Universitá di Trento

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in Informatica

Tesi Finale

RICONOSCIMENTO E STIMA DISTANZA DI CARTELLI STRADALI TRAMITE WEBCAM

Relatore:
Prof. Luigi Palopoli

Graduant:
Luca Zamboni

Co-Relatore:
Federico Moro

Anno academico 2013-2014

Indice

Capitolo 1

Panoramica generale

1.1 Introduzione

[1]

Negli ultimi anni si é sviluppata la necessitá di realizzare robot o macchine in grado di spostarsi autonomamente nell' ambiente senza bisogno dell' aiuto umano. In questo progetto per riuscire a far muovere il robot nell' ambiente ci serviamo di due telecamere una frontale una laterale montate su di esso e di una linea nera sul superficie del pavimento. La linea nera rappresenta il percorso che il robot deve riuscire seguire attraverso le due telecamere. Il robot conosce a priori il percorso e deve sapere esattamente dove si trova in esso. Ogni telecamera é collegata a una scheda Beaglebone che serve per l' elaborazione delle immagini. Entrambe le telecamere servono per riuscire a riconoscere la linea nera sul terreno e riuscire a guidare il robot attraverso il percorso. La telecamera frontale elabora le immagini riconoscendo il percorso e pianificando la strada da seguire. La telecamera laterale invece serve per sapere a che distanza é la linea e cerca di correggerne la traiettoria. Inoltre si é deciso di aggiungere anche dei segnali lungo il percorso che servono per dare delle indicazioni al robot e anche per sapere esattamente in che parte del percorso esso si trova stimandone la distanza da essi. In questo progetto si parte da un algoritmo giá realizzato per riconoscere i segnali posizionati sul terreno e si propone un modo per migliorarne le prestazioni nella velocitá dell' elaborazione delle immagini, aumentare la precisione e la distanza a cui viene riconosciuto un segnale e infine stimare la distanza da esso.

1.2 Il progetto

splittare l'introduzione

1.3 Algoritmo riconoscimento

L' algoritmo di riconoscimento cartelli prende in input le immagini dallo streaming video della telecamera frontale. Esso deve elaborare ogni frame e riuscire a individuare se in essa é presente un cartello e in seguito riuscire a riconoscere di che tipo di cartello si tratti. Dal momento che l'algoritmo deve essere eseguito su un corpo in movimento le decisioni devono essere prese alla svelta perché basta un attimo di indecisione che il robot puó uscire di strada. Questo impone che l'algoritmo deve avere un tempo di esecuzione abbastanza ristretto. Facendo un esempio se il robot si muovesse alla velocitá di 10 Km/h esso percorrerebbe poco meno di 3 m/s (2.7777 m/s). Quindi se l'algoritmo per riconoscere il cartello impiegasse un secondo, significherebbe che se un cartello é posizionato a meno di 3 metri allora esso viene riconosciuto solo quando viene superato. Dunque si cerca di contrarre il piú possibile il tempo di esecuzione dell' algoritmo. Oltre a questo bisogna che il riconoscimento del cartello sia robusto. Per robustezza si intende la capacitá di:

- Riconoscere il segnale a varie distanze sia da vicino che da lontano.
- Individuare il cartello a diversi livelli di luminositá. Questo perché il robot deve essere in grado di guidare autonomamente sia di giorno, caratterizzato da un' intensa illuminazione, che di notte quando invece la luminositá é scarsa.
- Riuscire a riconoscere il segnale da varie angolature, attuando una trasformazione prospettica. Questo perché mentre si é in moto non sempre si é perfettamente allineati con il cartello.
- Riuscire a filtrare le forme geometriche dei cartelli, tralasciando le altre. Se ad esempio in un'immagine sono presenti piu quadrati, individuare tra questi solo quelli relativi ai cartelli.

1.4 Motivazioni

ahahahah ma che motivazione

1.5 Outline

Nel secondo capitolo verrá descritto come lavora l'algoritmo esistente e verranno analizzate prestazioni e efficacia. Nel terzo capitolo sono proposte soluzioni per migliorare l'algoritmo descritto nel capitolo precedente. Nel

quarto capitolo invece sono esposti i dati raccolti delle prove sperimentali riguardo i nuovi cambiamenti. Nel quinto capitolo ci sono l'analisi dei dati del capitolo precedente e le conclusioni del progetto.

Capitolo 2

Analisi algoritmo esistente

In questo capitolo verrá analizzato il codice l'algoritmo esistente e verrano valutate prestazioni ed efficacia di esso.

2.1 Ambiente di sviluppo

L' hardware utilizzato per il progetto é il seguente:

- Processore : AM335x 720MHz ARM Cortex-A8
- RAM: 256 MB
- Telecamera: WebCam 640 x 480 pixel

L' algoritmo per riconoscere i segnali ha la necessitá di elaborare le immagini provenienti da uno stream video. L' essendo che il codice viene eseguito su una scheda con processore non particolarmente potente e avendo a disposizione poca RAM le performance per questo algoritmo sono un aspetto fondamentale. Per questo motivo é stato scelto come linguaggio c++ che offre performance ottime. Inoltre viene utilizzata la libreria open source chiamata OpenCV che offre molte funzioni per l' elaborazione immagini e video e molto indicata per applicazioni real time. Inoltre OpenCV non obbliga nessuna restrizione sulla licenza del software.

2.2 I Cartelli

I cartelli individuati da questo algoritmo sono quelli rappresentati nelle figure 2.1. Essi hanno un contorno nero su sfondo bianco con all'interno una figura che puó essere un triangolo o un cerchio. É stato scelto di usare un bordo

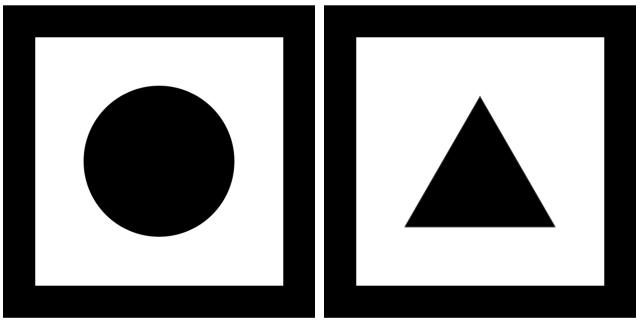

Figura 2.1: Cartelli riconosciuti

nero per facilitare il riconoscimento del cartello. Questo perché se fosse stato tutto bianco e nel'immagine ci fosse stato del bianco sullo sfondo del cartello nell'immagine sarebbe stato difficile se non impossibile distinguere il cartello. Invece cosi facendo si ha uno stacco netto con le figure sul retro del cartello. La larghezza della parte bianca del cartello ha una proporzione di $\frac{8}{5}$ rispetto al diametro del cerchio e alla base del trangolo.

2.3 Impostazione generale

Passiamo adesso all' analisi dell' algoritmo esistente dicendo a grandi linee i principali passi di esecuzione che verranno analizzati successivamente. L'algoritmo prende in input un' immagine presa dallo streaming video della telecamera frontale. L'immagine é rappresentata con una matrice di pixel N x M (nel progetto tipicamente 640x480 pixel). La cella [i,j] contiene il colore del pixel corrispondente. Ad alto livello l'algoritmo esegue le seguenti operazioni:

- Individuazione dei cartelli nell' immagine nel nostro caso dei quadrati chiamati ROI (Region of Interest).
- Deve eseguire una trasformazione prospetica delle ROI cioé se l'immagine é stata presa da una certa angolatura essa deve essere trasformata portandola ad una visualizzazione frontale .
- Individuazione di un triangolo o di un cerchio all'interno dei quadrati cioé riconoscere il cartello scartando presunte figure non tali.

Di seguito passeremo ad analizzare nel dettaglio i passi dell'algoritmo.

2.3.1 Individuazione cartello

Prima di procedere all' effettiva individuazione del cartello é necessario trasformare l'immagine proveniente dallo stream video da colorata a bianco e nero. Per fare tale operazione si utilizza la funzione di OpenCV cv:cvtColor(). Successivamente all' immagine in scala di grigi si applica il filtro di Canny. Il filtro di Canny serve a individuare i bordi degli oggetti in un immagine. L'algoritmo di Canny a grandi linee funziona nel seguente modo: si passano due valori alla funzione che saranno poi le nostre soglie. Poi si passa ad analizzare punto per punto dell'immagine calcolandone il gradiente in esso. Se il gradiente é:

- Inferiore alla soglia bassa, il punto é scartato;
- Superiore alla soglia alta, il punto é accettato come parte di un contorno;
- Compreso fra le due soglie, il punto é accettato solamente se contiguo ad un punto giá precedentemente accettato.

Per eseguire il filtro di Canny all' immagine su utilizza l'omonima funzione cv::Canny() di OpenCV. Una volta ottenuti i bordi essi si dilatano con la funzione cv::dilate() per eliminare eventuali buchi ai vertici. Poi si utilizza la funzione cv::approxPolyDP() che prende in input i bordi e ricava i poligoni approssimati. Da questi poi si tengono solo i poligoni che hanno 4 lati che siano convessi. Dopo di che si fa un filtraggio e si controlla che la differenza dei lati opposti sia entro un certo errore cioé che siano di lunghezza simile e si eliminano anhce eventuali duplicati. I quadrati risultanti sono le nostre ROI (Region of interest) cioé i nostri presunti cartelli dove andare a cercare il simbolo all interno.

2.3.2 Trasformazione prospettica

Dopo aver trovato i presunti cartelli bisogna eseguire una trasformazione prospettica. Essa consiste nel raddrizzare' il contenuto della ROI per rendere accurata la fase successiva cioé riconoscere la forma all' interno del cartello. Per effettuare la trasformazione si usa la funzione cv::getPerspectiveTransform() che calcola la matrice di trasformazione del sistema. Poi viene usata la funzione cv::warpPerspective() che effettua l' effettiva trasformazione. Immagine esempio

2.3.3 Riconoscimento cartello

Come ultimo passo bisogna determinare di che tipo di cartello si tratta oppure scartarlo, se esso era un falso cartello cioé una forma geometrica simile al cartello e riconosciuto come tale dai punti precedenti dell'algoritmo. Questo passo prende in input le ROI riconosciute ed elaborate in precedenza. Per riuscire a riconoscere i cerchi e i triangoli si usa una tecnica simile a quella per ricavare i quadrati dalla foto originale. In entrambe si usa il filtro di Canny per ricavare i bordi della figura. Una volta ricavati i bordi le strade si dividono per le due forme. Per riconoscere il triangolo si usa la funzione cv::approxPolyDP() come per riconoscere i quadrati peró in questo caso ovviamente si ricavano solo i poligoni con 3 facce. Invece per riconoscere il cerchio si usa la funzione cv::HoughCircle(). Questa funzione é contenuta nella libreria OpenCV e prende semlicemente in input l'immagine e restituisce i contorni che formano un cerchio entro un certo errore. Poi per entrambi i casi si controlla che la figura sia circa al centro del quadrato del cartello entro un certo scarto, altrimenti si scarta e si controlla pure che abbia le proporzioni giuste cioé la larghezza del cartello é $\frac{8}{5}$ della larghezza della figura interna anche qui entro un certo errore.
Arrivati a questo punto si é riusciti a determinare il tipo di cartello a cui appartiene la ROI passata in input.

2.4 Analisi dati algoritmo

Adesso passiamo all' analisi prestazioni e precisione algoritmo.
Per prima cosa analizziamo la precisione dell'algoritmo con immagini prese: a varie distanze, con angolature non superiori ai 15° e di giorno cioé con luce accettabile. Questi sono i dati ricavati utilizzando cartelli con larghezza circa di un lato corto di un foglio A4.

	% ROI riconosciuti	% Segnali riconosciuti
0 metri	100 %	100 %
1 metro	100 %	80 %
2 metri	100 %	10%
3 metri	60 %	0 %
4 metri	20 %	0 %

Da una prima analisi si evince che l'algoritmo ha delle difficoltá dovute alla distanza. Quello che si nota analizzando i dati della tabella é che l'algoritmo trova delle difficoltá a trovare ROI nell'immagine quando il cartello é posto a piďi due metri di distanza. Inoltre ha difficoltá ancora maggiori

quando ha trovato il cartello ma deve determinare di che tipo di segnale si tratta. Infatti anche in questo caso se si va oltre il metro la precisione cala drasticamente.

mettere dati lentezza se ce la fai

2.5 Conclusioni

Sicuramente la parte principale dell' algoritmo é la lentezza in quanto impiega circa un secondo per riconoscere un segnale. Se per esempio il robot si sposta alla velocitá di 10 km/h esso riconosce il cartello solo dopo 1 secondo cioé quando il robot ha percorso 3 metri. Quindi si cerca migliorare, cioé di ridurre il piú possibile il tempo di esecuzione.
Un altro problema importante é che l'algoritmo non riesce a riconoscere i cartelli se questi sono posti a piú di un metro di distanza. Questo implica che il robot deve andare lentissimo perché un segnale abbia efficaca e che sia riconosciuto prima di averlo sorpassato. Anche in questo punto bisogna cercare aumentre la distazna a cui vengono riconosciuti i cartelli.
Nel capitolo successivo verranno proposti dei metodi per risolvere il problema della lentezza dell' algoritmo e il problema della poca efficacia del riconoscimento di un cartello da una distsanza superiore al metro. Inoltre si implementa anche un metodo per calcolare la distanza dal cartello.

Capitolo 3

Miglioramenti

In questo capitolo si espongono degli algoritmi per migliorare prestazioni e robustezza dell'algoritmo. Inoltre si propone un metodo per calcolare la distanza dal cartello.

3.1 Cambio Cartelli

In questa sezione si analizza la problematica della distanza e si propone un metodo per risolverla.

3.1.1 Analisi problematica

Come analizzato nel precedente capitolo uno dei problemi principali é l'individuare il cartello in lontananza e riconoscere che tipo di cartello si tratta. Essendo la risoluzione non troppo elevata (640x480) quando ci si allontana dalla fotocamera le figure sono sempre piú piccole. Questo ovviamente ha delle ripercussioni sull' algoritmo. Di seguito verranno proposti dei metodi per cercare di risolvere questa problematica.

3.1.2 Cambio Cartelli

Ora passiamo ad analizzare il problema del riconoscimento delle figura all' interno del cartello.
I cartelli in lontananza hanno una figura interna facilmente confondibile o addirittura non si riescono a riconoscere. Questo é dovuto al fatto che la risoluzione é bassa e quindi l'interno di un cartello presenta pochi pixel. Immagine Queste immagini sono l'interno di un cartello a distanza di 3 metri. Come si puó vedere le due figure sono poco nitide e l'algoritmo ha un elevato

tasso di errore nel riconoscerle perché basta una differenza di pochissimi pixel per fallire il riconoscimento. Per ovviare a questo inconveniente é stata cambiata la grafica dei cartelli.

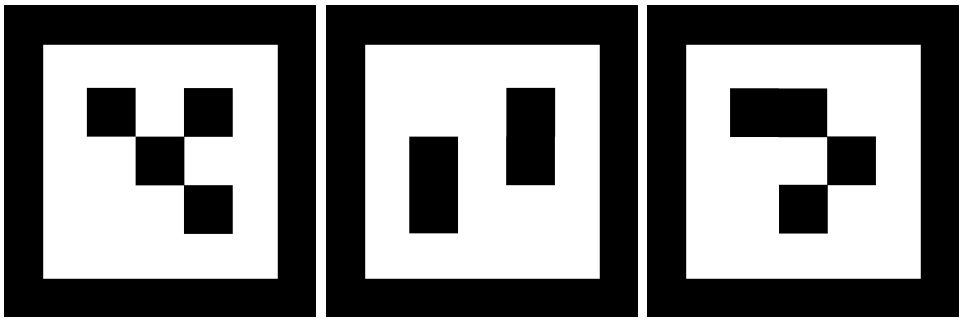

Figura 3.1: Nuovi cartelli

3.1.3 Analisi nuovi cartelli

Nella grafica dei nuovi cartelli é stato mantenuto il bordo nero e il fondo bianco che permetto di trovarli facilmente nell'immagine. É stato cambiato invece la grafica interna come si puó vedere nella figura 3.1. In centro alla parte bianca del cartello c'é una griglia di 3x3 quadrati di uguali dimensioni dei quali solo alcuni sono neri. Il metodo utilizzato per decidere se un quadrato é nero o bianco é che si cerca di massimizzare la differenza fra i vari cartelli. Nel nostro caso i tre cartelli hanno una differenza tra uno e l'altro del 66%. Cio significa che se si prendono due cartelli solo 3 quadrati su sei della griglia sono colorati nella stessa maniera.
é stato scelto questo tipo di cartello perché ogni segnale differsice molto da un altro segnale e quindi é difficile confondere i due anche se ci sono errori di qualche pixel dovuti alla scarsa risoluzione.

3.1.4 Implementazione nuovo riconoscimento

L' algoritmo per l'implementazione é totalmente diverso da quello precedente. Mentre quello preesistente si avvaleva del filtri di canny e cercava le forme geometriche dentro il cartello queto utilizza un piú semplice pattern matching ottenendo sia risultati migliori sotto forma di prestazioni che di percentuale di cartelli riconosciuti.
La funzione prende in input la porzione di immagine interna al cartello (ROI) in scala di grigi e il vettore contenente i cartelli conosciuti. Per prima la porzione all'interno dell immagine viene convertita in bianco e nero utilizzando

la funzione di openCV cv::threshold(). Il livello di threshold é deciso facendo la media di livello di grigio di tutta l'immagine. Il vantaggio di avenre un livello ti threshold variabile e non fisso é che se l'immagine viene presa in un ambiente scuro esso si alza automaticamente discriminando il bianco dal nero come nell'immagine successiva. Immagine.
A questo punto l'immagine viene ingrandita a una dimensione nota e si tagliano i bordi neri che potrebbero esserci come rimanenze del bordo del cartello e si taglia anche la parte bianca attorno alla griglia di modo che rimanga solamente la figura interna. Per tagliare i bordi neri si prendono in considerazione i pixel esterni che formano la cornice. Se nella cornice é presente almeno un pixel nero allora la si elimina e si prende in considerazione la nuova cornice eseguendo il passo precedente fintanto che la cornice é formata da soli punti bianchi. Ora si taglia la parte bianca attorno alla figura. Per eliminarla si esegue un procedimento analogo solo che in questo caso si taglia il bordo solo se la cornice ha i pixel colorati tutti di bianco. Il taglio dei bordi consente di estrarre la figura interna al cartello ma serve anche come filtro a falsi cartelli riconosciuti come tali dai passi precedenti dell'algoritmo. Un cartello non é tale con molta probabilitá verrá tagliato praticamente tutto restituendo solo un pixel quando si taglia il nero perché a meno che non abbia bordi bianchi ecco continuerá a trovare del nero sul bordo fino a superare la grandezza minima dell'interno del cartello impostata al 70% della larghezza dell'algoritmo.
Adesso si portano sia i cartelli che l'immagine interna al cartello a una grandezza uguale. A questo punto si ha solo la griglia interna al cartello e si passa all'effettivo riconoscimento di esso. Per identificare la griglia interna facendo un semplice pattern matching fra cartelli e parte interna che cartello che vogliamo riconoscere. Per fare pattern matching basta fare uno xor pixel a pixel tra le due immagini da confrontare. Se un cartello ha una percentuale di pixel corrispondenti piú alta degli altri e piú alta di una soglia impostata del 70% verrá riconosciuto come cartello corrispondente.

3.2 Vanishing Point

Qualcosa forse altrimenti nnt

3.2.1 Analisi ambiente

I cartelli vengono sempre posizionati alla fine di un rettilineo e prima di una curva e mai durante o appena dopo di essa. Cosi facendo le immagini che dovranno essere analizzate saranno piú nitide per il semplice fatto che

la telecamera sta facendo meno movimento laterale e le immagini risultano meno 'mosse' di conseguenza meno disturbate.
Analizzando i frame catturati dalla fotocamera frontale si puó notare che la posizione dei cartelli all' interno dell' immagine é abbastanza fissa. Infatti in quasi tutte le immagini i cartelli si trovano sulla linea dell' orizzonte dato che essi sono fissi su di esso. Piú ci si allontana e piú' i cartelli sono fissi sull orizzonte.

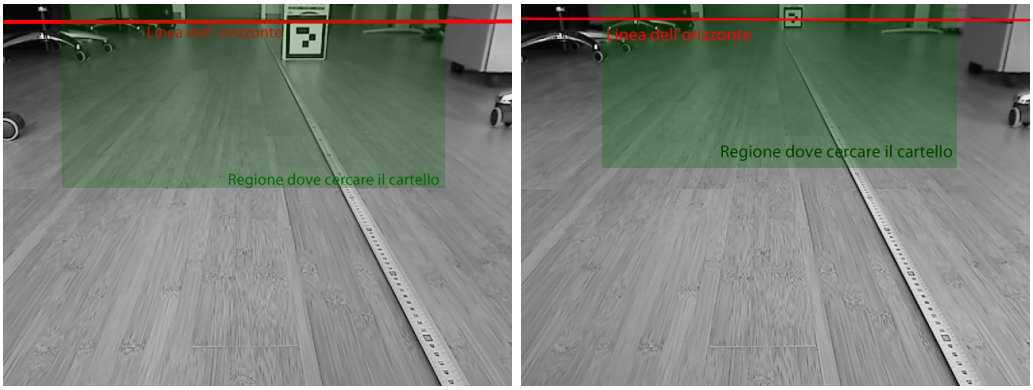

Figura 3.2: In rosso la linea dell'orizzonte e in verde la regione dell'immagine dove dovrebbe essere cercato il cartello

3.2.2 Punto sull'orizzonte

Nella sezione precedente é stato detto che il cartello é conveniente cercarlo prima sulla linea dell'orizzonte. Ma come fare riconoscere l' orizzonte? La soluzione é l' implementazione di un algoritmo che calcola il punto sull'orizzonte detto anche Vanishing Point.

Il vanishing point é un punto nell'immagine che é il risultato dell' intersezione del prolungamento delle linee parallele sul piano di immagine. Per esempio le linee bianche che stanno ai bordi di una strada sono parallele ma nell'immagine se si prolungano si incontreranno sulla linea dell' orizzonte. Sfruttando questo principio é possibile calcolare il Vanishing Point.

3.2.3 Calcolo Vanishing Point

Come detto in precedenza per calcolare il Vanishing Point servono le linee ai bordi della strada quindi per prima cosa bisogna estrarre i segmenti dall' immagine. Per fare ció é stata usata la funzione lsd() (line segment detection) che prende in input l'immagine e restituisce un array di **double**. Gli elementi dell' array vanno presi 7 alla volta e vanno utilizzati come segue:

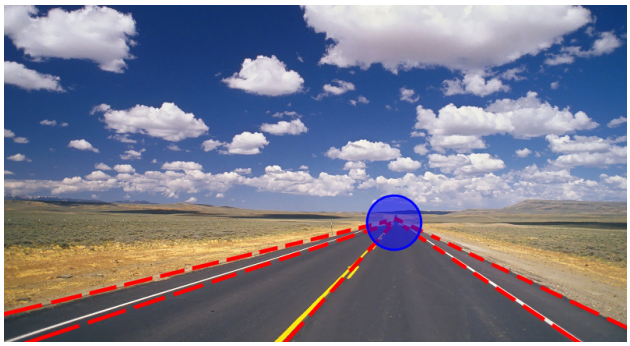

Figura 3.3: In rosso le linee che dovrebbero essere rilevate e in blu l'area dove approssimativamente esse si incontrano

1. **x** del punto di partenza del segmento nell' immagine
2. **y** del punto di partenza del segmento
3. **x** del punto di arrivo del segmento
4. **y** del punto di arrivo del segmento
5. **w** lunghezza del segmento
6. **p** p é la precisione con cui é calcolato il segmento
7. **logNfa**

Successivamente per ogni segmento si calcola il punto medio di esso e la pendenza. I segmenti successivamente vengono filtrati. I segmenti devono:

- Essere piú lunghi di una certa lunghezza, impostata al 10% della larghezza dell'immagine, altrimenti potrebbero essere solo rumore o oggetti al bordo della strada.
- Avere una pendenza tra 10° e 70° questo perché le linee orizzontali non vengono utilizzate e le linee verticali, cioé piu di 70°, potrebbero essere segmenti non appartenenti alla strada per esempio un edificio.

Adesso si deve determinare l'intersezione tra tutti i segmenti. Per calcolare l'intersezione tra due segmenti si calcola per ogni segmento la retta passante per esso e poi si calcola l'intersezione tra esse. Si scartano i punto di intersezione che escono dall'immagine. Per ogni intersezione si calcola anche un peso. Il peso é dato dal delle due lunghezze dei segmenti elevate al quadrato

$$weight = (seg1.width * seg2.width)^2$$

Figura 3.4: In nero si vedono i segmenti rilevati e filtrati

Questo per assegnare un importanza maggiore all' intersezione di due segmenti piú lunghi. Il Vanishing Point temporaneo é calcolato come la media

Figura 3.5: In bianco le intersezioni tra segmenti calcolate

pesata dei punti di intersezione calcolati in precedenza.

$$\sum_{i \in intersezioni} i * \frac{i.weigth}{sommaWeigth}$$

Infine per calcolare il Vanishing Point effettivo si fa la media con il Vanishing Point calcolato nell'immagine precedente.

3.2.4 Implementazione

L'algoritmo per il calcolo del punto sull'orizzonte viene chiamato prima di fare qualsiasi operazione sull' immagine. L'algoritmo una volta calcolato il punto sull'orizzonte controlla che essso sia nell'immagine perché il punto sull'orizzonte potrebbe essere anche esterno ad essa. Se questo é esterno non si prende in considerazione altrimenti si taglia l'immagine nel suo intorno. Il taglio viene eseguito tenendo una percentuale di pixel rispetto alla larghezza

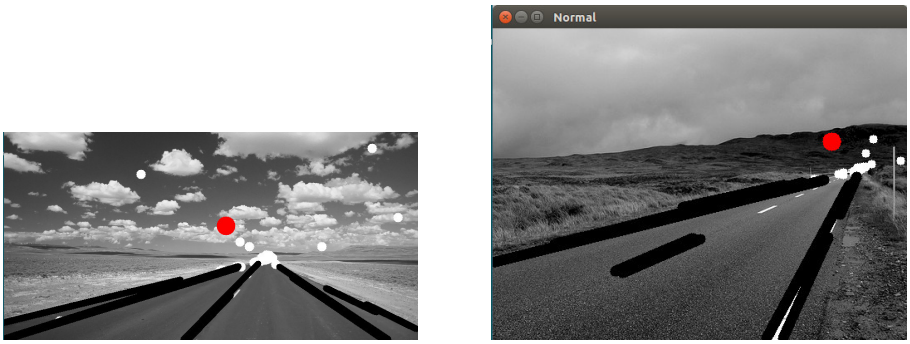

Figura 3.6: In rosso i Vanishing Point calcolati

dell' immagine. Questa percentuale é fissata al 60% rispetto alla larghezza e anche all'altezza. Siccome il punto sull'orizzonte é quasi sempre calcolato nella parte alta come nella figura 3.2 la parte tagliata in altezza sará quasi sempre minore al 60% riducendo cosí la superfice da analizzare successivamente aumentando le prestazioni.
Il calcolo del vanishing Point come vedremo nella prossima sezione ha un costo computazionale parecchio elevato di circa un secondo quindi é stato scelto di calcolarlo solo sulla prima immagine quando si arriva in rettilineo. Successivamente se si trova il cartello nell'intorno del punto sull'orizzonte allora si passa alla prossima immagine e su di esso non si ricalcola tenendo quello precedente, altrimenti si cerca il cartello in tutta l'immagine e si cercherá di ricalcolare il vanishing point nel prossimo frame.

3.2.5 Analisi prestazioni

Adesso passiamo ad analizzare le prestazioni dell'algoritmo per calcolare il punto sull'orizzonte.
Per fare questi test sono state utilizzate delle immagini prese con il robot in movimento una di seguito all' altra partendo da una distanza dal cartello di 6 metri. La grandezza dei segnali é pari al lato corto di un foglio A4. Purtroppo le immagini sono state prese in laboratorio e non all'aperto su una strada peró il vanishing Point viene calcolato correttamente anche aiutato dalle righe del pavimento che sono parallele alla direzione del robot. Sono state fatte 9 prove con il robot in movimento a diverse velocitá. Prima é stato fatto cercare il cartello usando l'algoritmo spiegato precedentemente sul vanishing Point e poi senza calcolarlo. Quelo che é emerso é riportato nel grafico seguente. Grafico. Le velocitá di esecuzione diminuisce del TOT % se si é giá calcolato il punto sull'orizzonte. Invece se deve essere ancora calcolato il tempo di esecuzine aumenta parecchio. Infatti il tempo per calcolare il punto

sull'orizzonte é di circa 1 secondo e quindi pesante per la nostra esecuzione dell'algoritmo. Per questo é stato scelto, come detto precedentemente, di calcolarlo sulla prima immagine e non su tutte a meno che il cartello non sia stato trovato allora si ricalcola.

3.3 Distanza dal cartello

Dopo aver riconosciuto il cartello per essere di aiuto a sapere il posizionamento del robot nel percorso bisogna calcolare la distanza da essi. Di seguito si implementa un algoritmo per calcolare la distanza dal cartello individuato in precedenza.

3.3.1 Analisi ambiente

I passi precedenti dell' algoritmo individuano il cartello restituendo il i punti dell' rettangolo che lo contengono. Sapendo la grandezza dell' cartello a priori, si potrebbe ricavare la larghezza di esso e poi in funzione di essa si puó sapere la distanza. Il problema peró é sempre la risoluzione della telecamera. Infatti dopo aver preso delle immagini sperimentali di cartelli posti a 2 e 3 metri si sono analizzate le larghezze in pixel dei due cartelli e si é notato che essi differivano di un pixel o addirittura in certi casi erano identiche. Un idverso approccio sfrutta IPM (Inverse Perspective Mapping)

3.3.2

Capitolo 4

Esperimenti

Capitolo 5

Conclusioni

Bibliografia

[1] Fok Jr. My article, 2006.