 3.2: Prima dell'allineamento [2]

Nella figura 3.3, dopo l'allineamento, due nuovi vertici (cerchiati in rosso) vengono aggiunti in P ed uno in Q.

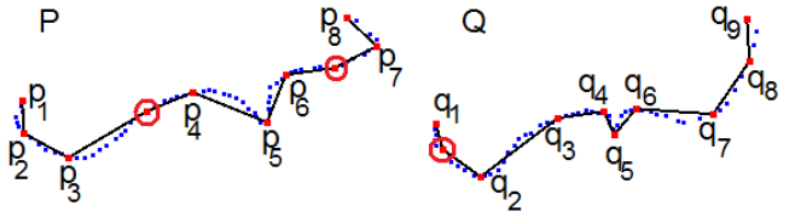


Figura 3.3: Dopo l'allineamento [2]

3.3 Distanza

Nella terza fase, dopo l'allineamento, viene calcolata la distanza tra le due gesture. Per ogni polilinea si deriva una sequenza di vettori: ogni segmento della polilinea è descritto come un vettore di due componenti: *intensità* ed *angolazione*. La grandezza è pari alla lunghezza relativa del segmento e la direzione è rappresentata dalla pendenza del segmento, cioè l'angolo tra il segmento e l'asse orizzontale. La grandezza dell'i-esimo vettore corrisponde all'i-esimo segmento della polilinea P e viene calcolata con la seguente formula:

$$\vec{P_i} = \frac{length(P_i)}{length(P)} \tag{3.6}$$

Le due gesture sono quindi approssimate come polilinee aventi lo stesso numero di segmenti. La distanza viene calcolata sommando l'ampiezza dei vettori ottenuta dalla differenza di ciascuna coppia nelle corrispondenti sequenze di vettori. Considerate due polilinee P e Q, entrambe composte da n segmenti, la loro distanza viene calcolata con la seguente formula:

$$Distance(P,Q) = penalty * \sum i = 1^n |\vec{P_i} - \vec{Q_i}| \tag{3.7}$$

dove penalty è un fattore calcolato sulla base del numero di *match* e *mismatch* avuti nella fase di allineamento: $penalty = 1 + \frac{mismatch}{matches + mismatches}$.
Il calcolo del valore *penalty* viene utilizzato per aumentare la distanza D proporzionalmente al numero di *mismatch* ottenuti durante l'allineamento.

3.3.1 Rotazione

Prima del calcolo della distanza, il riconoscitore effettua un'operazione di rotazione, i gesti vengono ruotati per ridurre il *rumore* nell'orientamento.

Viene determinato l'angolo indicativo della gesture, definito come la distanza fra centroide e primo punto della gesture. PolyRec può essere utilizzato sia in un contesto sensibile alla rotazione sia in un contesto in un cui la rotazione è invariante. Nel primo caso, la gesture viene ruotata a 0, nel secondo caso viene adottata la strategia usata in [24]: l'angolo indicativo di una gesture viene allineato con la direzione più vicina rispetto agli otto orientamenti principali di una gesture. Come ultimo step viene utilizzata la *Golden Section Search* per determinare la distanza rispetto al miglior angolo.

3.3.2 Setting dei parametri

Per stabilire i valori dei parametri di default, riassunti nella tabella 3.1, è stata eseguita una fase di perfezionamento su un dataset composto da 400 curve [24], precedentemente utilizzato per testare algoritmi per la ricerca di angoli nelle curve digitali. A tal fine sono stati misurati i valori di una funzione obiettivo in corrispondenza di tutte le possibili combinazioni di valori di parametri (discretizzati a passi regolari e variati entro intervalli plausibili). Per evitare un'esplosione combinatoria, ossia la rapida crescita della complessità del problema, sono stati sintonizzati separatamente i parametri per la pre-elaborazione e quelli per l'algoritmo di Needleman-Wunsch.

Tabella 3.1: Parametri di default per il riconoscitore [2]

Parametri	Descrizione	Valore
t	tolleranza in Douglas-Peucker	$\frac{diagonale\ gesture}{26}$
θ	soglia per la fusione dei segmenti	22 gradi
g	penalty nell'equazione 3.1	0.4
b	bilanciamento nell'equazione 3.2	0.6

Per la fase di pre-elaborazione, la funzione obiettivo (da minimizzare) è stata scelta come la differenza assoluta tra il numero ideale di punti campionati (ad esempio, per una polilinea che rappresenta un triangolo, il numero ideale

di punti campionati è 4: le due estremità e i due angoli interni) e il numero di punti campionati attraverso la pre-elaborazione del riconoscitore. Sono stati quindi scelti i valori di t e θ che minimizzano la differenza, calcolata come media su tutte le curve. Per stabilire il valore dei parametri utilizzati nell'algoritmo di Needleman-Wunsch, è stato semplicemente eseguito il riconoscitore sull'intero dataset e sono stati scelti i valori per g e b che massimizzavano l'accuratezza del riconoscimento.

3.3.3 Prestazioni

Il riconoscitore PolyRec è stato testato, in un contesto in cui la rotazione è invariante, su due differenti dataset. Le performance sono state anche confrontante con Protractor [18], il dataset utilizzato è il medesimo di [9]. Per ogni utente, velocità e classe di gesture, sono stati scelti in maniera casuale dei template ed eseguite su di essi 100 prove. Il confronto è stato effettuato sia applicando la Golden Section Search che non applicandola.

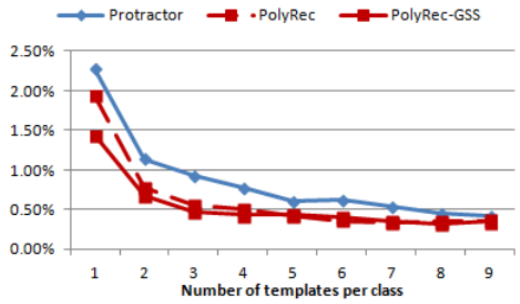


Figura 3.4: Tasso di errore in base al numero di templates per ogni classe sul dataset [2]

Il grafico mostra un chiaro vantaggio in termini di accuratezza del riconoscimento per PolyRec. Il vantaggio, sia con che senza Golden Section Search, è altamente significativo.

CAPITOLO 4

POLYREC.JS

Il seguente capitolo descrive lo sviluppo della libreria PolyRec.js per tutte le sue tre fasi: progettazione, sviluppo e testing.

4.1 Progettazione

L'obiettivo, preposto durante la fase di progettazione, è lo sviluppo di una libreria JavaScript capace di assicurare i medesimi risultati della versione Java [3] ma che potesse assicurare gli stessi vantaggi ed un uso mirato nel Web Development. La scelta di JavaScript come linguaggio di programmazione rende la libreria eseguibile su tutti i maggiori Browser (Chrome, Safari, Firefox, Edge...) e pertanto testabile su un qualsiasi dispositivo provvisto di Browser.

4.1.1 Requisiti

Da una prima analisi iniziale sono nati i requisiti funzionali, relativi alle funzionalità della libreria e i requisiti non funzionali, relativi alle caratteristiche prestazionali. Data la grandezza non elevata del progetto, i requisiti hanno tutti la stessa priorità.
Requisiti funzionali:

- La libreria deve permettere di riconoscere una gesture data in input.
- La libreria deve permettere di poter caricare nuovi template.
- La libreria deve permettere di poter rimuovere template inseriti.
- La libreria deve permettere di poter utilizzare un dataset iniziale pre-caricato.
- La libreria deve permettere di caricare più gesture d'esempio per un singolo template.

Requisiti non funzionali:

- La libreria deve assicurare i medesimi risultati della versione Java [3] pre-esistente.
- La libreria deve assicurare prestazioni e tempi di risposta comparabili a quelli della versione Java pre-esistente.
- La libreria deve essere utilizzabile ed inclusa in progetti esterni, in modo semplice, anche da programmatori novizi.
- La libreria deve poter offrire le proprie funzionalità nascondendo all'utente la parte implementativa.
- La libreria deve essere scalabile ed eseguibile su tutti i browser più utilizzati.
- La libreria deve essere utilizzabile sia lato FrontEnd che lato BackEnd senza alcuna differenza di risultati.
- La libreria non deve utilizzare librerie o moduli esterni.

4.1.2 Caso d'uso

Nel ramo dell'ingegneria del software un caso d'uso individua e descrive gli scenari elementari di utilizzo del sistema, in questo caso della libreria, da

parte degli utenti che si interfacciano con esso/a. Un possibile caso d'uso della libreria PolyRec.js può essere il seguente:

1. L'utente, interessato all'uso della libreria, provvede a scaricare quest'ultima in versione minimizzata o non.
2. L'utente include nel proprio progetto la libreria.
3. L'utente può optare tra due modalità di impostazione:
   (a) L'utente può caricare, con un metodo apposito, i suoi template per le gesture.
   (b) L'utente può utilizzare il set di gesture presenti di default nella libreria.
4. L'utente sceglie come raccogliere le coordinate dei punti della gesture in input.
5. L'utente, attraverso la funzione principale di riconoscimento, dà in input la lista di punti al riconoscitore.
6. L'utente userà il risultato del riconoscimento, contenente informazioni quali: nome della gesture riconosciuta, punteggio del matching ottenuto e tempo necessario al riconoscimento.

4.1.3 JavaScript

Come pre-annunciato dal nome stesso della tesi, la libreria è completamente scritta in JavaScript. JavaScript, abbreviato con JS, è un linguaggio di programmazione interpretato ad alto livello. Standardizzato per la prima volta nel 1997 con il nome di ECMAScript, è insieme all'HTML e al CSS una delle tecnologie alla base del World Wide Web. Di seguito vengono mostrati alcuni dei punti di forza di JavaScript, che spiegano in parte la scelta adottata.

- **Librerie**: oltre quelle standard, JavaScript offre una moltitudine di librerie sviluppate da una community molto attiva, per i scopi più disparati, dalla manipolazione del *Document Object Model* (DOM) alla gestione di database fino al riconoscimento delle gesture.

- **Supporto universale**: tutti i moderni browser web supportano nativamente JavaScript tramite l'utilizzo di un interprete.

- **Multi-paradigma**: JavaScript è un linguaggio multi-paradigma in quanto supporta i più utilizzati paradigmi di programmazione: la programmazione guidata dagli eventi, la programmazione funzionale, quella imperativa, inclusa la programmazione orientata agli oggetti.

- **Portabile**: JavaScript è un linguaggio altamente portabile, può essere utilizzato su diverse piattaforme e sistemi operativi: *Linux, Windows, OSx, iOS, Android*. In quanto linguaggio interpretato e non compilato, funziona su qualsiasi macchina sia presente un interprete.

- **Semplice**: JavaScript ha una curva di apprendimento non troppo ripida, è adatto come linguaggio accademico in quanto è ad alto livello.

- **Gratuito**: JavaScript è totalmente gratis, è possibile utilizzarlo e distribuirlo senza alcuna restrizione di copyright.

- **Sia Front-End che Back-End**: nonostante JavaScript nasce come linguaggio Client-side, è stata successivamente introdotta una nuova implementazione per lo scripting server-side, la più conosciuta è Node.js.

4.1.4 Class Diagram

Un class diagram fornisce una vista strutturale (statica) del sistema in termini di classi, mostrandone gli attributi e le operazioni eseguibili su di essi. Esso mostra anche le relazioni (associazione, generalizzazione,...) tra le varie classi di un sistema.

Nella seguente sotto sezione verrà mostrata la differenza tra il class diagram derivante dalla versione Java di PolyRec e quello della versione JavaScript.

Dato che Java è un linguaggio orientato agli oggetti, la struttura del riconoscitore è un insieme di dichiarazione di classi che interagiscono tramite metodi e funzionalità. La versione JavaScript è stata pensata per ottimizzare l'uso delle classi: vengono definite solo le classi principali, mostrate in figura 4.1. In figura 4.2 viene invece mostrato il class diagram generato dalla versione Java di PolyRec. Le classi presenti, scelte in fase di progettazione, sono quelle le cui istanze sono essenziali per il procedimento.

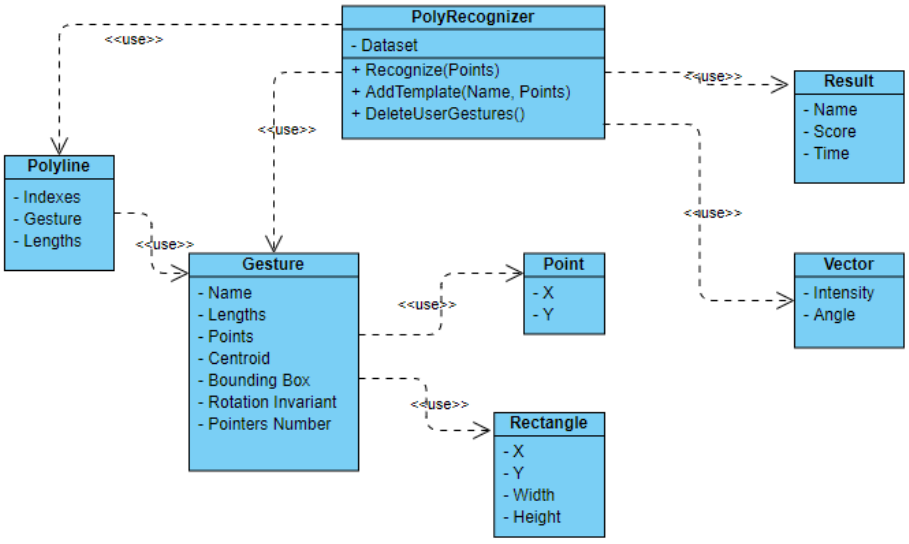


Figura 4.1: Class diagram libreria

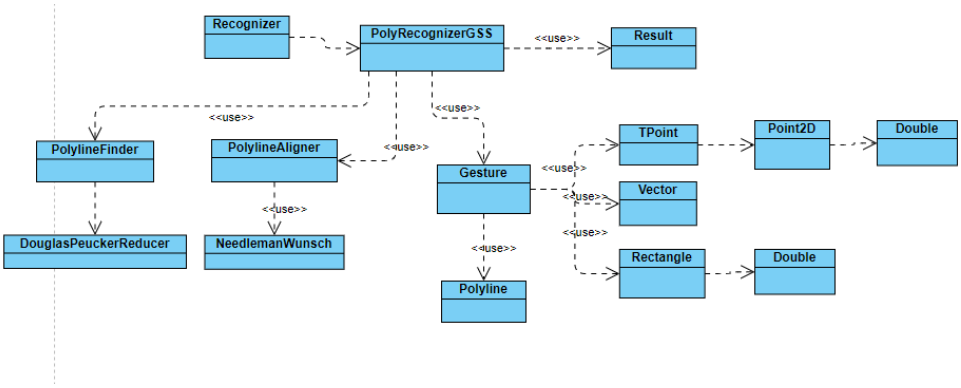


Figura 4.2: Class diagram versione Java

4.2 Sviluppo

La fase di sviluppo della libreria segue il class diagram in figura 4.1, soprattutto per la definizione delle principali classi. JavaScript, a differenza di Java, non necessità di una struttura a classi pertanto sono state scelte solo le classi utili all'elaborazione del riconoscitore.

- **Point**: un punto, rappresentato dalle sue coordinate X, Y su un piano cartesiano a due dimensioni.

```
1 function Point(x, y){
2     this.X = x;
3     this.Y = y;
4 }
```

- **Rectangle**: un rettangolo bidimensionale, rappresentato dalle coordinate X, Y del punto in alto a sinistra, grandezza ed altezza.

```
1 function Rectangle(x, y, width, height){
2     this.X = x;
3     this.Y = y;
4     this.Width = width;
5     this.Height = height;
6 }
```

- **Vector**: un vettore, rappresentato da due fattori: intensità ed angolazione.

```
1 function Vector(intensity, angle){
2     this.Intensity = intensity;
3     this.Angle = angle;
4 }
```

- **Result**: oggetto che rappresenta il risultato del riconoscimento con 3 campi: nome della gesture riconosciuta, score del matching e tempo di esecuzione in secondi.

```
1 function Result(name, score, time){
2     this.Name = name;
3     this.Score = score;
4     this.Time = time;
5 }
```

- **Gesture**: rappresenta la singola gesture. È caratterizzata da un nome identificativo, una lista di punti, il centroide della lista di punti, le distanze tra ogni punto della lista e la bounding box della gesture ossia il rettangolo con la misura più piccola entro cui sono contenuti tutti i punti.

```
1 function Gesture(name, points){
2     this.Name = name;
3     this.Points = points;
4     this.Centroid = CalculateCentroid(points);
5     this.Lengths = CalculateLengths(points);
6     this.BoundingBox = CalculateBoundingBox(points);
7 }
```

- **Polyline**: rappresenta una polilinea, un insieme finito ed ordinato di segmenti, risultante dalla riduzione di una gesture. È rappresentata dalla gesture di cui ne è la riduzione, una lista di indici che indica i punti salvati post riduzione e una lista delle distanze tra i punti ridotti.

```
1 function Polyline(gesture. indexes){
2     this.Gesture = gesture;
3     this.Indexes = indexes;
4     this.Lengths = CalculateLenghts(gesture.Lenghts, indexes);
5 }
```

- **PolyRecognizer**: è la classe principale, il riconoscitore. Ha un campo DataSet per memorizzare i template come coppie nome/lista di punti e 3 metodi principali:

  Recognize(): riconosce la gesture come lista di punti passata in input.

  AddTemplate(): aggiunge un nuovo template al riconoscitore come coppia nome/lista di punti.

  DeleteUserGestures(): rimuove tutti i template utente inseriti.

```
1 function PolyRecognizer(default){
2     this.DataSet = new Array();
3     this.Recognize = function(points){...}
4     this.AddTemplate = function(name, points){...}
5     this.DeleteUserGestures = function(){...}
6 }
```

Una volta definite le classi necessarie, lo sviluppo della libreria è avvenuto in due parti. Nella prima parte le funzioni sono state divise in maniera logica per argomento, un file JavaScript per ogni insieme di funzioni comuni che lavorano sullo stesso insieme di dati. Nella seconda parte tutti i moduli sono stati uniti, previa operazione di testing e refactoring, in un unico file JavaScript *"polyrec.js"*.

4.2.1 Funzioni geometriche

Il primo modulo sviluppato è quello che contiene le funzioni minori di tipo geometrico. Sono presenti funzioni per il calcolo del centroide, della bounding box, di cui un esempio in figura 4.3 ,per il calcolo della distanza tra punti o tra vettori e così discorrendo. Sono funzioni di base utilizzate da tutti i moduli superiori. Il modulo contiene inoltre la definizione delle classi minori quali *Point*, *Vector*, *Rectangle* e *Result*.

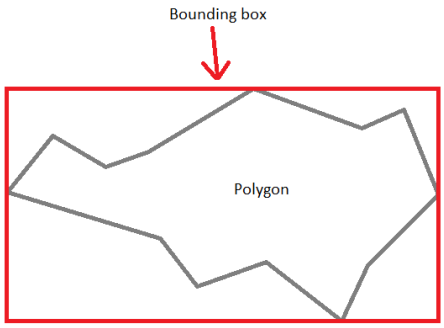


Figura 4.3: Bounding Box di un poligono

4.2.2 Polyline

Modulo che contiene la definizione delle classi necessarie alla libreria. Sono presenti le classi *Gesture*, *Polyline* e tutte le funzioni che sono utilizzate per la gestione delle caratteristiche delle due classi. Sono comprese funzioni per il calcolo della diagonale della gesture, per l'angolo indicativo o per la trasformazione della gesture in una sequenza di vettori.

4.2.3 Douglas Peucker

Modulo che contiene tutte le funzioni da applicare per l'algoritmo di Douglas-Peucker [22]. L'algoritmo riduce il numero dei punti della gesture lasciando invariato ciò che la gesture rappresenta. La riduzione avviene secondo una tolleranza, calcolata come differenza fra la diagonale della bounding box della gesture e un parametro prestabilito.

Per ogni punto della gesture viene calcolata la distanza ortogonale rispetto al segmento che connette primo e ultimo punto della gesture. Ricorsivamente viene etichettato per essere rimosso il punto la cui distanza risulta maggiore della tolleranza.

In un secondo momento, dopo la prima rimozione, vengono rimossi iterativamente i punti che sono al vertice del minimo angolo nella gesture. Completata la fase di rimozione il modulo ritorna la risultante polilinea con i punti ridotti.

4.2.4 Needleman Wunsch

Modulo che su due polilinee esegue l'allineamento dei punti con una versione modificata dell'algoritmo di Needleman-Wunsch [23]. Siano l_1 ed l_2 rispettivamente il numero di segmenti delle due polilinee da confrontare, viene creata una matrice di dimensioni $l_1 * l_2$. La matrice è utilizzata per eseguire l'algoritmo (maggiori dettagli nel 3) e contare il numero di punti aggiunti per ognuna delle polilinee. Il modulo restituisce il numero di match avuti e una lista contenente tutti i punti che hanno avuto un matching e le cui coordinate sono strettamente positive.

4.2.5 Polyline Aligner

Modulo che esegue l'allineamento di due gesture. Due gesture vengono date in input al modulo Needleman-Wunsch, come risultato si ottiene la lista di punti che hanno generato un matching. Iterativamente si inserisce nella polilinea uno o più punti con lo scopo di ottenere due gesture con uguale numero di segmenti su cui poi calcolare la distanza.

4.2.6 Recognizer

Modulo principale, contiene la classe PolyRecognizer e i 3 metodi ad essa associati. Un attributo DataSet memorizza i template che il riconoscitore dovrà utilizzare. Sono presenti i 3 metodi principali che permettono di aggiungere un nuovo template, cancellare quelli presenti ed, il più importante, riconoscere una gesture in input.

4.2.7 Esecuzione

Dopo aver testato le singole componenti, di cui si fa riferimento nelle sezioni successive, è stato eseguito un processo di unione in un singolo file *"polyrec.js"*, quella che sarà la libreria. Di seguito viene descritto, in dettaglio, il percorso d'esecuzione dell'utilizzo della libreria, percorso che tocca tutti gli aspetti, le funzioni e le classi implementate.

1. Viene creata una nuova istanza del riconoscitore *var recognizer = new PolyRecognizer(b)*. Sono previste tre opzioni tenendo nota che l'argomento b è un booleano:

   se b = true nell'istanza del riconoscitore vengono caricate le gesture inserite di default;

   se b = false nel riconoscitore non vengono caricate gesture iniziali;

   terza alternativa prevede di poter creare un'istanza del riconoscitore senza argomenti (*var recognizer = new PolyRecognizer()*) ed equivale ad un'istanza in cui non vengono caricate gesture iniziali.

2. L'utente può caricare i propri template con *recognizer.AddTemplate(name, points)*. Nel riconoscitore non è la gesture ad essere memorizzata ma la relativa polilinea come coppia nome/polilinea, dove il nome è l'identificativo.

3. Per effettuare il riconoscimento di una gesture di input viene richiamato il metodo *recognizer.Recognize(inputpoints)*.

4. Dai punti in input viene creata una istanza di gesture e generata con la funzione *DouglasPeucker(gesture)* la relativa polilinea.

5. Iterativamente viene confrontata ed allineata la polilinea di input con ognuno dei template attraverso la funzione *AlignPolylines(input, template)*. Si ricava dal confronto una penalità calcolata in base ai vertici che sono stati aggiunti o rimossi durante l'allineamento.

6. Le due polilinee risultanti sono utilizzate per calcolare la migliore distanza (*DistanceAtBestAngle(input, template)*): per prima cosa viene calcolato, per ognuna delle due gesture, l'angolo indicativo, ossia l'angolo tra il centroide e il primo punto della gesture, di cui un esempio in figura 4.4. In secondo momento ogni polilinea viene trasformata in una lista di vettori, ogni vettore sarà definito da angolazione e intensità. Iterativamente viene calcolata la distanza tra l'i-esimo vettore della gesture di input e l'i-esimo vettore della gesture template.

L'iterazione va avanti finché la differenza delle distanze è maggiore di una certa tolleranza pre-calcolata. La distanza finale risultante sarà quella con valore minore. In ultimo momento viene calcolata la distanza migliore, ossia il prodotto fra la distanza calcolata nel punto precedente e la penalità. Questo valore sarà utilizzato per calcolare lo score finale.

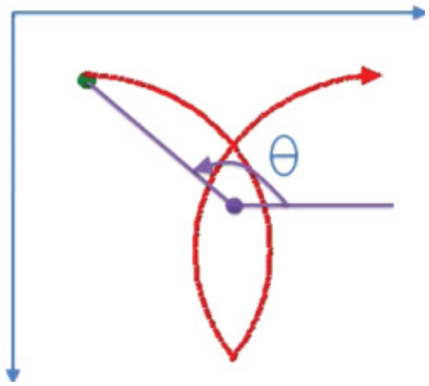


Figura 4.4: Esempio angolo indicativo di una gesture [25]

7. Viene memorizzato il nome del template che ottiene score maggiore.

8. *recognize.Recognize()* restituisce infine il risultato contenente nome del template risultante, score (da 0 a 100) e tempo d'esecuzione in secondi.

4.3 Testing

La fase finale, come di consueto, riguarda il testing della libreria. Si divide in due parti: il testing delle singole componenti e il testing dell'intera libreria. Entrambi i test sono stati effettuati confrontando i risultati ottenuti con la controparte Java [3]. Il testing è stato effettuato sul medesimo insieme di gesture, utilizzato dal progetto [9]. L'insieme consta di 4800 gesture in formato XML, un esempio in figura 4.5, divise in base alla velocità con cui la gesture è stata disegnata: tratto lento, tratto medio, tratto veloce.
Sono presenti 16 template, come visibile dall'immagine 4.6, ognuno di essi ha 300 varianti, che si differenziano l'una dall'altra per velocità, rotazione e forma.

```
<?xml version="1.0" encoding="utf-8" standalone='
<Gesture Name="arrow01" Subject="1" Speed="fast"
  <Point X="69" Y="228" T="1363112" />
  <Point X="67" Y="228" T="1363125" />
  <Point X="70" Y="225" T="1363153" />
  <Point X="73" Y="223" T="1363161" />
  <Point X="75" Y="222" T="1363167" />
  <Point X="80" Y="218" T="1363174" />
  <Point X="82" Y="216" T="1363181" />
  <Point X="85" Y="214" T="1363182" />
  <Point X="92" Y="209" T="1363191" />
```

Figura 4.5: Gesture formato XML [9]

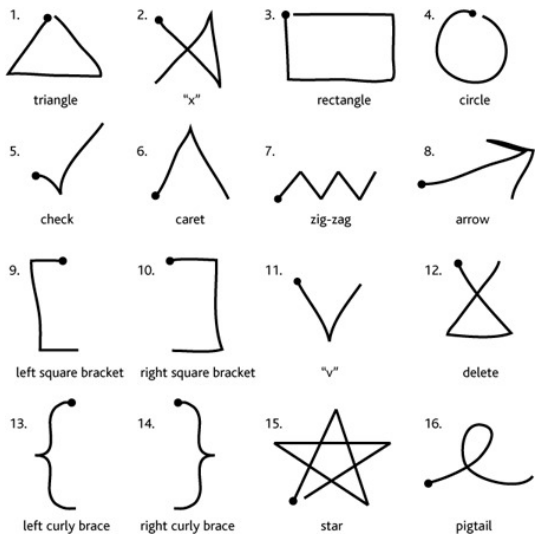


Figura 4.6: Templates di partenza [9]

La strategia di testing adottata è quella Bottom-Up, vengono prima collaudati i moduli di livello più basso nella gerarchia prodotta dalla progettazione, le unità più piccole ed elementari. Quando sono stati valutati correttamente i moduli minori, si passa ai moduli di livello superiore che vengono testati utilizzando le funzionalità del precedente livello, si risale quindi, continuando, fino alla libreria intera.

Nella fase di pre-testing, dopo aver istanziato un'istanza del riconoscitore, viene caricata un'unica gesture per ogni template. Il riconoscitore dovrà confrontare singolarmente ogni gesture in input con le 16 caricate in precedenza. Nella prima fase ogni funzione della libreria è stata testata singolarmente. Una volta che tutte le singole funzioni sono state testate, le

stesse sono state integrate in moduli più grandi e i moduli testati a loro volta. I 3 moduli testati separatamente sono relativi alla riduzione dei punti di una gesture(algoritmo di Douglas-Peucker), all'allineamento dei punti (algoritmo di Needleman-Wunsch) e all'allineamento delle polilinee.

Testate le 3 componenti, la libreria è stata testata interamente. In un'istanza del riconoscitore sono stati caricati 16 template, come sopra spiegato, ed è stato eseguito il metodo principale su ogni gesture del dataset. Il metodo principale del riconoscitore, *Recognize* permette di riconoscere una gesture in input partendo da una lista di punti; tocca tutte le funzioni e i moduli presenti permettendo di testare nel completo il funzionamento della libreria. Per l'ultimo test il riconoscitore è stato eseguito sull'intero dataset di 4800 gesture. Dopo ogni singola esecuzione del riconoscitore sulla singola gesture, è stato inserito in un file il risultato, lo score, il tempo di esecuzione e il path della gesture in esame in quel momento. Su 4800 gesture il riconoscitore riconosce correttamente 4453 gesture, una porzione dei risultati, con confronto fra le due versioni Java e JavaScript, è mostrata in figura 4.7.

PolyRec.js (JavaScript version) results

Success: 4453
Failed: 347
Average execution time: 0.0006980024791666689 seconds / 0.698 ms

Result: arrow, score: 89.3, time: 0.0093741s +++ path: s02\fast\arrow01.xml
Result: arrow, score: 92.09, time: 0.0096914s +++ path: s02\fast\arrow02.xml
Result: caret, score: 80.01, time: 0.0067671s +++ path: s02\fast\arrow03.xml
Result: arrow, score: 92.33, time: 0.0048302s +++ path: s02\fast\arrow04.xml
Result: arrow, score: 92.15, time: 0.007719s +++ path: s02\fast\arrow05.xml
Result: arrow, score: 93.32, time: 0.0030999s +++ path: s02\fast\arrow06.xml
Result: arrow, score: 92.33, time: 0.0018007s +++ path: s02\fast\arrow07.xml
Result: arrow, score: 91.84, time: 0.0046963s +++ path: s02\fast\arrow08.xml
Result: arrow, score: 93.51, time: 0.002306s +++ path: s02\fast\arrow09.xml
Result: arrow, score: 92.76, time: 0.0016388s +++ path: s02\fast\arrow10.xml
Result: caret, score: 92.93, time: 0.0014648s +++ path: s02\fast\caret01.xml
Result: caret, score: 95.66, time: 0.001046s +++ path: s02\fast\caret02.xml
Result: caret, score: 95.75, time: 0.0007924s +++ path: s02\fast\caret03.xml
Result: caret, score: 96.45, time: 0.0006941s +++ path: s02\fast\caret04.xml
Result: caret, score: 96.05, time: 0.0007142s +++ path: s02\fast\caret05.xml
Result: caret, score: 94.16, time: 0.0010666s +++ path: s02\fast\caret06.xml
Result: caret, score: 96.46, time: 0.0009107s +++ path: s02\fast\caret07.xml
Result: caret, score: 94.74, time: 0.0021877s +++ path: s02\fast\caret08.xml
Result: caret, score: 94.53, time: 0.0021719s +++ path: s02\fast\caret09.xml
Result: caret, score: 95.25, time: 0.0019696s +++ path: s02\fast\caret10.xml

PolyRec (Java version) results

Success: 4453
Failed: 347
Average execution time: 0.0021737784166666664 seconds / 2.173 ms

Result: arrow, score: 89.3, time: 0.0198871s +++ path: s02\fast\arrow01.xml
Result: arrow, score: 92.09, time: 0.0041626s +++ path: s02\fast\arrow02.xml
Result: caret, score: 80.01, time: 0.0029302s +++ path: s02\fast\arrow03.xml
Result: arrow, score: 92.33, time: 0.0028497s +++ path: s02\fast\arrow04.xml
Result: arrow, score: 92.15, time: 0.0035442s +++ path: s02\fast\arrow05.xml
Result: arrow, score: 93.32, time: 0.0051552s +++ path: s02\fast\arrow06.xml
Result: arrow, score: 92.33, time: 0.0022852s +++ path: s02\fast\arrow07.xml
Result: arrow, score: 91.84, time: 0.0023378s +++ path: s02\fast\arrow08.xml
Result: arrow, score: 93.51, time: 0.0012953s +++ path: s02\fast\arrow09.xml
Result: arrow, score: 92.76, time: 0.0017855s +++ path: s02\fast\arrow10.xml
Result: caret, score: 92.93, time: 0.0018469s +++ path: s02\fast\caret01.xml
Result: caret, score: 95.66, time: 0.0011538s +++ path: s02\fast\caret02.xml
Result: caret, score: 95.75, time: 0.001128s +++ path: s02\fast\caret03.xml
Result: caret, score: 96.45, time: 0.0012524s +++ path: s02\fast\caret04.xml
Result: caret, score: 96.05, time: 9.781E-4s +++ path: s02\fast\caret05.xml
Result: caret, score: 94.16, time: 0.0015646s +++ path: s02\fast\caret06.xml
Result: caret, score: 96.46, time: 0.0015069s +++ path: s02\fast\caret07.xml
Result: caret, score: 94.74, time: 0.0014234s +++ path: s02\fast\caret08.xml
Result: caret, score: 94.53, time: 0.001251s +++ path: s02\fast\caret09.xml
Result: caret, score: 95.25, time: 0.0010231s +++ path: s02\fast\caret10.xml

Figura 4.7: Confronto testing versioni JavaScript(sinistra) e Java(destra)

Un ulteriore test è stato eseguito per confrontare i tempi di esecuzione delle due versioni, JavaScript e Java. Il riconoscitore è stato rieseguito sull'intero dataset di 4800 gesture, per ogni singolo riconoscimento è stato memorizzato il tempo di esecuzione in secondi. Nei risultati in figura 4.7 sono annotati, per ogni gesture di input, i tempi di riconoscimento. Il risultato finale (*average execution time*) è una media dei tempi su tutti i gesti. Nella versione JavaScript la media è un tempo medio pari a 0.698 millisecondi, migliore rispetto alla versione Java che ha totalizzato un tempo medio di 2.173 millisecondi.

CAPITOLO 5

POLYCODING

Dopo aver illustrato come la libreria PolyRec.js è stata progettata e sviluppata, nel seguente capitolo viene messo in pratica l'utilizzo della stessa con una demo. Per mostrare le potenzialità della libreria si presenta *"PolyCoding"*, una demo che offre come funzionalità principale la possibilità di scrivere codice Java con l'ausilio di gesture. Di seguito, in dettaglio, le varie fasi del progetto.

5.1 Progettazione

La demo *PolyCoding* nasce per sfruttare il rapido uso delle gesture e semplificare lunghi processi come, nel caso in esame, scrivere lunghe porzioni di codice. Si associa ad ogni costrutto di un linguaggio di programmazione una gesture. L'idea è semplice: ogni qual volta l'utente disegna una gesture, il costrutto relativo verrà inserito nel codice nel punto desiderato.

La scelta del linguaggio utilizzato è caduta su Java: linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica, che si appoggia sull'omonima piattaforma software di esecuzione, progettato per essere indipendente dalla piattaforma hardware di esecuzione.

Un esempio pratico è quello di poter aggiungere la dichiarazione di una nuova Classe Java con annesso costruttore, semplicemente con un gesto del puntatore. Gli obiettivi preposti durante la progettazione della demo sono vari: in primo velocizzare il processo di scrittura del codice, lasciando al programmatore il minor numero di cambiamenti da apportare; in secondo aiutare l'utente ad utilizzare la corretta sintassi del linguaggio, inserendo in automatico le parole chiave nei punti giusti; in terzo permettere una più facile memorizzazione del linguaggio, grazie all'associazione simbolica tra gesture e costrutto logico.

5.1.1 Requisiti funzionali

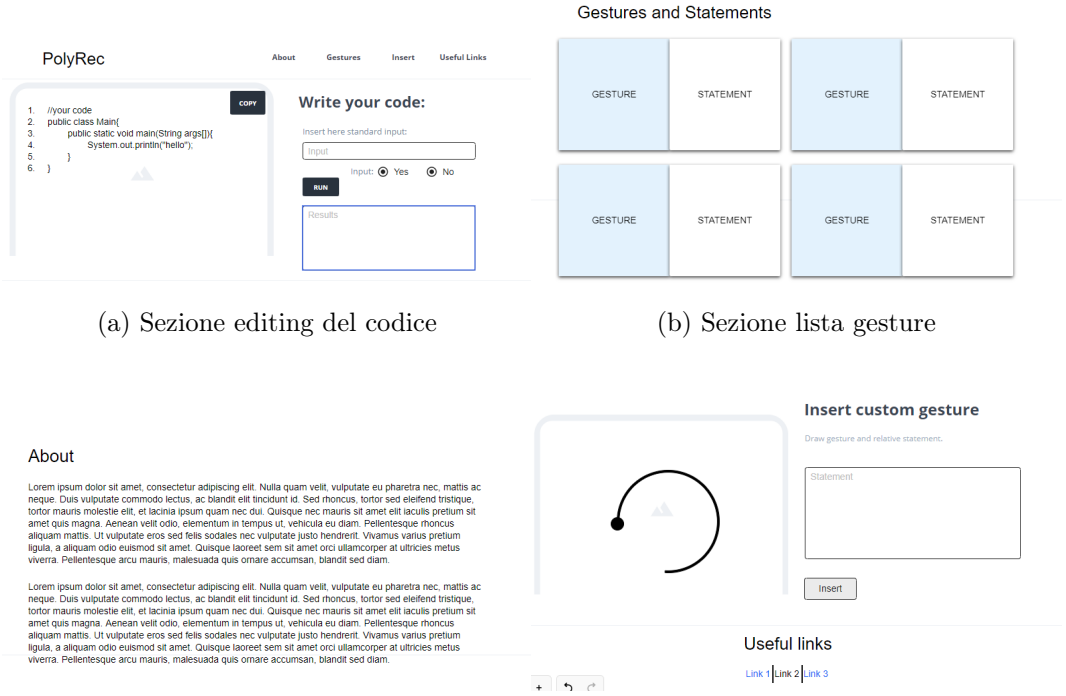
Di seguito i requisiti riguardanti le funzionalità che durante la fase di progettazione sono state scelte per essere implementate nella demo.

- Un editor di testo deve permettere di inserire codice Java.
- Un compilatore deve permettere di compilare ed eseguire codice Java per poterne testare la correttezza.
- Deve essere possibile tracciare gesture sull'editor di testo.
- Deve essere possibile riconoscere la gesture tracciata tramite la libreria PolyRec.js.
- La demo deve, in base alla gesture tracciata, inserire il corretto costrutto di programmazione Java.
- La demo deve permettere di poter copiare il codice scritto durante la sessione.
- La demo deve permettere di inserire nuove associazioni gesture - costrutto definite dall'utente.

5.1.2 Modello

Data la piccola grandezza di *PolyCoding*, si è preferito utilizzare un modello di sviluppo agile, che offrisse un approccio meno strutturato e focalizzato sul finire in tempi brevi il prodotto software. Il modello pensato e seguito è quello client-server. In generale è un modello costituito da un insieme di processi in esecuzione su diversi host: i processi che gestiscono una o più risorse sono detti server mentre quelli che richiedono l'accesso ad alcune di queste risorse distribuite sono detti client.

L'idea è di gestire le richieste HTTP tramite AJAX, basandosi su uno scambio di dati in background fra web browser e server, permettendo di aggiornare dinamicamente la pagina senza esplicito ricaricamento da parte dell'utente. Il lato front-end della demo è incaricato del tracciamento e della raccolta dei punti delle gesture disegnate su schermo e si occupa di mostrare all'utente i risultati delle operazioni che verranno svolte. Il riconoscimento delle gesture disegnate viene svolto anch'esso lato client, la libreria PolyRec.js si occupa di processare le coordinate ricevute in input e di restituire il risultato in tempi brevi.
Dalla progettazione front-end dell'applicazione sono nati i mockup in figura 5.1 che mostrano la suddivisione delle sezioni presenti nella demo. La parte server si occuperà invece di ricevere le richieste HTTP contenente il codice utente, di compilarlo ed eseguirlo per poi restituire al client l'output risultante.



(a) Sezione editing del codice

(b) Sezione lista gesture

(c) Sezione informazioni

(d) Sezione inserimento nuova gesture

Figura 5.1: I mockup relativi alla demo

5.1.3 Tecnologie utilizzate

Panoramica generale sulle tecnologie utilizzate per lo sviluppo della demo*PolyCoding*. Viene fatto uso di diverse tecnologie quali:

JavaScript

JavaScript, abbreviato con JS, è un linguaggio di programmazione interpretato ad alto livello. Si rimanda al capitolo precedente per una descrizione in dettaglio.

HTML/CSS

HTML è un linguaggio di markup nato per la formattazione e impaginazione di documenti ipertestuali; CSS è un linguaggio usato per definire la formattazione di documenti separando la stessa dal contenuto effettivo.

Node.js

Node.js è una tecnologia basata su JavaScript che permette di eseguire delle funzioni sul server. Come runtime JavaScript guidato da eventi asincroni, Node.js è progettato per creare applicazioni di rete scalabili.

NPM

NPM è uno strumento per la gestione dei moduli in Node.js, è in grado di scaricare e installare in completa autonomia una serie di pacchetti per aiutare lo sviluppo di applicazioni. Viene utilizzato via terminale e sono presenti vari comandi con varie opzioni preimpostate.

Express

Express è un framework per applicazioni WEB flessibile e leggero che fornisce una serie di funzioni per gestire chiamate HTTP e funzioni middleware.

Compile-run

Compile-run è un modulo di Node.js per la compilazione di codice in vari linguaggi: C, Cpp, Java, JavaScript, Python. Funziona da wrapper sui compilatori installati sul sistema. Fornisce delle API per eseguire programmi, generando un processo figlio per ogni esecuzione.

JQuery

JQuery è una libreria JavaScript per applicazioni web. Nasce con l'obiettivo di semplificare la selezione, la manipolazione degli eventi e l'animazione di elementi DOM in pagine HTML. Le sue caratteristiche permettono agli sviluppatori JavaScript di astrarre le interazioni a basso livello tra interazione e animazione dei contenuti delle pagine.

Bootstrap

Bootstrap è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, pulsanti e navigazione, così come alcune estensioni opzionali di JavaScript.

AJAX

AJAX è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive, basandosi su uno scambio di dati in background fra web browser e server, consentendo così l'aggiornamento dinamico di una pagina web avviene senza esplicito ricaricamento da parte dell'utente.

ACE

ACE è un editor di codice integrabile scritto in JavaScript. Corrisponde alle caratteristiche e alle prestazioni di editor nativi come Sublime, Vim e TextMate. Può essere facilmente incorporato in qualsiasi pagina Web e applicazione JavaScript.

5.2 Sviluppo

Nella pagina principale della demo è stato incluso uno script che integra un editor di testo ACE, si tratta di un editor inseribile nei propri progetti software via CDN. Una CDN o rete per la distribuzione dei contenuti è un gruppo di server distribuiti in più aree geografiche che velocizza la delivery dei contenuti web avvicinandoli di più alle posizioni geografiche degli utenti. Permette di integrare librerie come JQuery, funzionalità e come nel caso di ACE, un editor di testo altamente personalizzabile.

L'editor permette di stabilire a priori il linguaggio, si occupa di formattare il codice scritto e di identificare con una diversa colorazione le parole chiave. In figura 5.2 viene mostrata la sezione della demo dedicata

all'editor.

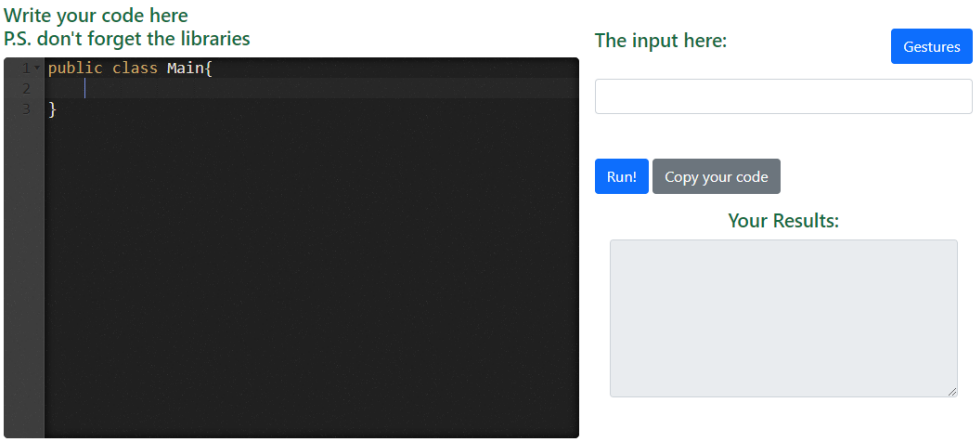


Figura 5.2: Sezione editor nella demo

Al caricamento della pagina vengono settati i parametri necessari a poter tracciare la gesture sull'area voluta:

- Un array JavaScript memorizza le coordinate X, Y, dei punti della gesture disegnata.
- Viene creata un'istanza *PolyRecognizer* del riconoscitore che carica i template di default.
- Viene calcolato il bounding box dell'area di disegno, un rettangolo per cui vengono memorizzate le coordinate del punto in alto a sinistra, l'altezza e la grandezza.

Nell'editor è possibile inserire codice tramite tastiera o, principale obiettivo, tramite delle gesture. La gesture può essere disegnata sull'editor e tracciata grazie agli eventi del Document Object Model applicati. Tutti e tre i metodi hanno come parametro di input le coordinate X, Y del puntatore utente. Le coordinate si trattano di due proprietà di sola lettura MouseEvent.clientX e MouseEvent.clientY dell'interfaccia MouseEvent che forniscono la coordinata orizzontale e verticale all'interno della finestra dell'applicazione in cui si è verificato l'evento. I tre eventi considerati:

- *onmousedown*: evento generato quando l'utente clicca per la prima volta sull'editor, una funzione listener si occupa di raccogliere le coordinate del pixel toccato, sottraendo le coordinate della bounding box a quelle passate come input. Il primo punto viene quindi aggiunto all'array di punti.

- *onmousemove*: evento generato quando il puntatore utente si muove nell'area interessata. La funzione listener associata memorizza nell'array tutti i punti che vengono tracciati durante il movimento del puntatore.

- *onmouseup*: evento generato quando viene rilasciato il puntatore dall'area. La funzione listener controlla se la gesture disegnata contiene almeno 10 punti, in caso affermativo esegue il riconoscitore sulla gesture con il metodo *Recognize(points)*.

Dopo aver eseguito il riconoscitore sulla lista di punti della gesture, una funzione si occupa di scegliere la porzione di codice Java da aggiungere. In input, la funzione riceve il risultato del riconoscitore e tramite un costrutto *Switch* viene scelto il codice opportuno da aggiungere nell'editor. In figura 5.3 un esempio di associazioni tra gesture e codice. *PolyCoding* permette inoltre, per una migliore comprensione del linguaggio, di compilare ed eseguire il codice Java utente tramite scambio di request e response HTTP con un server.

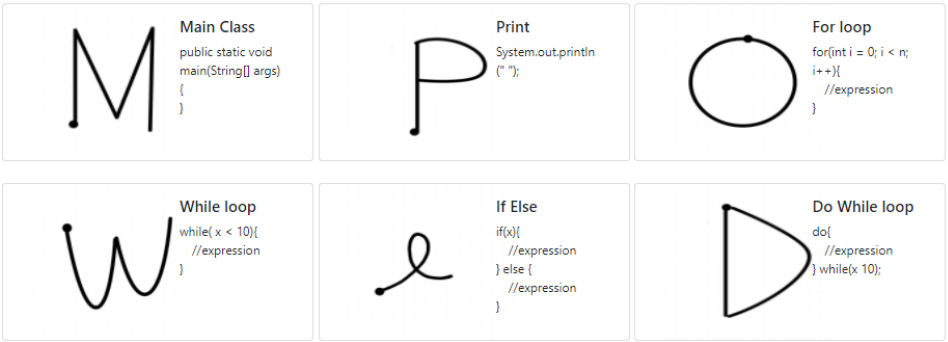
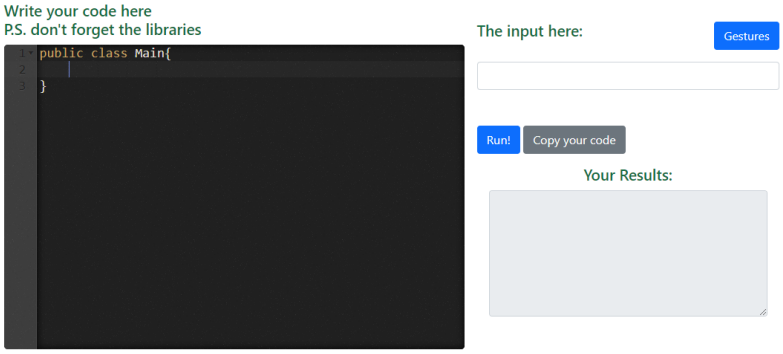


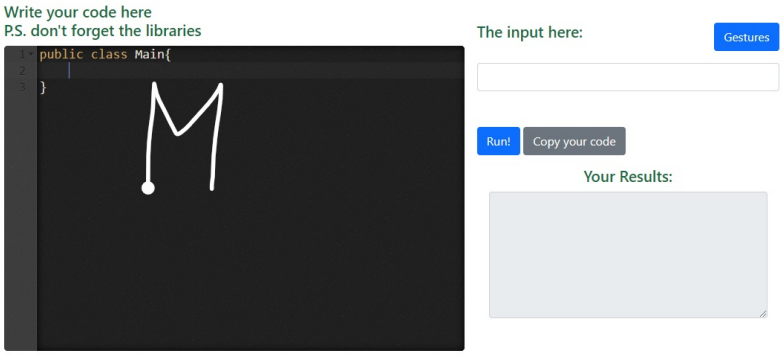
Figura 5.3: Associazione gesture - costrutto Java

In figura 5.4 una sequenza di immagini mostra la sequenza di azioni da svolgere per utilizzare la principale funzione sviluppata nella demo:

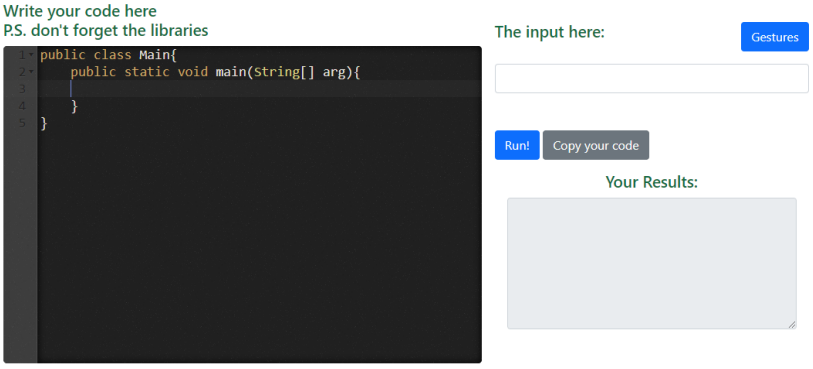
- Al caricamento della pagina l'editor mostra di default la dichiarazione di una classe *Main*(figura 5.4a).
- L'utente può tracciare una delle gesture pre-impostate, o una da lui definita per aggiungere il relativo costrutto. In figura 5.4b per una maggiore comprensione viene mostrata la gesture disegnata, nella demo ciò che viene tracciato è nascosto all'utente.
- Il costrutto viene inserito nell'editor nel punto in cui la gesture è stata disegnata(figura 5.4c).



(a) Sezione editor al caricamento



(b) Viene tracciata una gesture



(c) Il codice viene inserito nell'editor

Figura 5.4: Sequenza di utilizzo editor

Il server è stato sviluppato con Node.js ed il framework Express. Prima di lanciare l'applicazione viene attivato il server, per dare modo di ascoltare le request HTTP sulla porta 8080. In caso di request iniziale di tipo GET, la risorsa ritornata è la pagina principale, l'index. Un form nella pagina

principale permette, attraverso una request di tipo POST, di inviare al server informazioni quali:

- il codice utente da compilare ed eseguire in formato stringa.
- un possibile input utente in formato stringa.

Per fare in modo che l'utente non debba ricaricare la pagina per effettuare la request, la stessa avviene in background tramite AJAX. Il server processa le informazioni ricevute con il modulo *compile-run* ed invia nella response il risultato dell'esecuzione. Il risultato viene mostrato nella sezione dedicata senza che l'utente sia a conoscenza dell'organizzazione sottostante. È inoltre possibile controllare sul server i risultati di ogni esecuzione in quanto gli stessi vengono stampati sulla console Node.js.

Una ulteriore funzionalità disponibile in *PolyCoding* è quella di poter aggiungere nuove associazioni gesture - codice. Un elemento HTML Canvas permette di tracciare linee di qualsiasi forma nel campo dedicato. Un Canvas è una grande matrice di pixel, ognuno dei quali modificabile singolarmente nelle sue quattro componenti RGBA. La sezione della demo dedicata all'inserimento di nuove associazioni è mostrata in figura 5.5

Figura 5.5: Sezione inserimento gesture/costrutto

La gesture verrà disegnata sul Canvas in modo tale che l'utente possa vedere in tempo reale quello che viene disegnato. In un form laterale l'utente inserisce il nome della gesture e la porzione di codice da inserire ogni qual volta la gesture viene disegnata.
Le nuove gesture vengono memorizzate nel *Local Storage*, un oggetto che permette di salvare valori come coppie chiave/valore nel browser. I dati rimangono memorizzati anche in seguito al ricaricamento della pagina e a un riavvio del browser. Nel momento in cui il risultato del riconoscitore deve essere confrontato, viene ripreso l'elemento corrispondente dal Local Storage, confrontato ed in caso di match inserito nell'editor la relativa porzione di codice.

CAPITOLO 6

CONCLUSIONI

In questo lavoro di tesi è stata presentata la libreria JavaScript PolyRec.js per il riconoscimento di gesture mono-tratto a due dimensioni. La libreria permette di caricare nuove gesture utente, rimuoverle e principalmente applicare il riconoscitore. La libreria è di piccole dimensioni (23 KB per la verione minimizzata e 36 KB per la versione non minimizzata), portabile e facilmente eseguibile su tutti i maggiori browser web. È stata inoltre presentata una demo denominata *PolyCoding* che offre la possibilità di scrivere codice Java attraverso l'uso di gesture, cercando di rendere il processo di scrittura del codice più veloce ed efficiente. Come libreria JavaScript, PolyRec.js diviene disponibile per una miriade di scopi e contesti diversi.

Sviluppi futuri di PolyRec [2] e di PolyRec.js prevedono un'ottimizzazione delle prestazioni atta a rendere il riconoscitore maggiormente competitivo con le alternative esistenti nonostante le già ottime performance. Si potrebbe, inoltre, realizzare un porting su diversi linguaggi di programmazione per estendere le possibilità d'uso o renderla disponibile per la creazione di applicazione desktop. Tra le migliorie apportabili non manca l'estensione del riconoscimento verso gesture multi-tratto che prevedono simboli più complessi e maggiori applicazioni.

BIBLIOGRAFIA

[1] Frederic Kaplan. «Are Gesture-Based Interfaces the Future of Human Computer Interaction?» In: *Proceedings of the 2009 International Conference on Multimodal Interfaces*. ICMI-MLMI '09. Cambridge, Massachusetts, USA: Association for Computing Machinery, 2009, pp. 239–240. ISBN: 9781605587721. DOI: 10.1145/1647314.1647365. URL: https://doi.org/10.1145/1647314.1647365.

[2] Vittorio Fuccella e Gennaro Costagliola. «Unistroke Gesture Recognition Through Polyline Approximation and Alignment». In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2015, pp. 3351–3354. ISBN: 9781450331456. URL: https://doi.org/10.1145/2702123.2702505.

[3] Vittorio Fuccella e Gennaro Costagliola. *PolyRec GitHub Repository Java VersionUnistroke Gesture Recognition Through Polyline Approximation and Alignment*. https://github.com/cluelab/polyrec. 2015.

[4] *Microsoft Kinect*. https://developer.microsoft.com/it-it/windows/kinect/.

[5] *Leap Motion Controller*. https://www.ultraleap.com/product/leap-motion-controller/.

[6] *GestureRecognition Wikipedia*. https://it.wikipedia.org/wiki/Gesture_recognition.

[7] C. C. Tappert. «Cursive Script Recognition by Elastic Matching». In: *IBM Journal of Research and Development* 26.6 (1982), pp. 765–771. DOI: 10.1147/rd.266.0765.

[8] Dean Rubine. «Specifying Gestures by Example». In: *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '91. New York, NY, USA: Association for Computing Machinery, 1991, pp. 329–337. ISBN: 0897914368. DOI: 10.1145/122718.122753. URL: https://doi.org/10.1145/122718.122753.

[9] Jacob O. Wobbrock, Andrew D. Wilson e Yang Li. «Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes». In: *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*. UIST '07. Newport, Rhode Island, USA: Association for Computing Machinery, 2007, pp. 159–168. ISBN: 9781595936790. DOI: 10.1145/1294211.1294238. URL: https://doi.org/10.1145/1294211.1294238.

[10] Lisa Anthony e Jacob Wobbrock. «$N-protractor: a fast and accurate multistroke recognizer». In: mag. 2012, pp. 117–120. ISBN: 9781450314206.

[11] Radu-Daniel Vatavu, Lisa Anthony e Jacob O. Wobbrock. «Gestures as Point Clouds: A $P Recognizer for User Interface Prototypes». In: *Proceedings of the 14th ACM International Conference on Multimodal Interaction*. ICMI '12. Santa Monica, California, USA: Association for Computing Machinery, 2012, pp. 273–280. ISBN: 9781450314671. DOI: 10.1145/2388676.2388732. URL: https://doi.org/10.1145/2388676.2388732.

[12] Radu-Daniel Vatavu, Lisa Anthony e Jacob O Wobbrock. «$Q: a super-quick, articulation-invariant stroke-gesture recognizer for

low-resource devices». In: *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*. 2018, pp. 1–12.

[13] Radu-Daniel Vatavu. «Improving gesture recognition accuracy on touch screens for users with low vision». In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 2017, pp. 4667–4679.

[14] J Reaver, Thomas F Stahovich e James Herold. «How to make a Quick$ using hierarchical clustering to improve the efficiency of the Dollar Recognizer». In: *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*. 2011, pp. 103–108.

[15] James Herold e Thomas F Stahovich. «The 1¢ recognizer: A fast, accurate, and easy-to-implement handwritten gesture recognition technique». In: *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*. 2012, pp. 39–46.

[16] Eugene M Taranta II et al. «Jackknife: A reliable recognizer with few samples and many modalities». In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 2017, pp. 5850–5861.

[17] Jean Vanderdonckt, Paolo Roselli e Jorge Luis Pérez-Medina. «!!FTL, an Articulation-Invariant Stroke Gesture Recognizer with Controllable Position, Scale, and Rotation Invariances». In: *Proceedings of the 20th ACM International Conference on Multimodal Interaction*. ICMI '18. Boulder, CO, USA: Association for Computing Machinery, 2018, pp. 125–134. ISBN: 9781450356923. DOI: 10.1145/3242969.3243032. URL: https://doi.org/10.1145/3242969.3243032.

[18] Yang Li. «Protractor: A Fast and Accurate Gesture Recognizer». In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery,

2010, pp. 2169–2172. ISBN: 9781605589299. URL: https://doi.org/10.1145/1753326.1753654.

[19] Allan Christian Long Jr. *Quill: a gesture design tool for pen-based user interfaces*. University of California, Berkeley, 2001.

[20] Nathan Magrofuoco et al. «GestMan: A Cloud-Based Tool for Stroke-Gesture Datasets». In: *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. EICS '19. Valencia, Spain: Association for Computing Machinery, 2019. ISBN: 9781450367455. DOI: 10.1145/3319499.3328227. URL: https://doi.org/10.1145/3319499.3328227.

[21] Roberto Bufano et al. «PolyRec Gesture Design Tool: A tool for fast prototyping of gesture-based mobile applications». In: *Software: Practice and Experience* n/a.n/a (). DOI: https://doi.org/10.1002/spe.3024. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3024. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3024.

[22] DAVID H DOUGLAS e THOMAS K PEUCKER. «ALGORITHMS FOR THE REDUCTION OF THE NUMBER OF POINTS REQUIRED TO REPRESENT A DIGITIZED LINE OR ITS CARICATURE». In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10.2 (1973), pp. 112–122. DOI: 10.3138/FM57-6770-U75U-7727. eprint: https://doi.org/10.3138/FM57-6770-U75U-7727. URL: https://doi.org/10.3138/FM57-6770-U75U-7727.

[23] Saul B. Needleman e Christian D. Wunsch. «A general method applicable to the search for similarities in the amino acid sequence of two proteins». English (US). In: *Journal of Molecular Biology* 48.3 (mar. 1970). Funding Information: This work was supported in part by grants to one of us (S.B.N.) from the U.S. Public Health Service (1 501 FR 05370 02) and from Merck Sharp and Dohme., pp. 443–453. ISSN: 0022-2836. DOI: 10.1016/0022-2836(70)90057-4.

[24] Yiyan Xiong e Joseph J. LaViola Jr. «A ShortStraw-based algorithm for corner finding in sketch-based interfaces». In: *Computers and Graphics* 34.5 (2010). CAD/GRAPHICS 2009 Extended papers from the 2009 Sketch-Based Interfaces and Modeling Conference Vision, Modeling and Visualization, pp. 513–527. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2010.06.008. URL: https://www.sciencedirect.com/science/article/pii/S0097849310001044.

[25] *ResearchGate The-illustration-of-the-indicative-angle-th-of-G4*. https://www.researchgate.net/figure/The-illustration-of-the-indicative-angle-th-of-G4_fig3_282512284.