Università degli studi di Trento

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in Informatica

Tesi Finale

LOCALIZZAZIONE DI UN ROBOT SU PERCORSI NOTI UTILIZZANDO UNA WEBCAM IN POSIZIONE FRONTALE

Relatore:
Prof. Luigi Palopoli

Candidato:
Giulio Fornasaro

Correlatore:
Federico Moro

Anno accademico 2013-2014

Ai miei genitori

Indice

Elenco delle figure

Capitolo 1

Introduzione

1.1 Il progetto

Lo scopo del progetto è quello di ideare un robot in grado di muoversi lungo un percorso a terra contrassegnato da una linea nera. Il robot conosce a priori la struttura del percorso tuttavia deve riuscire a localizzarsi su di esso. Per sapere dov'è localizzato utilizza degli encoder posti sulle ruote, mantenendo un contatore dei giri compiuti per ciascuna di esse. In più si avvale di due telecamere: una laterale collegata a una scheda Beaglebone[1] e puntata verso il terreno, permette al robot di identificare la linea e mantenerla al fianco e una frontale, collegata a una scheda IGEP[2], che analizza le immagini del percorso, allo scopo di ottenere informazioni rilevanti alla localizzazione del robot all'interno di esso.

Grazie agli encoder si sa esattamente quanti giri ha fatto ciascuna ruota e si riesce a calcolare facilmente la posizione del dispositivo. Gli encoder tuttavia accumulano dei piccoli errori che tendono ad accumularsi nel tempo. Questo provoca errori di calcolo i quali portano ad avere una stima della posizione non affidabile a medio e lungo termine.

Di conseguenza è stato deciso di inserire dei metodi di controllo incrociato della posizione, tra cui l'inserimento nel percorso di cartelli in posizioni note [1] e un algoritmo che permetterà di riconoscere, utilizzando la telecamera frontale, il tipo di curva, permettendo un confronto con i dati memorizzati, riguardanti la struttura del percorso. Questo consentirà al robot di avere un riscontro sui dati degli encoder e di correggere eventualmente i valori di posizione.

[1] `http://beagleboard.org/bone`

[2] `https://www.isee.biz/products/igep-processor-boards`

1.2 Motivazioni

Dalla nascita dell'informatica e della robotica si è sempre sognato di creare robot, macchine e veicoli in grado di interagire autonomamente con l'ambiente circostante. Un requisito fondamentale per raggiungere questo obiettivo è la visione artificiale (o computer vision) la quale è caratterizzata da processi atti a riprodurre la visione che gli umani hanno della realtà e dello spazio circostante.

Lo sviluppo recente in questo campo ha permesso il raggiungimento di obiettivi importanti nell'ambito automobilistico, permettendo ad esempio di evitare pericoli o riconoscere automaticamente segnali stradali. L'evoluzione tecnologica in questo campo sta ponendo le basi per la creazione di sistemi più complessi. Grazie all'unione tra visione artificiale e sensoristica di controllo nelle automobili, sarà possibile ideare sistemi che porteranno un giorno al completo controllo della viabilità stradale.

1.3 Obiettivi

In questo progetto si affronterà il problema di implementazione dell'algoritmo di riconoscimento della tipologia di curva, attraverso le immagini prese dalla telecamera frontale. Utilizzando come base un algoritmo di identificazione di una linea su un immagine, si propone un modo per identificare il comportamento del percorso, elaborando i dati forniti da esso.

L'algoritmo di pathshape ricerca nell'immagine una linea compatibile a dei parametri impostati e restituisce una matrice contenente le coordinate x e y dei punti all'interno della linea e un angolo di inclinazione del percorso in quella particolare posizione. L'algoritmo di riconoscimento della curva, usando i dati ottenuti dall'algoritmo di pathshape, identifica delle eventuali curve. Se in quest'ultimo trova effettivamente una o più curve nell'area studiata ne calcola la distanza e se possibile il raggio, l'angolo e il centro di curvatura.

Ovviamente lavorando su un sistema embedded è necessario prestare attenzione alle prestazioni, a maggior ragione se consideriamo che l'algoritmo verrà eseguito su un oggetto in movimento.

Inoltre ci si aspetta che l'algoritmo riesca a riconoscere correttamente curve con raggio e angolo di curvatura differenti mantenendo un errore limitato, in modo da avere una affidabilità maggiore nella fase di matching.

1.4 Outline

Dopo questa breve introduzione, la tesi si struttura su quattro capitoli.

- Capitolo 2: offre una descrizione di come opera l'algoritmo denominato pathshape.
- Capitolo 3: viene mostrata nel dettaglio la struttura dell'algoritmo di identificazione della curva.
- Capitolo 4: vengono esposti e analizzati i dati ottenuti dalle prove sperimentali svolte sull'algoritmo di identificazione della curva, confrontandoli con i dati raccolti sull'algoritmo di pathshape.
- Capitolo 5: conclusioni del lavoro svolto.

Capitolo 2

L'identificazione della linea

Esistono molte applicazioni nell'ambito della robotica che richiedono al robot la capacità di muoversi in modo autonomo. Questo obiettivo risulta notevolmente semplificato con l'aggiunta di un marcatore nell'ambiente, in grado di suggerire un possibile percorso o delimitare una corsia. É quindi possibile far muovere il robot facendo si che mantenga una distanza fissa dal marcatore, permettendogli di seguire il percorso. Tuttavia se il percorso inizia a incurvarsi, mantenere il robot a distanza fissa dal marcatore potrebbe non essere la scelta migliore.

Per questo motivo è stato sviluppato un algoritmo, denominato *pathshape* [2]. Questo algoritmo è in grado di identificare un marcatore, nello specifico una linea nera di spessore noto, analizzando immagini raccolte da una webcam. Questo permette al robot di ottenere informazioni utili sul percorso che ha di fronte a se.

2.1 Analisi dell' algoritmo di pathshape

L' algoritmo di pathshape ha la necessità di elaborare le immagini provenienti da uno stream video e analizzarle in modo da identificare un possibile marcatore all'interno di esse. Essendo il codice eseguito su una scheda con processore non particolarmente potente (~1GhZ) e avendo a disposizione poca RAM(~1GB), le performance per questo algoritmo sono un aspetto fondamentale. Per questo motivo l'algoritmo è stato scritto in linguaggio C++ in virtù delle ottime prestazioni offerte.

2.2 L'algoritmo di pathshape

L'algoritmo di pathshape si basa essenzialmente su 4 fasi principali. In una prima fase la parte di piano visibile nell'immagine catturata dalla telecamera frontale viene processata attraverso l'algoritmo di Canny [3], in grado di rimuovere la maggior parte degli elementi visibili nell'immagine, mantenendo solamente i contorni.

A questo punto la nuova immagine contenente solo i contorni degli oggetti viene proiettata su un piano orizzontale, in modo da rendere più efficente questa fase. É quindi definito un set di posizioni V e viene creata una telecamera virtuale che inizia a scorrere l'immagine proiettata posizionandosi nelle posizioni definite nel set V. Su ognuna delle immagini generate per la telecamera virtuale utilizza l'algoritmo RANSAC [4] (RANdom SAmple and Consensus) per trovare due linee parallele a distanza nota tra loro (rapprentanti i margini della linea utilizzata come marcatore). L'algoritmo presi due punti casuali all'interno del set ne calcola la retta passante per entrambi. Quindi determina il sottoinsieme $S1$ come il numero dei punti passanti vicino alla retta appena calcolata entro un certo errore. Se il numero di punti trovati supera una certa soglia $|S1| > n$ l'algoritmo crea un modello adeguato usando i punti dell'insieme $S1$. Questo processo viene ripetuto più volte per aumentare le chance di ottenere un risultato corretto essendo RANSAC un algoritmo non deterministico.

Dopo che due soluzioni soddisfacenti sono state trovate le soluzioni vengono confrontate tra di solo per verificare che siano effettivamente parallele e alla distanza prevista. In caso positivo viene calcolato un punto tra le due soluzioni che verrà usato dall'algoritmo di pathshape come punto di partenza. In questo posizione di partenza idoneo, dove le due linee rappresentanti i contorni della linea a nera a terra vengono identificate correttamente e alla giusta distanza, l'algoritmo comincia a scorrere la linea trovata effettuando piccoli passi di entità fissa [1] in direzione della linea. Nel caso in cui la linea abbia un andamento curvilineo risulta più complesso in quanto in questo caso l'algoritmo RANSAC deve comunque adattare una retta alla serie di punti. Essendo questi non perfettamente in linea potrebbero non rientrare nei limiti di tolleranza imposti. Questo rende necessario avvicinare la telecamera virtuale al terreno cosi che le linee della curva appaiano più diritte, semplificando il processo di adattamento.

Raggiunta la fine della parte di linea visibile nell'immagine, o dopo comunque una certa distanza l'algoritmo RANSAC non riuscirà più a trovare correttamente le linee. A questo punto l'algoritmo di pathshape riporta la

[1] il valore di *pshapeStep* che definisce l'entità di questi passi è stato fissato a 0.04-0.06m

posizione della telecamera frontale al punto di partenza e inizia a svolgere lo stesso procedimento in direzione opposta. Questo per completare il set di punti che andranno poi a identificare la linea.

Capitolo 3

L'identificazione della curva

L'algoritmo di identificazione della curva è composto da diverse funzioni, ciascuna di esse svolge delle operazioni sui dati ottenuti dall'algoritmo del pathShape, che permettono di estrarre i dati necessari per eseguire il confronto tra una eventuale curva identificata nell'immagine (catturata dalla telecamera frontale) e la lista delle curve presenti nel percorso.

Una prima funzione si occupa di riordinare e normalizzare i dati. Si procede quindi a segnare i dati punto per punto, utilizzando come criterio il comportamento di ogni dato rispetto a quello successivo. Per finire si utilizzano i dati estratti sul comportamento per calcolare i parametri delle curve visibili nell'immagine, che verranno infine usati per il confronto.

3.1 L'aggiustamento dei dati dell'algoritmo di pathshape

Come analizzato nel precedente capitolo l'algoritmo di pathshape ritornare una matrice di vettori 3×N, nella quale ciascun vettore contiene le coordinate (x, y, α) di un punto all'interno della curva. α è equivalente all'angolo di curvatura della linea nel punto considerato dal pathShape.

Le prime due problematiche affrontate sono state:

- I punti ritornati dal pathshape non erano ordinati, quindi è stato necessario riordinarli prima di svolgere qualsiasi altra operazione. É infatti necessario che i dati siano in ordine per svolgere le operazioni successive con più efficenza.

- Tra i punti ritornati dal pathshape si riscontravano delle leggere fluttuazioni e si avevano spesso situazioni in cui un punto era leggermente

spostato verso un lato della linea, mentre il successivo risultava leggermente spostato verso il lato opposto. Questa problematica è stata riscontrata soprattutto nel caso di punti all'interno di una linea curva. Si è reso quindi necessario l'uso di una funzione che regolarizzasse leggermente i dati.

Il primo problema è stato risolto modificando l'algoritmo del pathshape. L'algoritmo infatti, dopo una fase di ricerca nella parte dell'immagine soprastante il primo punto identificato, cominciava una fase di ricerca nella parte sottostante. Questo portava ad avere una matrice in cui gli ultimi k vettori risultavano essere le posizioni più vicine al punto mantenuto a fianco del robot seguendo la linea. Sapendo questo è bastato modificare l'algoritmo in modo da salvarsi un valore che indicasse l'indice in cui questa inversione avveniva, in modo da riordinare subito dopo la matrice. É possibile infatti spostare i vettori apparsi dopo l'indice salvato all'inizio della matrice, spostando in avanti il resto dei dati con complessità O(n). Il raggiungimento dell'obiettivo con complessità lineare è importante per ridurre il tempo di calcolo necessario.

Per risolvere la seconda problematica è stato deciso di adottare un algoritmo che calcoli una media pesata dei punti intorno al punto considerato. Questo metodo consiste nel sostituire ogni punto $p_i = (x_i, y_i, \alpha_i)$ con il nuovo punto ottenuto $\overline{p_i} = (\overline{x_i}, \overline{y_i}, \overline{\alpha_i})$, riducendo significativamente le fluttuazioni all'interno della matrice. La funzione lavora quindi su ogni punto singolarmente facendo una media pesata del punto considerato con i 4 elementi più vicini. Ai valori più distanti viene assegnato un peso minore, in modo da influenzare il risultato in maniera minore e ottenere una matrice di punti con un andamento più dolce, senza che questi siano eccessivamente influenzati dai punti intorno. Quindi il nuovo punto $\overline{p_i}$ viene calcolato come

$$\overline{p_i} = \frac{w_1 \cdot (p_{i-2} + p_{i+2}) + w_2 \cdot (p_{i-1} + p_{i+1}) + p_i}{2 \cdot w_1 + 2 \cdot w_2 + 1}$$

con w_1 e w_2 come pesi assegnati e $w_1 < w_2$.

3.2 Lo studio dell'angolazione nei singoli punti della curva

Risolti i problemi riguardanti i dati del pathshape si rende necessario identificare quando il percorso sta curvando e quando sta procedendo in linea retta. È quindi necessario avere una misura della variazione dell'angolo tra un punto e il successivo. Inizialmente è stato necessario ricavare un vettore

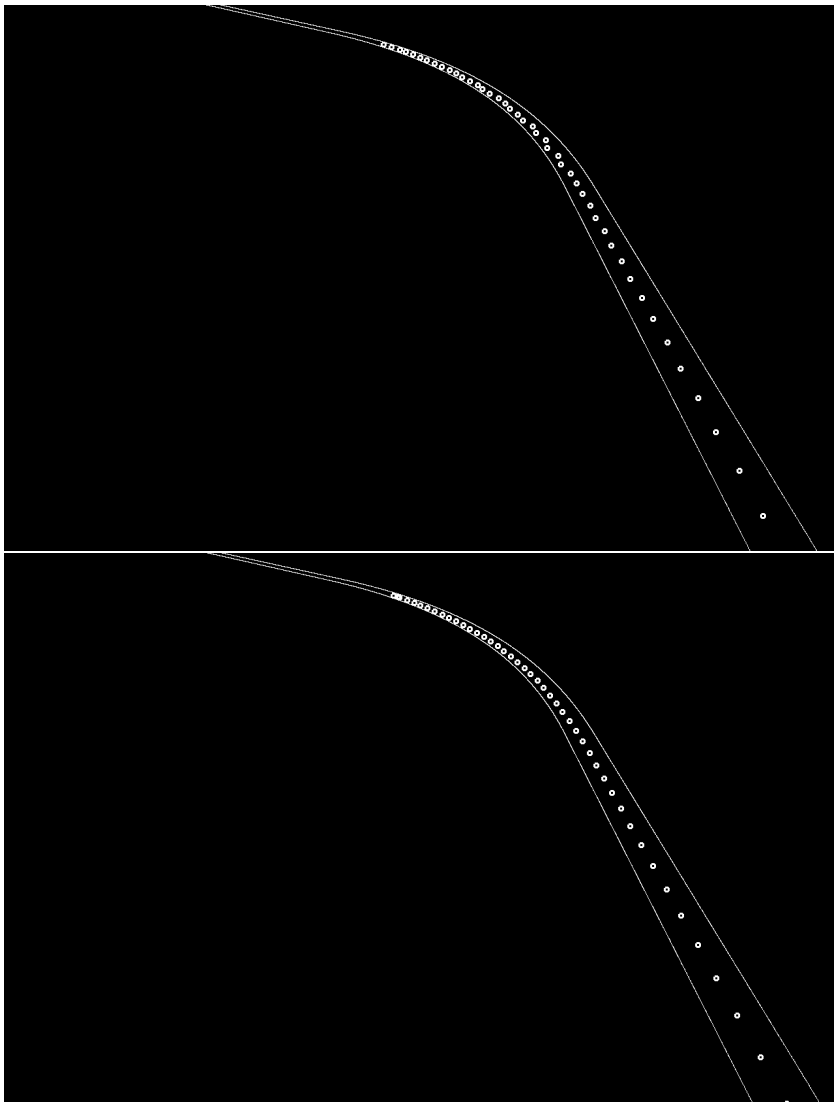

Figura 3.1: Esempio di come risultavano i punti restituiti dall'algoritmo del pathshape prima (sopra) e dopo (sotto) aver applicato la funzione per regolarizzarne i dati.

contenente tutte le differenze tra un punto e il successivo $\delta_i = \alpha_i - \alpha_{i+1}$. Da questo è stato sviluppato un semplice algoritmo che scorrendo i punti marca ciascuno di essi con un numero che indica il suo comportamento, ovvero lo spostamento verso destra, lo spostamento verso sinistra o lo spostamento in linea retta.

Questo risultato è stato ottenuto attraverso un algoritmo che linearmente scorre il vettore delle differenze e per ogni punto effettua un controllo sul

valore della differenza, sapendo che nel caso di spostamenti a destra δ_i mantiene un valore a segno positivo, viceversa a valori di δ_i positivi si assoceranno piccoli spostamenti verso sinistra. Si procede quindi a marcare il punto utilizzando un sistema di regole rigide basate su dei valori fissi W_1 e W_2, con $W_1 > W_2$ (da non confondere con i pesi w_1 e w_2 usati precedentemente). Questi due valori sono sono impostati al valore di $W_1 \mapsto 0.065 \cdot pshapeStep$ e $W_2 \mapsto 0.015 \cdot pshapeStep$, sono quindi dipendenti dalla lunghezza del passo impostato nell'algoritmo di pathshape. Questo perchè il valore di δ dipende direttamente dalla larghezza del passo compiuto quindi senza questo accorgimento sarebbe necessario normalizzare i valori di δ per il valore di *pshapeStep*.

L'algoritmo mantiene in memoria l'ultimo valore assegnato: $flag_{prec}$, e utilizza un contatore in modo da garantire una leggera resistenza nel passaggio da un caso in cui i punti siano posti in curva al caso in cui in punti successivi siano parte di un rettilineo. Questo accorgimento è stato adottato per evitare che degli errori nell'algoritmo di pathShape possano alterare il risultato. Andiamo quindi ad analizzare le regole con cui vengono marcati i singoli punti:

- Se il punto risulta angolato di un valore δ_i con $\delta_i > W_1$ o $\delta_i < -W_1$, viene immediatamente marcato come un punto in curva, a sinistra nel primo caso: $flag_i \mapsto -1$ o a destra nel secondo caso: $flag_i \mapsto 1$.

- Se il punto risulta angolato di un valore δ_i, con $W_1 > \delta_i > W_2$ o $-W_2 > \delta_i > -W_1$, si mantiene il valore del flag precedentemente usato, a meno che δ_i non indichi uno spostamento inverso rispetto a quello indicato da $flag_{prec}$: caso in cui $flag_{prec} = -1 \wedge \delta_i < 0$, caso in cui $flag_{prec} = 1 \wedge \delta_i > 0$. In questi casi si ritorna in una situazione di rettilineo $flag_i \mapsto 0$.

- Se il valore soddisfa invece il seguente criterio: $W_2 > \delta_i > -W_2$, allora l'algoritmo controlla il valore dal contatore c. Se $c > 2$, quindi se ci sono stati almeno altri 2 punti entro il range di valori considerato prima di quello corrente, allora imposta il valore del punto corrente marcandolo come parte di un rettilineo, altrimenti incrementa di 1 il contatore c. Questo meccanismo di fatto sospende l'assegnazione di un valore di $flag_i$ per alcuni punti, il che rende necessario controllare il valore di c ogni volta che si assegna un valore a $flag_i$ ed eventualmente assegnare lo stesso valore anche ai k punti lasciati in sospeso.

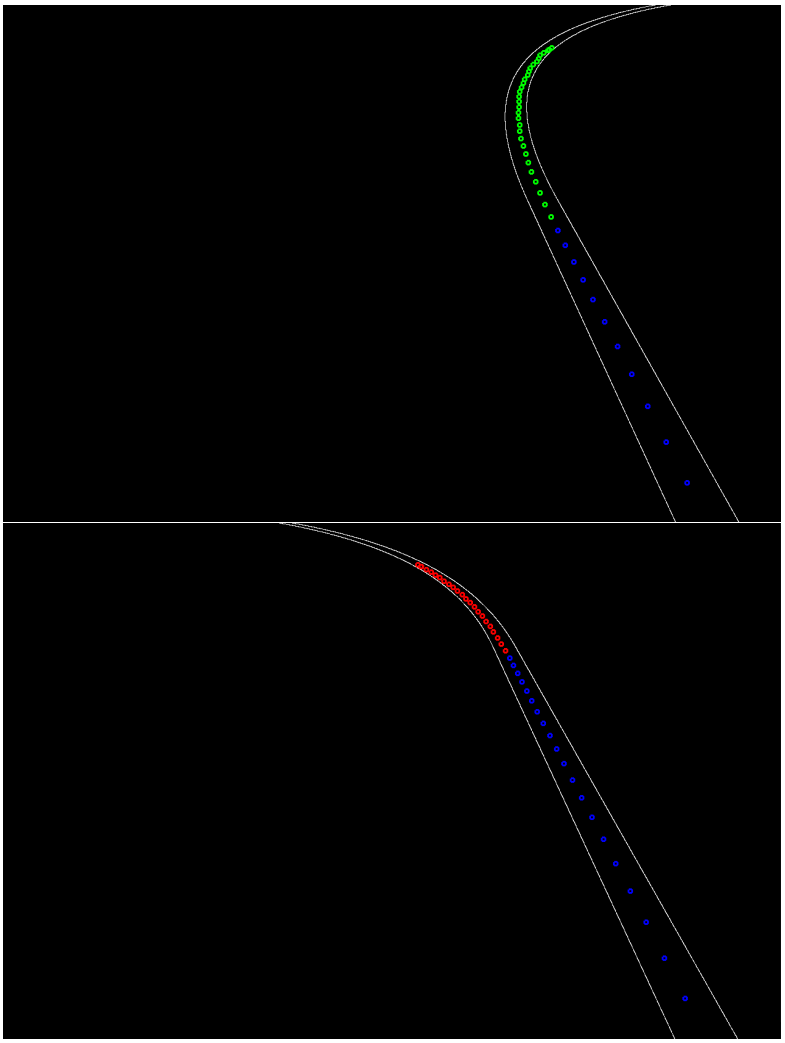

Figura 3.2: In queste due immagini possiamo osservare un paio di risultati dell'algoritmo di studio della curva: blu nel caso il punto sia in rettilineo, rosso se viene identificata una curvatura verso sinistra, verde se invece il percorso sta curvando verso destra

3.3 La procedura di calcolo dei parametri della curva

Una volta che ciascuno dei punti è stato marcato il risultato viene passato alla funzione *findCurvature*. La funzione in una prima fase separa le curve identificate nell'immagine, quindi per ognuna di esse svolge una serie di calcoli necessari a trovare i parametri necessari. In primo luogo applica una decomposizione a valori singolari tra una prima matrice N×3, con N numero

di punti nella curva e costruita nel seguente modo:

$$\begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix}$$

e un secondo vettore di dimensione N costruito nel seguente modo:

$$\begin{bmatrix} -(x_0^2 + y_0^2) \\ -(x_1^2 + y_1^2) \\ \vdots \\ -(x_n^2 + y_n^2) \end{bmatrix}$$

dal risultante vettore X di dimensione 3 è possibile ottenere il raggio e il centro dell'arco di curvatura nel modo seguente:

$$r = \sqrt{\frac{X_0^2 + X_1^2}{4} - X_2}$$
$$x_c = \frac{X_0}{2}$$
$$y_c = \frac{X_1}{2}$$

Ottenuti raggio e centro dell'arco di curva possiamo calcolare l'angolo dell'arco trovato usando le coordinate del punto iniziale dell'arco $I = (x_i, y_i)$, le coordinate dell'ultimo punto dell'arco $F = (x_f, y_f)$ e le coordinate del centro dell'arco di curva $C = (x_c, y_c)$. Come primo passo è quindi necessario identificare la distanza tra il punto finale dell'arco di curva e il segmento costituito dal centro dell'arco appena calcolato e il punto iniziale dell'arco di curva nel seguente modo:

$$l = (x_i - x_c)^2 + (y_i - y_c)^2$$
$$u = \frac{(x_f - x_i) \cdot (x_c - x_i) + (y_f - y_i) \cdot (y_c - y_i)}{l}$$

A questo punto è possibile trovare le coordinate del punto P di intersezione tra il segmento $\overline{IC}$ e la sua perpendicolare passante per F (riferimento alla figura 3.3):

$$(x_p, y_p) = (x_i + u * (x_c - x_i), y_i + u * (y_c - y_i))$$

Da qui si può calcolare da distanza $\overline{FP}$ e risolvere l'angolo θ utilizzando l'arcoseno:

$$d = \sqrt{(x_p - x_f)^2 + (y_p - y_f)^2}$$

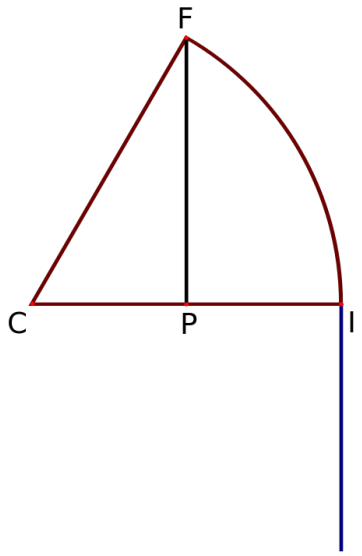

Figura 3.3: Rappresentazione visuale degli elementi utilizzati in questa sezione.

$$\theta = \arcsin^{-1}(d/r)$$

Una volta compiute le operazioni sopra descritte viene restituita una matrice contenente i parametri delle k curve che appaiono nell'immagine, costruita nel modo seguente:

$$\begin{bmatrix} flag_1 & \theta_1 & x_{c_1} & y_{c_1} & r_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ flag_n & \theta_n & x_{c_n} & y_{c_n} & r_n \end{bmatrix}$$

Tutte le curve che hanno meno di 5 punti identificati dall'algoritmo di path-shape vengono scartate, in quanto nonostante il processo di aggiustamento dei punti le piccole fluttuazione di essi possono completamente fuorviare i risultati.

3.4 Il processo di confronto

Il confronto tra curve deve risolvere un problema fondamentale, è infatti necessario avere una misura di distanza tra le curve in modo da poter identificare curve simili. In questo caso è possibile risolvere il problema usando una funzione che calcola la differenza tra i parametri di due curve date, ritornando un valore numerico ottenuto sommando le differenze dei parametri in input alla funzione. I parametri considerati dalla funzione di distanza sono

- **Il flag assegnato alle curve:** è il parametro considerato come il più importante, infatti una differenza su questo valore indica che le due curve confrontate si sviluppano in direzioni diverse quindi è impossibile che le due curve siano simili. La funzione di distanza è quindi programmata per restituire un numero grande in questo caso.

- **Il raggio della curva:** la funzione processa questo parametro con una semplice differenza in valore assoluto $\delta_r = |r_1 - r_2|$ sommata poi al valore restituito.

- **L'angolo di curvatura:** questo caso risulta leggermente più complicato dei precedenti, infatti la scarsa distanza a cui l'algoritmo di pathshape identifica la linea sul terreno rende complicato ottenere un angolo di curvatura corretto. É quindi necessario distinguere tra il caso in cui l'angolo di curvatura calcolato risulti minore dell'angolo da confrontare $\alpha_1 < \alpha_2$ e il caso inverso $\alpha_1 > \alpha_2$, assegnando dei pesi diversi ai due casi. Nel primo caso possiamo assumere che la curva non sia stata identificata completamente quindi assegneremo un peso basso alla differenza.

  $$\delta_\alpha = |\alpha_1 - \alpha_2| * 0.1$$

  Nel secondo caso invece, nonostante la curva non sia stata completamente identificata, possiamo osservare una parte di curva maggiore rispetto alla curva usata come confronto, quindi non assegnamo nessun peso alla differenza in questo caso $\delta_\alpha = |\alpha_1 - \alpha_2|$

Il valore di ritorno della funzione di distanza viene quindi calcolato come:

$$ret = \delta_r + \delta_\alpha$$

Risolto il problema della funzione di distanza la parte di confronto risulta un semplice calcolo del minimo tra le distanze. La curva identificata nell'immagine viene confrontata con una serie di curve date in input dal programmatore (che può quindi gestirsi la posizione ottenuta dagli encoder per inserire solo i parametri delle curve presumibilmente vicine al robot). Se i valori di distanza di una o più curve risultano abbastanza vicini vengono ritornati i valori di distanza calcolati e gli indici corrispondenti nella matrice data in input dal programmatore. In caso contrario viene restituita una matrice vuota.

Capitolo 4

Esperimenti

Per verificare i risultati ottenuti è stato necessario eseguire una serie di esperimenti, utili a valutare oggettivamente le prestazioni delle singole funzioni dell'algoritmo proposto. Tutti i test sono stati eseguiti sul mio pc personale del quale elenco le caratteristiche rilevanti:

- Processore: Intel i7 2100MhZ×8
- Ram: 8GB
- GPU: NVIDIA 630m 2GB

4.1 Uso di MATLAB

Per la maggioranza degli esperimenti svolti sono stati utilizzati degli script in MATLAB. Grazie all'uso di questo programma è stato possibile generare velocemente immagini, permettendo in questo modo di ottenere una quantità adeguata di immagini per la procedura di testing. É importante notare che le immagini generate in questo modo sono esenti da rumore e contengono curve perfette.

Un primo frammento di codice si occupa di prendere in input i parametri del percorso, grazie a delle funzioni che permettono di aggiungere uno alla volta tutti gli elementi necessari. Queste funzioni generano curve e rettilinei in base alla parte di percorso appena inserita, utilizzando come parametri lunghezza, differenza di angolazione nel caso di parti in rettilineo, o raggio e angolo di curva nel caso di elementi curvilinei.

Una volta impostato il percorso è possibile inserire una lista di coordinate (x, y, α) (con α inteso come orientamento del robot rispetto all'asse delle x) e un secondo script si preoccupa di posizionare un robot immaginario sul

percorso generando immagini analoghe a quelle ottenute facendo veramente muovere il robot lungo il percorso. Questo permette di velocizzare notevolmente le fasi di testing, soddisfacendo le necessità di immagini per eseguire test senza la necessità di usare effettivamente il robot.

4.2 Test sulla funzione di calcolo dei parametri della curva

Un primo test importante da eseguire è sulla funzione che calcola i parametri della curva. Infatti, in un caso ideale, l'algoritmo riuscirebbe a ricostruire la distanza dall'inizio della curva, il raggio e l'angolo di curvatura della stessa il più precisamente possibile.

I test sono stati svolti su un campione di 540 immagini differenti, ognuna contenente una curva a distanze diverse comprese tra 1 e 2.6 metri, inoltre ogni curva ha parametri differenti di raggio e angolo di curvatura, con un raggio che oscilla tra 1 e 5 metri. L'angolo di curvatura varia invece tra un angolo di -90° (curva di 90° a sinistra) e 90° (curva di 90° a destra). Inoltre i dati proposti indicano di solito una percentuale di errore sul calcolo del raggio di curva (di fatto il parametro più rilevante nella parte di confronto), quindi un errore del 5% può indicare un errore che varia tra 5 e 20 cm in base alla curva considerata. I dati ottenuti sono stati divisi in base a differenti criteri e conteggiati. Alcuni dati sono mostrati come percentuale sul numero totale (Fig. 4.1b, Fig. 4.2), mentre altri saranno proposti come semplice numero (Fig. 4.1a).

D. curva[m]\errore[%]	$\leq 5\%$	$\leq 10\%$	$\leq 15\%$	$\leq 20\%$	$\leq 50\%$	$\geq 50\%$
1.0	64	13	5	3	16	7
1.4	58	19	7	4	10	10
1.8	50	25	7	5	10	11
2.2	18	30	27	9	13	11
2.6	6	5	6	4	44	43

Err. medio su α: 45%

Tabella 4.1: Numero di dati per percentuale di errore sul calcolo, raggruppati in base alla distanza dalla curva.

É già stato discusso come sia complesso ottenere la curva completa, infatti l'algoritmo di pathshape risulta veramente affidabile per distanze inferiori ai ~220cm, con distanze massime ~3m. Per questo motivo identificare nell'immagine la curva nella sua interezza risulta essere un evento piuttosto raro.

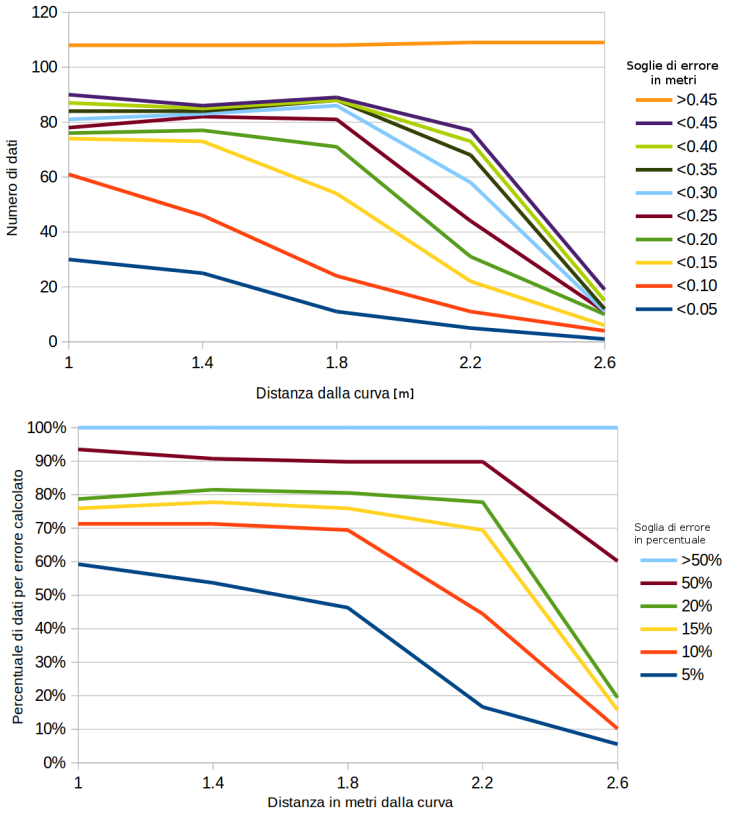

Figura 4.1: Anche in questo caso è stata calcolata la percentuale d'errore sul raggio e ordinati in base allla percentuale di errore.

Questo rende il calcolo dell'angolo di curvatura completamente inutile nella maggioranza dei casi, eccettuando i casi limite in cui abbiamo un raggio di curvatura relativamente piccolo (< 1.5m) e un angolo di curvatura inferiore ai 30°. A questo punto compare però un problema di affidabilità sul calcolo dei valori di raggio e centro dell'arco di curvatura. Avendo infatti dei parametri di questo tipo il numero di punti posizionati sulla curva sarà piuttosto piccolo, ciò provoca fluttuazioni maggiori portando ad errori maggiori nel calcolo.

I risultati ottenuti sul raggio di curvatura e sul punto di inizio della curva risultano invece più soddisfacenti (Tab. 4.1), infatti riusciamo a raggiungere nel 60% dei casi una percentuale di errore inferiore al 5% per dati ottenuti da curve a una istanza di 1 metro. Invece, se cominciamo a considerare valori ottenuti da curve posizionate a una distanza maggiore notiamo che l'errore comincia ad aumentare. Osservando infatti i dati ottenuti da curve tra gli 1.4 e i 2 metri vediamo che meno del 50% dei dati ha un errore inferiore al

5%, nonostante ciò si osserva che una percentuale di dati maggiore al 20% presenta un errore di calcolo tra i 5 e il 10%. In un ultimo caso osservando i valori calcolati su curve a distanze maggioridi 2 metri notiamo un peggioramento sensibile, con un numero di dati intorno al 40% che manifesta errori tra il 20% e il 50% e un ulteriore 40% dei dati che risulta errato di più di 50% del valore.

Concludendo risultano comunque dati peggiori di quanto sperato. Si nota particolarmente una diminuzione nella precisione del calcolo all'aumentare della distanza dalla curva. Questo è dovuto a un limite attribuibile all'algoritmo di pathshape. Infatti, è stato osservato che a distanze superiori ai 2.5 metri l'algoritmo raramente identifica dei punti all'interno della linea. In aggiunta a ciò i pochi punti ottenuti oltre i 2 metri risultano afflitti da errori, soprattutto se questi vengono identificati all'interno di una curva. La presenza di pochi punti afflitti da errori causa forti problemi nella fase di calcolo dei parametri della curva, portando a delle alterazioni significative nel risultato calcolato.

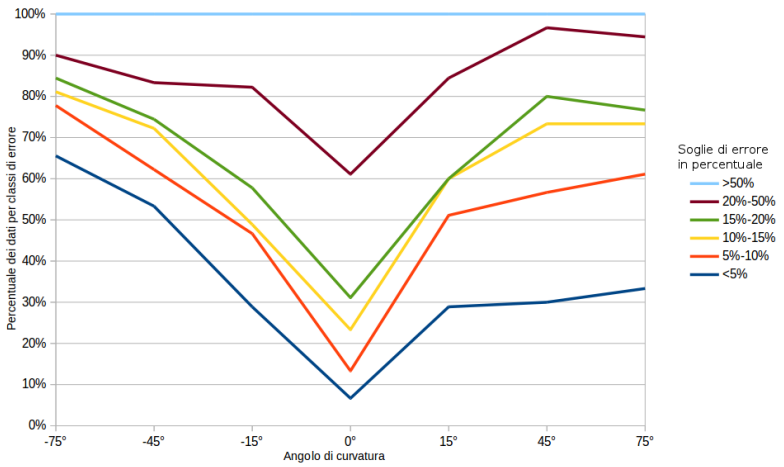

Figura 4.2: La percentuale di errore è stata calcolata comparando il raggio calcolato dall'algoritmo con il raggio noto. É quindi stato effettuato un conteggio sul numero di dati in base all'errore.

4.3 Test sulla funzione di matching

Per verificare le prestazioni ottenute dalla funzione di matching sono stati costruiti una serie di percorsi utilizzando MATLAB, contenenti diverse tipologie di curve. Per ogni curva sono state create circa 3 immagini, ognuna a

distanza diversa dalla curva presa in analisi quindi i parametri di ogni curva sono stati inseriti all'interno dell'algoritmo di matching.

Sono stati testati 5 percorsi differenti, ognuno contenente almeno 5 curve. Le curve avevano diversi raggi e angoli di curvatura per differenziare i vari casi tra di loro e avere un set di testing il più vario possibile. É importante notare che i parametri delle curve sono stati generati in modo tale da avere un minore numero di alterazioni dovute al comportamento dell'algoritmo di pathshape. Questo ci ha permesso di giudicare l'efficenza della funzione di matching indipendentemente dalle prestazioni dell'algoritmo di pathshape.

Percorso	Immagini #	Risultati positivi #	Successo %
Percorso 1	16	14	87%
Percorso 2	16	14	87%
Percorso 3	16	12	81%
Percorso 4	18	17	94%
Percorso 5	21	19	90%
Totale	87	76	87%

Tabella 4.2: Risultati ottenuti nei test della funzione di matching, per risultati positivi si intendono in casi in cui l'algoritmo identifica la curva correttamente.

Osservando la Tabella 4.3 si osserva che i risultati ottenuti dalla funzione di matching sono buoni. La curva osservata nell'immagine viene infatti riconosciuta nell'87% dei casi studiati. Inoltre, è stato riscontrato che nei casi di insuccesso l'errore era causato solitamente da due fattori:

- **Eccessiva distanza del robot dalla curva:** in alcuni casi ciò comportava un mancato riconoscimento dei punti in curva, in quanto essa risultava essere troppo distante, mentre in altri casi si riscontrava un errore nel calcolo dei parametri della curva dovuto a un numero basso di punti identificati all'interno di essa.
- **Errori dovuti all'algoritmo di pathshape:** in condizioni particolari (solitamente nel caso di curve con un raggio molto basso) l'algoritmo di pathshape manifestava delle forti deviazioni dal percorso, posizionando dei punti in posizioni anche molto lontane dal marker. Questo provoca spesso dei grossi errori nel calcolo, dovuti a questi punti altamente fuorvianti che non riescono a essere corretti nella fase preliminare.

In altri casi invece si osserva un errore in percorsi in cui vengono inserite curve con parametri molto simili tra loro. Questo avviene perché nel calcolo della distanza la curva identificata risulta vicina a entrambe portando ad

avere un riconoscimento errato. Nonostante ciò la curva corretta viene sempre restituita nel set delle curve vicine dalla funzione di matching.

Concludendo si osservano dei risultati positivi nell'87% dei test (numero che potrebbe essere portato a ~95% rimuovendo i dati fuorviati dal pathshape) numero piuttosto soddisfacente.

4.4 Confronto tra i tempi di esecuzione dell'algoritmo di pathshape e l'algoritmo di identificazione della curva

Raccogliendo i dati sul tempo computazionale necessario è stato possibile ottenere informazioni rilevanti pur eseguendo i test su un hardware molto più potente di quello effettivamente a disposizione, infatti è possibile utilizzari i dati sul tempo di esecuzione dell'algoritmo di pathshape come confronto per i dati sul tempo di esecuzione dell'algoritmo da me proposto.

T.totale[ms]	T.pathshape[ms]	T. id. curva[ms]	Rapporto[%]
43.07	42.90	0.17	0.95%

Tabella 4.3: Sommario del tempo necessario per l'esecuzione della parte di algoritmo di identificazione della curva, descritta nel capitolo 3, in rapporto al tempo di esecuzione dell'algoritmo di pathshape.

Possiamo osservare in questo modo che la parte di codice da me proposta ha un tempo di esecuzione che oscilla tra l'1‰e il 6‰dell'algoritmo di pathshape, quindi il suo tempo di esecuzione risulta trascurabile.

Capitolo 5

Conclusioni

La tesi propone un metodo per identificare e successivamente calcolare i parametri delle curve catturate da una webcam posta in posizione frontale rispetto al robot.

L'algoritmo di pathshape è in grado di identificare una linea all'interno dell'immagine e di ottenere un insieme di punti che stanno su di essa. Questo insieme di informazioni è stato usato come punto di partenza per il lavoro descritto. Abbiamo quindi iniziato osservando l'algoritmo stesso per aggiustare alcuni difetti scoperti durante le fasi iniziali. Successivamente il lavoro è proseguito con la costruzione di un sistema in grado di riconoscere la curvatura nella linea identificata nell'immagine. A questo punto dopo aver separato e isolato le varie curve presenti nella stessa, è stata implementata una funzione in grado di calcolarne i parametri. É infine stata aggiunta un'ulteriore funzione in grado di calcolare la distanza tra due curve aventi parametri differenti. I dati ottenuti nel passo precedente vengono quindi confrontati con i parametri delle curve presenti in memoria. Come ultimo passo viene restituito un set di curve simili a quella considerata.

Per verificare la precisione con cui questi algoritmi operano è stata adottata una tecnica di testing basata su simulazioni utilizzando MATLAB. Sono stati quindi valutati la correttezza dei parametri calcolati dall'algoritmo di identificazione della curva e il tempo di esecuzione della parte di riconoscimento della stessa. Questa valutazione è stata svolta in relazione al tempo di esecuzione dell'algoritmo di pathshape.

I risultati riguardanti il tempo di esecuzione sono stati senza dubbio ottimi. Il tempo di esecuzione della parte di codice relativa all'identificazione della curva e al pattern matching è risultato molto basso. Il rapporto tra il tempo necessario a eseguire la parte alla quale ho contribuito e quello necessario a eseguire l'algoritmo di pathshape è stato valutato in media inferiore a 1/100. Quindi per ogni unità di tempo impiegata per identificare e trovare

una curva corrispondente, il pathshape impiega solitamente più di 100 unità di tempo per l'elaborazione dell'immagine e l'estrapolazione della linea.

I risultati ottenuti sulla parte di identificazione dei parametri sono invece risultati non del tutto soddisfacenti. Infatti, è stato ottenuto un errore nel calcolo dei parametri della curva distribuito come segue:

- Nel 70% dei casi riguardanti dati ricavati da curve lontane al massimo due metri dal robot è stato riscontrato un errore inferiore al 10%.
- Nel 78% dei casi riguardanti dati ricavati da curve a una distanza compresa tra i 2 e i 2.5 metri è stato riscontrato un errore inferiore al 20%.
- Nel 21% dei casi riguardanti distanze superiori ai 2.5 metri risulta invece molto difficile ottenere dati precisi e l'errore nel calcolo dei parametri della curva si riduce a valori inferiori al 20%. Questo errore è dovuto ai limiti riscontrati nell'algoritmo di pathshape, i quali rendono quasi impossibile lavorare oltre distanze simili.

Un'ulteriore problema riscontrato nell'algoritmo utilizzato riguarda particolari tipi di curve aventi un angolo di curvatura inferiore a circa 20°, dove l'errore di calcolo si attesta intorno al 20% solo nel 50% dei test. Nonostante ciò, i risultati possono essere comunque sfruttati dopo una fase di pianificazione del percorso adeguata.

Possiamo concludere affermando che l'algoritmo che esegue la parte di matching della curva identificata nell'immagine risulta efficace. Dai dati è risultata infatti una percentuale di riconoscimento piuttosto alta, nonostante gli errori apparsi nella fase preliminare.

5.1 Lavori futuri

Uno dei primi aspetti che necessita una fase di miglioramento è l'affidabilità dell'algoritmo di pathshape. Questo può essere ottenuto aumentando sia la distanza a cui la linea viene identificata dall'algoritmo, sia la precisione su distanze superiori ai 2 metri. Ciò permetterebbe di eseguire l'algoritmo di identificazione della curva sulla base di dati più affidabili, ottenendo risultati migliori di quelli riscontrati attualmente. Infatti, l'aumento di precisione aiuterebbe ad identificare la curva con più sicurezza. In aggiunta l'aumento della distanza di identificazione permetterebbe lo studio dell'intera curva, consentendo un utilizzo più efficace dei parametri riguardanti l'angolo di curvatura. Una valida alternativa a questa soluzione è quella di migliorare la parte preliminare di correzione dei dati, in modo tale da poter lavorare con valori più

regolari. Tuttavia, questa soluzione non risolve il problema della distanza di identificazione.

Il potenziamento dell'algoritmo di pathshape, adottando tecniche di clustering quali [5], risulterà molto più attuabile dopo la sostituzione della scheda embedded utilizzata con una versione più potente. Inoltre, l'utilizzo di una GPGPU[1] (General Purpose GPU) permetterebbe di paralelizzare meglio le fasi di elaborazione delle immagini, migliorando ulteriormente le prestazioni dell'algoritmo. Infine un'altra ottimizzazione consiste nel migliorare l'algoritmo che processa e marca i punti dell'algoritmo di pathshape. L'obiettivo è quello di gestire un maggior numero di errori riguardanti i valori restituiti dall'algoritmo di pathshape, aumentandone quindi l'efficacia e l'affidabilità.

[1] `http://gpgpu.org/`

Ringraziamenti

Desidero ringraziare il Prof. Luigi Palopoli e Federico Moro che mi hanno seguito e consigliato durante lo sviluppo di questo progetto.
Un sentito ringraziamento va a mia madre e a mio padre che mi hanno sempre aiutato, sostenuto e incoraggiato.
Esprimo la mia gratitudine per i miei compagni di corso: Giovanni, Alex, Roberto, Luca, Mirko e a tutta la mia compagnia per avermi accompagnato durante questi tre anni di università.

Appendice A

Pseudocodice e documentazione

Algorithm 1: L'algoritmo **filterPoints**, necessario alla correzione preliminare del codice

Dati:
matrix: matrice $3 \times n$ contenente i valori ritornati dall'algoritmo di pathshape
n: numero di vettori contenuti in *matrix*
rev: indice al quale avviene l'inversione citata nel capitolo 2
Risultato: La matrice di vettori matrix ordinata e normalizzata

$tmp \leftarrow [\ ]$;
for $i \leftarrow 0$ **to** rev **do**
    tmp $[i] \leftarrow$ matrix $[i]$;
$j \leftarrow rev + 1$;
for $i \leftarrow n - 1$ **to** $rev + 1$ **do**
    tmp $[j] \leftarrow$ matrix $[i]$;
    $j \leftarrow j + 1$;
`// qui si conclude il riordinamento e comincia la fase di normalizzazione`
for $e \in$ tmp **do**
    prendere r_1 *e* l_1 *come elementi precedente e successivo a* e, r_2 *come precedente di* r_1 *e* l_2 *come successivo di* l_1;
    $e \leftarrow (r_2w_2 + r_1w_1 + e + l_1w_1 + l_2w_2)/(2w_2 + 2w_1 + 1)$;

Algorithm 2: L'algoritmo **filterCurve** effettua uno studio preliminare sul comportamento dei singoli punti, riportando dati utili all'identificazione della curva.

Dati: matrix: matrice dei dati del pathshape ordinati, dim: numero di vettori presenti nella matrice

Risultato: Viene restituito un vettore contenente dei flag indicanti il comportamento della curva

```
behavior ←[ ];
flag ← 0; // il valore può essere 0, -1 o +1
counter ← 0;
W_1 ← 0.065 · pshapeStep;
W_2 ← 0.015 · pshapeStep;
if dim < 5 then                    // Pochi elementi su cui lavorare
    return;
for i ← 1 to dim − 1 do
    // inizialmente è necessario trovare la differenza di
       angolazione tra un punto e il successivo
    α ← matrix [i].angle-matrix [i + 1].angle;
    if α >W_1 then
        // --|----|-0-|----|***
        flag ← +1; // Indicante curva a destra
        assegnare ai counter elementi precedenti behavior [i] il valore
        flag;
        counter ← 0;
    else if α >W_2 then
        // --|----|-0-|********|--
        if flag == -1 then  // In caso di una inversione di
           tendenza la flag viene istantaneamente resettata
           a 0
        flag ← 0;
        ;
        assegnare ai counter elementi precedenti di behavior [i] il valore
        flag;
        counter ← 0;
    ⋮
```

```
for i ← 1 to dim − 1 do
    ⋮
    else if α > 0 then
        // --|----|-0**|----|--
        if flag != 0 then        // Se il punto risultava in curva
            counter ← counter +1;
            if counter > 2 then
                // Nel caso la condizione venga saltata
                   l'assegnazione di behavior viene sospesa per
                   un giro
                flag ← 0;
                assegna a behavior [i], behavior [i − 1] e behavior [i − 2]
                il valore di flag;
                counter ← 0;
        else
            behavior [i] ← 0; // Viene assegnato come rettilineo
    else if α > -W_2 then
        // --|----|**0-|----|--
        if flag != 0 then
            counter ← counter +1 if counter > 2 then
                flag ← 0;
                assegna a behavior [i], behavior [i − 1] e behavior [i − 2]
                il valore di flag;
                counter ← 0;
        else
            behavior [i] ← 0;
    else if α > -W_1 then
        // --|********|-0-|----|--
        if flag == +1 then  flag ← 0;
        ;
        assegnare ai counter elementi precedenti di behavior [i] il valore
        flag;
        counter ← 0;
    ⋮
```

```
for i ← 1 to dim − 1 do
    ⋮
    else
        // ***|----|-0-|----|--
        flag ← -1; // Indicante curva a sinistra
        assegnare ai counter elementi precedenti di behavior [i] il valore
        flag;
        counter ← 0;
if counter > 0 then
    Assegnare gli ultimi counter punti sospesi al valore di flag;
```

Algorithm 3: L'algoritmo **splitCurves** si occupa di scorrere i dati precedentemente ottenuti dalla funzione *filterCurve* per ricavare i punti dove le eventuali curve iniziano e finiscono.

Dati: curve_data: vettore contenente i flag assegnati usando la funzione *filterCurve*, matrix: la matrice dei dati ottenuta dal pathshape, dim: il numero di punti contenuti in matrix e curve_data

Risultato: Viene restituita una matrice di dimensione $3 \times n$, con n pari al numero di curve rilevate. Ogni riga viene riempita con dei valori che indicano l'indice dove la curva ha inizio e l'indice dove essa ha fine.

```
ret_matrix ← [ ]; last ← 0; // mantiene l'ultimo fag in memoria
count ← 0; // mantiene in memoria il numero di curve trovate
curve ← 0; // usato per caricare in memoria il flag corrente
for i ← 1 to dim do
    curve ← curve_data [i];
    if curve != last then
        if last != 0 then
            // caso di passaggio * → curva o viceversa
            ret_matrix [count].end ← i;
            ret_matrix [count].flag ← last;
            count ← count + 1;
            if curve != 0 then ret_matrix [count].begin ← i;
            // passaggio curva → altra curva
        else ret_matrix [count].begin ← i;          // per gestire il
        passaggio da rettilineo a curva
if curve != 0 and last == curve then   // curva idenfiticata ma
non ancora chiusa
    ret_matrix [count].end ← dim;
    ret_matrix [count].flag ← last;
return count
```

Algorithm 4: L'algoritmo **findCurvature** si occupa di trovare i parametri della curva visibile nell'immagine.

Dati: matrix: La matrice ottenuta dal pathshape, ordinata e integrata con i flag indicanti la curvatura nei singoli punti,
beginpoint: indice del primo punto della curva da considerare,
endpoint: indice dell'ultimo punto della curva da considerare,
dim: il numero di vettori in matrix
Risultato: Un vettore contenente raggio, coordinate del centro e distanza dell'arco di curva

costruire la matrice mat di dimensioni $3 \times (\text{endpoint} - \text{beginpoint})$ *inserendo in ogni linea un vettore* $(x_i, y_i, 1)$ *costruito come* $x_i = \text{matrix}[\text{beginpoint} + i]$ *e* $y_i = \text{matrix}[\text{beginpoint} + i]$;

costruire il vettore verticale vect di dimensione $(\text{endpoint} - \text{beginpoint})$ *inserendo in ogni linea il valore seguente:* $vect_i = -(x_i^2 + y_i^2)$ *con* x_i *e* y_i *come sopra*;

si applica una decomposizione a valori singolari sulla matrice mat per ottenerne la pseudoinversa mat^{-1};
`// il passo precedente è necessario per poter calcolare la divisione sinistra` $mat \setminus vect$
$X \leftarrow mat^{-1} \cdot vect$; `//` X `risulta essere un vettore di 3 valori`
`// qui sotto vengono calcolati raggio e centro dell'arco di curva`
$r \leftarrow sqrt(X_1^2 + X_2^2)/4 - X_3$;
$x_c \leftarrow -0.5 \cdot X_1$;
$y_c \leftarrow -0.5 \cdot X_2$;
`// si passa quindi a calcolare l'angolo di curvatura`
$x_{start}, y_{start} \leftarrow$ matrix [beginpoint].x, matrix [beginpoint].y;
$x_{end}, y_{end} \leftarrow$ matrix [endpoint].x, matrix [endpoint].y;
$l \leftarrow (x_{start} - x_c)^2 + (y_{start} - y_c)^2$;
$u \leftarrow ((x_{end} - x_{start}) \cdot (x_c - x_{start}) + (y_{end} - y_{start}) \cdot (y_c - y_{start}))/l$;
$x_p, y_p \leftarrow (x_{start} + u \cdot (x_c - x_{start})), (y_{start} + u \cdot (y_c - y_{start}))$;
$d \leftarrow sqrt(x_{end} - x_p)^2 + (y_{end} - y_p)^2$ $\alpha \leftarrow \arcsin(d/r)$;
return $[\alpha, x_c, y_c, r]$;

Algorithm 5: L'algoritmo **curveDistance** serve per calcolare una distanza fra 2 curve.

Dati: α_1, r_1, $type_1$: angolo, raggio e direzione (sx/dx) della prima curva salvata in memoria; α_2, r_2, $type_2$: angolo, raggio e direzione della seconda curva calcolati grazie agli algoritmi precedentemente descritti.

Risultato: Viene restituito un numero indicativo della distanza tra le due curve, più alto il numero meno simili sono le due curve.

$k \leftarrow 0$;
if $type_1 != type_2$ **then** `// le curve muovono in direzioni opposte`
    $k \leftarrow k + 1000$;
    return k;
else
    `// il secondo caso viene pesato di meno in previsione agli errori di calcolo verificatisi`
    if $\alpha_1 < \alpha_2$ **then** $k \leftarrow k+(\alpha_2-\alpha_1)$;
    else $k \leftarrow k+(\alpha_1-\alpha_2)*0.25$; //
    $k \leftarrow k + |r_2 - r_1|$;
return k;

Algorithm 6: L'algoritmo **matchPath** effettua il matching tra i dati calcolati dall'immagine di una curva e un set di curve in memoria.

Dati: curvedata: struttura dati contenente i diversi parametri della curva studiata, pathdata: vettore contenente delle strutture dati che mantengono i parametri di tutte le curve lungo il percorso.
Risultato: Una o più strutture aventi i dati delle curve più vicine.

```
minel ← ⊥;
mindist ← 1100;
maxtol ← 0.3;
set ← ∅;
for el ∈ pathdata do
    dist ← curveDistance(el.ang, el.r, el.flag, curvedata.ang,
    curvedata.r, curvedata.flag);
    if dist < mindist then
        mindist ← dist;
        minel ← el;
if mindist < 2 then          // nessuna curva è troppo distante
    set.add(minel);
    for el ∈ pathdata do
        if el == minel then continue;
        dist ← curveDistance(el.ang, el.r, el.flag, curvedata.ang,
        curvedata.r, curvedata.flag);
        if dist < (mindist + maxtol) then
            set.add(el);
return set;
```

Bibliografia

[1] Luca Zamboni. Riconoscimento e stima distanza di cartelli stradali tramite webcam., 2014.

[2] Moro Federico. Vision-based robust path reconstruction for robot control, 2013.

[3] J. Canny. A computational approach to edge detection, ieee trans. on pattern analysis and machine intelligence, 8(6), pp. 679-698, 1986.

[4] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[5] Nitin Aggarwal and William Clement Karl. Line detection in images through regularized hough transform. In *ICIP*, pages 873–876, 2000.