UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXI Ciclo

SISTEMI DI PERCEZIONE FRONTALE PER VEICOLI INTELLIGENTI MEDIANTE VISIONE ARTIFICIALE

Coordinatore:

Chiar.mo Prof. Carlo Morandi

Tutor:

Chiar.mo Prof. Alberto Broggi

Dottorando: *Paolo Zani*

Gennaio 2009

...non bisogna aver paura delle strade,
perchè in fondo sono solo questo – strade.
Non importa chi va e chi viene, il passare delle stagioni,
e cosa succede in città:
quel che conta è avere ancora voglia di mettersi in viaggio, e seguirle.

Sommario

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Sicurezza stradale e percezione dell'ambiente

Ogni giorno strade e autostrade sono teatro di incidenti, spesso gravi, talvolta mortali, con costi sociali notevoli.

Uno studio del 2006 [21] quantifica rispettivamente in 1.018.200€, 143.100€ e 23.100€ l'impatto economico di una morte, un ferimento grave ed uno lieve; se si considera che sul territorio dell'Unione Europea le statistiche [3, 24] parlano di 42.500 morti e 1.705.319 feriti in 1.275.867 incidenti nel solo 2007 si può ben comprendere la recente proposta, da parte della Commissione Europea [19] di richiedere l'introduzione di sistemi di sicurezza avanzati sui veicoli in circolazione, in linea con l'obiettivo di dimezzare in dieci anni, a partire dal 2000, il numero di decessi registrati. In Fig. 1.1 sono rappresentati i dati relativi agli incidenti (mortali) nel periodo 1990-2007: come si può notare, l'andamento è positivo, con una riduzione dei sinistri che, però, non risulta ancora sufficiente a raggiungere l'obiettivo prefissato in tempo utile.

Per soddisfare questa esigenza di una sempre maggiore sicurezza dei veicoli si sono studiati, nel corso degli anni, sistemi di diversa natura, che possono essere classificati in due categorie principali: la prima comprende quelli realizzati con lo scopo di ridurre i danni prodotti da un incidente (come cinture di sicurezza ed airbag), men-

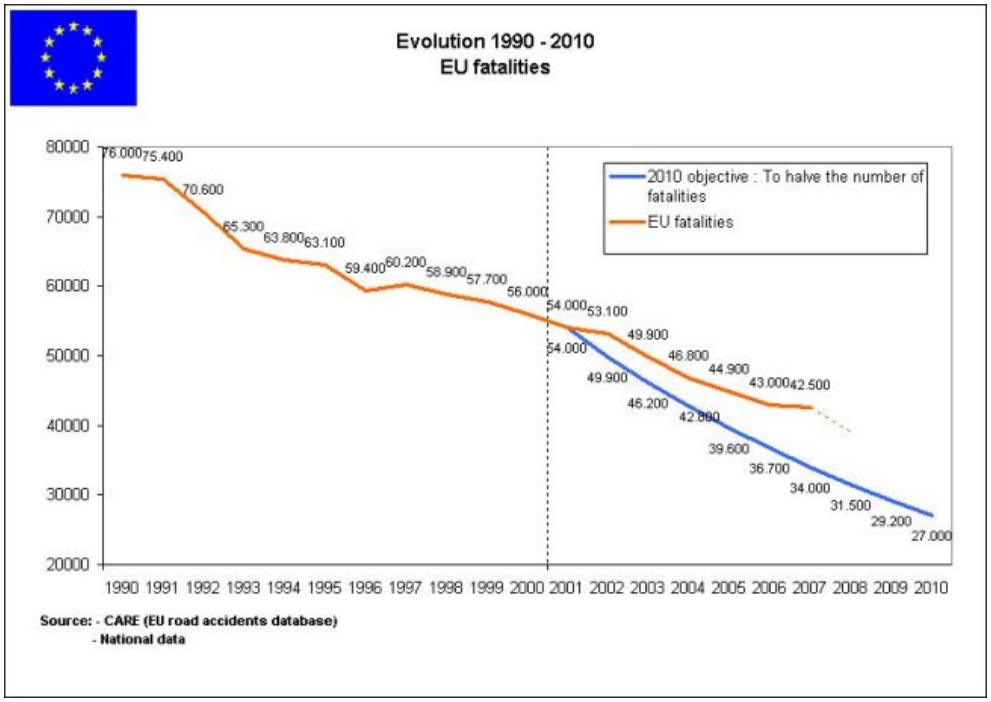


Figura 1.1: Evoluzione dei sinistri sul territorio europeo, secondo i dati CARE [3].

tre la seconda quelli che assistono il conducente durante la guida, aiutandolo a gestire al meglio il mezzo nelle situazioni di pericolo (come ABS e ESP), o, meglio ancora, a prevenirle.

L'ultima classe di sistemi di sicurezza si distingue per il modo in cui opera: riconoscere l'insorgere di una situazione di rischio in modo tempestivo, per prevenirne o ridurne le conseguenze, significa avere la capacità di percepire l'ambiente circostante, e possedere l'"intelligenza" necessaria per decidere quando sia il caso di intervenire. In questo contesto si inquadrano progetti come l'"Intelligent Car Initiative"[1], promossa dall'Unione Europea con lo scopo di accelerare l'introduzione di questi sistemi nel mercato europeo ed internazionale con il ricorso a strumenti politici, di

[1] http://ec.europa.eu/information_society/activities/intelligentcar/index_it.htm

ricerca e di comunicazione, e PReVENT[2], progetto durato quattro anni, e conclusosi con una dimostrazione a Versailles il 18-22 settembre 2007, che ha avuto lo scopo di sviluppare, integrare in un'unica piattaforma e dimostrare un insieme di sistemi di sicurezza per assistere e proteggere sia i guidatori che gli altri utenti della strada [33].

La ricerca comunque si è spinta oltre, cercando di produrre veicoli del tutto autonomi, in grado, cioè, di navigare nell'ambiente senza la necessaria presenza di un autista a controllarli. I primi esperimenti in questo campo risalgono alla fine degli anni settanta, seguiti negli anni novanta da un periodo di grande interesse per l'argomento; la complessità del problema, e i limiti dei calcolatori elettronici dell'epoca però hanno rallentato i progressi in questo ambito fino al 2004, anno della prima *DARPA Grand Challenge*[3], gara in cui veicoli autonomi hanno gareggiato su un percorso tracciato nel deserto del Mojave, tra Los Angeles e Las Vegas. A questa è seguita una seconda edizione dell'evento, nel 2005, in cui cinque dei ventitré veicoli finalisti hanno raggiunto il traguardo (in Fig. 1.2, i *team* schierati all'arrivo). Nel 2007, con la *DARPA Urban Challenge*, la competizione si è spostata in un ambiente cittadino simulato, dove i mezzi hanno dovuto dimostrare la loro capacità sia di rispettare il codice della strada, sia di adattarsi alle varie condizioni di traffico, anche intenso, affrontando incroci con precedenza, immissioni nel traffico, sorpassi, inversioni e parcheggi; sei delle undici squadre finaliste hanno completato il percorso.

Nonostante i tempi non sembrino ancora maturi per l'introduzione su larga scala di veicoli senza conducente, sia per le implicazioni legali che per quelle sociali che una scelta del genere porterebbe con sé, queste tecnologie hanno dimostrato di essere mature a sufficienza per l'impiego in ambiti specifici, in cui poter fare a meno della presenza di un operatore umano a bordo del mezzo risulta un notevole vantaggio: ambienti ostili, situazioni pericolose per l'uomo e compiti ripetitivi possono essere ora affrontati con strumenti più adeguati.

Questa breve panoramica sul tema della tecnologia applicata alla sicurezza dei veicoli mette in luce la presenza di due grandi filoni di attività: il primo consiste nella realizzazione di sistemi che vadano ad intervenire in modo mirato sui vari fattori di

[2] http://www.prevent-ip.org
[3] http://www.darpa.mil/grandchallenge

Figura 1.2: I *team* che hanno concluso con successo la *DARPA Grand Challenge 2005*, schierati sul traguardo.

rischio legati all'utilizzo dei veicoli, da integrare su mezzi destinati ad un pubblico il più ampio possibile, mentre il secondo cerca di produrre prototipi in grado di portare a termine il loro compito senza supervisione in scenari complessi, dinamici e di cui si ha una conoscenza a priori limitata o, talvolta, anche nulla.

Tenendo conto della suddivisione appena esposta, questa tesi presenterà due sistemi molto diversi tra loro, sia per il tipo di applicazione per cui sono stati concepiti, sia per le soluzioni *hardware* e *software* che li caratterizzano, ma accomunati dallo stesso obiettivo, quello di rendere il veicolo su cui sono montati "conscio" di alcune delle caratteristiche dell'ambiente che lo circonda.

Nella parte I verrà presentato il sistema di percezione frontale degli ostacoli rea-

lizzato per il veicolo autonomo TerraMax[4], uno dei finalisti della *DARPA Urban Challenge 2007*, visibile in Fig. 1.3.

Figura 1.3: Il veicolo autonomo TerraMax.

Nella parte II ci si occuperà di un dispositivo di *Lane Detection* basato su una piattaforma DSP[5] *embedded* (Fig. 1.4), pensato per essere installato con facilità a bordo di un'automobile, e in grado di rilevare in modo accurato la geometria della segnaletica orizzontale presente di fronte al veicolo.

[4] http://www.terramax.com

[5] acronimo di *Digital Signal Processor*, un genere di microprocessore specificamente progettato per l'elaborazione numerica dei segnali, e spesso impiegato in applicazioni *real-time*

Figura 1.4: Dispositivo *embedded* per il rilevamento della segnaletica orizzontale.

Parte I

Rilevamento frontale di ostacoli: il veicolo autonomo TerraMax

Capitolo 2

I sensori a bordo del TerraMax

Un veicolo autonomo, per poter operare, ha bisogno di percepire l'ambiente circostante, interpretare la situazione e reagire in modo appropriato, traducendo poi il comportamento pianificato in comandi per gli attuatori: in sostanza, deve avere capacità di percezione, decisione e controllo.

Per quanto riguarda il controllo, la tecnologia ha raggiunto un livello di sviluppo molto avanzato, con sistemi come ABS, EBD ed ESP sul mercato da tempo, e ormai di serie su molti mezzi. Percezione e decisione, d'altro canto, risultano problemi più difficili da risolvere in modo affidabile: spesso ci si trova a fare i conti con un gran numero di situazioni differenti, e scenari sconosciuti. Quando ci si occupa di percezione si deve tener conto che gli elementi presenti devono essere rilevati e classificati, e un ambiente complesso come quello urbano –costellato dagli oggetti più disparati– può portare ad errori ed introdurre un alto livello di rumore sui risultati. Un discorso simile si applica al sistema di decisione, che deve gestire non solo le dinamiche interne al veicolo (come invece accade per la parte di controllo), ma anche le numerose situazioni che si vengono a creare durante l'interazione con tutto ciò che lo circonda.

Nonostante sia un prototipo, il TerraMax è stato progettato utilizzando tecnologie consolidate, con lo scopo di facilitare la successiva realizzazione di un sistema di serie da integrare su veicoli di nuova produzione, o di renderlo disponibile come pacchetto di aggiornamento di mezzi esistenti; il TerraMax, inoltre, è in grado di

operare anche utilizzando soltanto sensori passivi, e questa scelta è stata determinata sia dall'ambito militare a cui è rivolto, sia dalla volontà di ridurre al minimo la possibilità di interferenze nel caso di gruppi di veicoli in funzione a stretto contatto tra di loro.

La configurazione finale prevede l'impiego della visione artificiale come sensore principale, con undici telecamere (in rosso in Fig. 2.1) che forniscono una copertura a 360° dell'area circostante il veicolo. Questo *setup* consente di ottenere tutte le informazioni rilevanti per il sistema di pianificazione, come la presenza di ostacoli, altri mezzi in movimento, linee di demarcazione delle corsie e segnaletica orizzontale in genere. Tre *laserscanner* (in verde in Fig. 2.1), di cui due posizionati nel lato frontale del veicolo, ed uno sul retro, completano il comparto dedicato alla percezione, ed oltre a fornire al livello superiore le proprie informazioni elaborate, mettono a disposizione dei vari sistemi di visione i dati grezzi acquisiti, da fondere con quelli provenienti dalle telecamere per migliorare, sotto diversi aspetti, la qualità dei risultati finali. La localizzazione è garantita sia dalla presenza di un sistema GPS (in blu in Fig. 2.1) che dalla disponibilità dei dati odometrici del veicolo.

Il sistema trinoculare (in arancione in Fig. 2.2) rileva le caratteristiche dell'area antistante il veicolo, tra cui le linee di demarcazione delle corsie, i bordi della carreggiata e possibili ostacoli presenti lungo la traiettoria da percorrere. Due sistemi stereo identici (in viola in Fig. 2.2) controllano le immediate vicinanze del fronte e del retro del veicolo, per rilevare in modo preciso la disposizione di ostacoli e segnaletica orizzontale, cosa che si rende necessaria a causa delle dimensioni non indifferenti del TerraMax. Due sistemi monoculari (in verde in Fig.2.2) sorvegliano i lati del mezzo quando questo si avvicina agli incroci; di norma spente, queste telecamere entrano in funzione non appena ci si ferma, e controllano il traffico proveniente da tutte le direzioni, in modo che il sistema di pianificazione possa effettuare le manovre di immissione in sicurezza. Altri due sistemi monoculari (in azzurro in Fig.2.2) tengono infine sotto controllo il traffico nelle corsie adiacenti; i loro risultati vengono utilizzati nel momento in cui il TerraMax si appresta a compiere una manovra di sorpasso.

I sistemi di visione condividono la stessa piattaforma *hardware*, costituita da un

Figura 2.1: I sensori del TerraMax: in rosso –le telecamere; in verde –i *laserscanner*; in blu –le antenne GPS.

PC *rugged* prodotto da SmallPC[1], equipaggiato con una scheda madre in formato mini-ITX della Mini-Box.com[2] con chipset Intel[3] 945GM, un processore Intel Core 2 Duo T2500, 2 GB di RAM ed un disco Compact Flash da 4 GB su cui è installato il sistema operativo, scelto per evitare la presenza di parti mobili, soggette a guasti

[1] http://www.smallpc.com
[2] http://www.mini-box.com
[3] http://www.intel.com

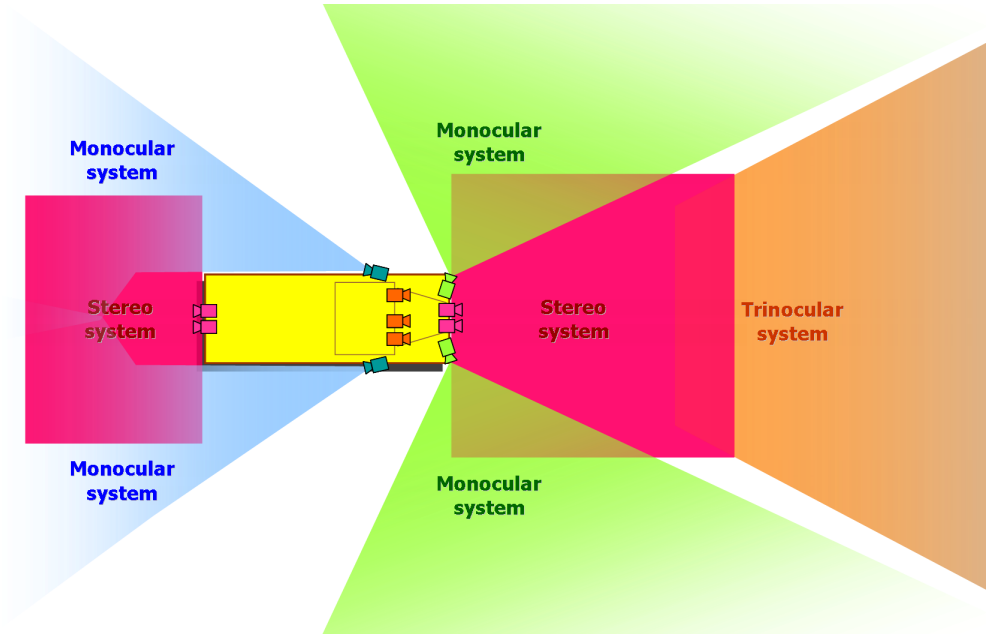
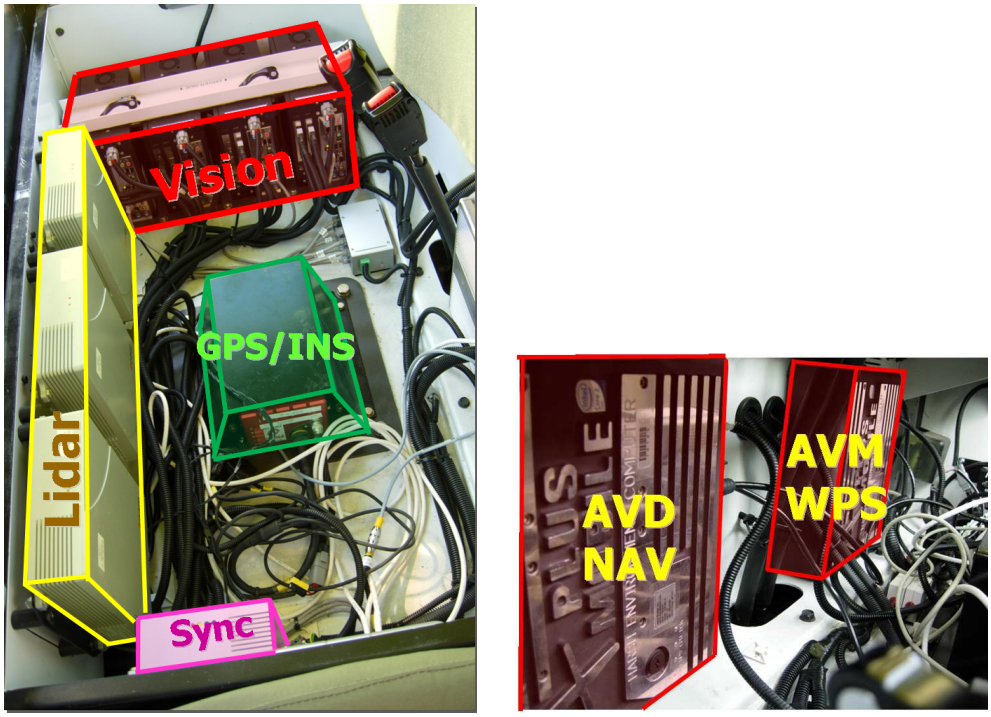


Figura 2.2: Disposizione e area d'interesse dei vari sistemi di visione installati a bordo del TerraMax: in arancione –il sistema trinoculare; in viola –i sistemi stereo; in verde –il sistema laterale; in azzurro –il sistema retrovisore.

in presenza di continue sollecitazioni meccaniche. Due schede FireWire 800, di cui una PCI ed una PCI Express, mettono a disposizione 4 porte a cui connettere un sottoinsieme delle 11 telecamere a disposizione (vedi Tab. 2.1); filtri polarizzatori posizionati di fronte a ciascun obbiettivo riducono i riflessi e migliorano il contrasto delle immagini. Tutti i PC sono alloggiati all'interno dell'abitacolo, sotto il sedile dei passeggeri, assieme alle *Electronic Control Unit* dei *laserscanner*, all'INS/GPS, e al dispositivo che garantisce l'acquisizione sincronizzata delle immagini e dei dati provenienti dai *laserscanner* ad una frequenza di 12.5 Hz (Fig. 2.3-a); in questo stesso spazio trovano anche posto sia i sistemi di pianificazione –AVM e WPS, che quelli che governano gli attuatori –AVD e NAV (Fig. 2.3-b).

Tabella 2.1: Panoramica sui sottosistemi

	TRINOCULAR	**STEREO**	**LATERAL**	**REARVIEW**
Telecamere	3@1024×768	4@1024×768	2@1920×1080	2@1024×768
Posizione	Nella parte alta del parabrezza, all'interno della cabina	Due davanti e due dietro, tutte rivolte verso il basso	Davanti, rivolte ai lati	Sopra la cabina, all'esterno, rivolte all'indietro e verso il basso, ruotate di 90°
Ottiche	4.16 mm, f2.16, FoV: 60°×42°	1.8 mm, f1.4, FoV: 154°×115°	8 mm, f1.4, FoV: 80°×63°	4.16 mm, f2.16, FoV: 60°×42°
Laser	Fronte	Fronte, Retro	N/A	Retro
Algoritmi	*Lane detection*, *stereo obstacle detection*	*Lane detection*, *stopline detection*, *curb detection*, *short-range stereo obstacle detection*	*Moving traffic detection*	*Overtaking vehicles detection*
Copertura	7 → 40 m	0 →10 m, -6.5 → -16.5 m	10 → 130 m	-4 → -50 m
Note	3 sistemi stereo con *baseline*: 1.156 m, 0.572 m, 1.728 m	2 sistemi stereo (fronte e retro)	Abilitato quando il veicolo è fermo agli incroci	Controlla le corsie adiacenti prima di un sorpasso



(a) (b)

Figura 2.3: (a) Le unità di controllo dei sensori del TerraMax: in rosso –il sistema di visione; in giallo –i *laserscanner*; in verde –GPS/INS; in viola –il dispositivo *hardware* per sincronizzare l'acquisizione delle immagini da parte delle telecamere con la scansione del *laserscanner*. (b) Il sistema di pianificazione: l'unità AVD / NAV –*Autonomous Vehicle Driver* / *Navigation* e l'unità AVM / WPS –*Autonomous Vehicle Manager* / *World Perception Server*

Capitolo 3

Il sistema trinoculare

Il sistema trinoculare è stato progettato per garantire la capacità di rilevare in modo robusto sia la presenza di ostacoli che la segnaletica orizzontale nella zona compresa tra i 7 e i 50 m (dove l'estremo inferiore è determinato dalla presenza del cofano motore nella zona inquadrata dalle telecamere). L'algoritmo di rilevamento ostacoli permette di gestire situazioni che con l'utilizzo dei soli *laserscanner* risulterebbero difficili –se non impossibili– da affrontare, come il riconoscimento del numero di veicoli che precedono in colonna agli incroci (informazione utile per il rispetto delle regole sulla precedenza vigenti negli USA), della presenza di paletti e delle nuvole di polvere che si vengono a creare nelle zone non asfaltate (altrimenti spesso percepite come oggetti solidi); fornisce inoltre informazioni aggiuntive sugli elementi individuati (come altezza e colore), e costituisce una soluzione affidabile anche in caso di rottura dei *laserscanner*. Gli ostacoli individuati sono infine utilizzati per migliorare le prestazioni del modulo di *lane detection*, attraverso una mascheratura delle regioni dell'immagine corrispondenti. L'algoritmo di rilevamento delle linee di demarcazione delle corsie fornisce al *path planner* una descrizione dettagliata della geometria della strada, assieme ad una classificazione di ciascuna linea rilevata (come il tipo e il colore).

Le tre telecamere (delle Point Grey Research[1] Flea2 con sensore Sony da 1/3 " e

[1] http://www.ptgrey.com

risoluzione 1024×768 *pixel*) sono installate nella parte alta del parabrezza, rivolte in avanti, e sono fissate al telaio con dei supporti Huber[2], che permettono di correggere gli angoli di *pitch* e *yaw*, e forniscono viti di fissaggio per prevenire movimenti dovuti alle vibrazioni. Le lenti Myutron[3] da 4.2 mm rappresentano un buon compromesso tra un ampio *field of view* ed una buona risoluzione a grande distanza.

Per fornire risultati accurati è necessaria una adeguata calibrazione e, data la dimensione delle *baseline* in gioco (fino a 1.5 m), questo requisito diventa piuttosto complesso da soddisfare. Una griglia di calibrazione di 5×25 m, distesa in una zona pianeggiante come mostrato in Fig. 3.1 viene utilizzata per stimare i parametri estrinseci, poi raffinati utilizzando i dati laser a disposizione. Il posizionamento delle telecamere è reso più semplice dall'utilizzo di una rettificazione via *software* delle immagini acquisite, che compensa piccoli disallineamenti; in ogni caso, eseguire tutte le misurazioni necessarie, ed estrarre i valori corretti dai dati raccolti è un'attività affatto banale e dispendiosa in termini di tempo, che va eseguita ogni volta che si interviene sulle telecamere, e dunque da pianificare in modo adeguato, per evitare di rendere il sistema inutilizzabile troppo a lungo.

La configurazione asimmetrica secondo cui sono installate le telecamere, già sperimentata con successo nel corso delle edizioni della *DARPA Grand Challenge* [13], dà origine a tre possibili *baseline*: questa soluzione fornisce un decisivo grado di libertà, che permette di soddisfare le diverse esigenze che insorgono durante la navigazione autonoma.

L'elaborazione ha inizio con la conversione dei dai grezzi (codificati secondo il pattern di Bayer) acquisiti in immagini monocromatiche sotto-campionate; queste immagini sono poi rettificate in vista della successiva elaborazione stereo, correggendo le inevitabili differenze di orientazione delle telecamere, in modo da rendere le rispettive rette epipolari orizzontali, e permettere misure più precise. Uno speciale filtro derivativo [12] –utile per evidenziare le *feature* del terreno– viene quindi applicato alle immagini provenienti dalla *baseline* maggiore, e il suo risultato viene utilizzato per calcolare la cosiddetta *V-Disparity Image* [27]. Questa immagine vie-

[2] http://www.hubermounts.com
[3] http://www.myutron.com/index_e.asp

Figura 3.1: La griglia di calibrazione del sistema trinoculare.

ne quindi analizzata per stimare il beccheggio istantaneo del veicolo, come descritto in [13].

Il passo successivo consiste nel selezionare una *baseline* tra le tre disponibili per generare una *Disparity Space Image*: baseline più ridotte garantiscono risultati meno rumorosi, che sono utili durante le manovre a bassa velocità, mentre baseline più lunghe permettono di avere una zona di percezione più ampia, in modo da rilevare gli ostacoli mentre questi sono ancora distanti dal veicolo.

Il calcolo della *Disparity Space Image* viene effettuato utilizzando un approccio basato su confronto locale, con una SAD (*Sum of Absolute Differences*) come funzione di costo relativa, il che permette di avere implementazioni molto efficienti; il calcolo viene eseguito in modo incrementale, risultando così indipendente dalla dimensione delle finestre [23]. L'algoritmo è stato migliorato ricercando le corrispondenze lungo le rette epipolari solo per valori di disparità maggiori di quello previsto per il terreno (che cambia lungo le righe dell'immagine) : questo approccio miglio-

ra la qualità della mappa risultante, riducendo il numero di *match* spurii, ma risulta piuttosto complesso da seguire, dato che genera un elevato numero di effetti di bordo di cui bisogna tener conto.

Sia la *V-Disparity Image* che la *Disparity Space Image* sono calcolate sfruttando i diversi gradi di parallelismo offerti dall'*hardware* a disposizione: le immagini vengono suddivise in strisce orizzontali ed elaborate in parallelo su più *thread* indipendenti tra loro, in modo da distribuire il carico computazionale su entrambi i processori presenti; l'algoritmo di confronto tra finestre, inoltre, sfrutta le capacità di calcolo SIMD[4] disponibili mediante l'utilizzo dei *set* di istruzioni MMX ed SSE, operando allo stesso tempo su più *pixel* e righe (*V-Disparity Image*) o più valore di disparità (*Disparity Space Image*).

Una volta completato il calcolo dell'immagine di disparità, e delle coordinate dei punti del mondo corrispondenti, si procede col rilevamento degli ostacoli. Il primo passo consiste nel rimuovere i punti appartenenti al terreno, poi si passa a raggruppare, colonna per colonna, i punti con disparità simile. Uno dei limiti dell'algoritmo presentato in [16] –realizzato per la *DARPA Grand Challenge 2005* ed utilizzato come punto di partenza per lo sviluppo– era la sua incapacità di gestire insiemi di ostacoli allineati lungo la stessa colonna dell'immagine, rilevando solo quello di dimensioni maggiori; la nuova versione, invece, rimuove questa limitazione, permettendo, ad esempio, di riconoscere in modo corretto gruppi di macchine in fila ad un incrocio, incolonnate nell'immagine per effetto prospettico. La fase successiva consiste nel valutare la compattezza di ogni gruppo, in modo da eliminare gli elementi troppo dispersi; i candidati rimanenti ricevono quindi un punteggio proporzionale al numero di *pixel* che li compone, e questo valore viene aumentato nel caso ci siano colonne adiacenti con disparità simile. Il valore risultante viene quindi moltiplicato per la distanza media del gruppo dalla telecamera, in modo da avere una metrica uniforme tra i vari gruppi, ed infine una soglia determina quali elementi costituiscano reali ostacoli per il TerraMax.

Un sistema di fusione tra le immagini e i dati provenienti dai *laserscanner* fron-

[4] *Single Instruction Multiple Data*: modalità di elaborazione vettoriale in cui una stessa istruzione viene eseguita in simultanea su un insieme di dati distinti

tali migliora le prestazioni in aree prive di *texture*: i punti laser vengono aggiunti a quelli corrispondenti agli ostacoli, andando a formare l'insieme di semi per la procedura finale di espansione. Va notato che nonostante provengano da sensori del tutto differenti, i semi possono essere fusi assieme in quanto a valle dei filtraggi cui vengono sottoposti forniscono lo stesso tipo di informazione: un numero limitato di punti corrispondenti con alto grado di affidabilità ad ostacoli.

Un componente dedicato al *tracking* assegna infine ad ogni elemento rilevato un identificativo, età e velocità di movimento.

Capitolo 4

L'algoritmo per il calcolo della *Disparity Space Image*

Quando da una coppia di immagini stereoscopiche si vuole estrarre una mappa di disparità, sono possibili un notevole numero di approcci differenti, ognuno con le proprie possibilità di ottimizzazione e i propri limiti. Nel caso particolare del TerraMax, come si è visto nel capitolo 3, i vincoli sui tempi di esecuzione orientano la scelta su un algoritmo che operi confronti locali, mentre la disponibilità delle informazioni sull'andamento del terreno consente di limitare il campo di ricerca, offrendo l'opportunità di fornire risultati più affidabili. L'affidabilità dei valori di disparità restituiti è ulteriormente migliorata dall'applicazione di un insieme di filtri, che hanno lo scopo di riconoscere ed eliminare gli abbinamenti che con buona probabilità sono da considerare errati.

Ad alto livello, l'algoritmo che si viene dunque a delineare ha la struttura riportata nell'Algo. 1

4.1 Calcolo incrementale

Quel che appare subito evidente è che, pur ottenendo i risultati corretti, se si procede in questo modo la maggior parte dei calcoli viene ripetuta più volte inutilmente, con

Algorithm 1: Algoritmo per il calcolo della DSI

```
for i ← 0 to height do
    for j ← 0 to width do
        best_score ← ∞;
        best_d ← NaN;
        norm ← win_w ∗ win_h ∗ |∑(D, i, j, win_w, win_h)|
        for d ← d_min[i] to d_max[i] do
            sad ← SAD (S, D, i, j, win_w, win_h) ;
            score ← sad/norm;
            if score[d] < best_score then
                best_score ← score;
                best_d ← d;
            end
        end
        if ApplyFilters() successful then
            DSI[i][j] ← best_d;
        else
            DSI[i][j] ← UNKOWN;
        end
    end
end
```

un decadimento delle prestazioni che si fa sempre più sensibile man mano che si aumentano le dimensioni delle finestre di confronto; utilizzando un procedimento incrementale [23] è però possibile rimuovere questa limitazione.

Detta $SAD(x,y,d)$ il valore ottenuto confrontando due finestre di dimensioni $(2m+1)\times(2n+1)$ centrate alle coordinate (x,y) nell'immagine destra D, e $(x+d,y)$ in quella sinistra S, si ha che:

$$SAD(x,y,d) = \sum_{i=-m}^{m} \sum_{j=-n}^{n} |D(x+j,y+i) - S(x+j+d,y+i)| \tag{4.1}$$

$SAD(x,y,d)$ può però essere ottenuta da $SAD(x,y-1,d)$ in questo modo:

$$SAD(x,y,d) = SAD(x,y-1,d) + \Delta_r(x,y,d) \tag{4.2}$$

con

$$\begin{aligned}\Delta_r(x,y,d) = \\ \sum_{j=-n}^{n} |D(x+j,y+m) - S(x+j+d,y+m)| - \\ \sum_{j=-n}^{n} |D(x+j,y-m-1) - S(x+j+d,y-m-1)|\end{aligned} \tag{4.3}$$

inoltre $\Delta_r(x,y,d)$ può essere ricavato da $\Delta_r(x-1,y,d)$:

$$\begin{aligned}\Delta_r(x,y,d) = \\ \Delta_r(x-1,y,d) + \\ |D(x-n-1,y-m-1) - S(x-n-1+d,y-m-1)| - \\ |D(x-n-1,y+m) - S(x-n-1+d,y+m)| - \\ |D(x+n,y-m-1) - S(x+n+d,y-m-1)| + \\ |D(x+n,y+m) - S(x+n+d,y+m)|\end{aligned} \tag{4.4}$$

A questo punto il calcolo di $SAD(x,y,d)$ per una generica finestra all'interno dell'immagine viene effettuato con un numero di operazioni che non dipende più dal valore di m ed n. Se si riscrive l'Eq. 4.4 come

$$\Delta_r(x,y,d) = \Delta_r(x-1,y,d) - A + B \tag{4.5}$$

con

$$\begin{aligned} A &= |D(x-n-1,y+m) - S(x-n-1+d,y+m)| - \\ &\quad |D(x-n-1,y-m-1) - S(x-n-1+d,y-m-1)| \end{aligned} \tag{4.6}$$

$$\begin{aligned} B &= |D(x+n,y+m) - S(x+n+d,y+m)| - \\ &\quad |D(x+n,y-m-1) - S(x+n+d,y-m-1)| \end{aligned} \tag{4.7}$$

risulta più evidente che i termini A e B –che rappresentano rispettivamente le coppie di *pixel* negli angoli di sinistra e di destra della finestra in esame– hanno un ruolo intercambiabile, a meno di un *offset* di $2m+1$ elementi. Questo fatto permette di evitare il calcolo diretto di entrambi i termini, a patto di utilizzare una matrice d'appoggio Δ_p di dimensione $m \times max\_disp$, con $max\_disp = \max(search\_ranges(y_1,1)) - \min(search\_ranges(y_0,0)) \forall y_0, y_1 \in [0, height[$, dove di volta in volta si va ad estrarre dalla posizione $(x \mod (2m+1), d)$ il valore da utilizzare per A, per poi sostituirlo con quello calcolato per B; l'Eq. 4.4 si può dunque riscrivere come

$$\begin{aligned}\Delta_r(x,y,d) = \\ &\Delta_r(x-1,y,d) - \Delta_p(\bar{x},d) + \\ &|D(x+n,y+m) - S(x+n+d,y+m)| - \\ &|D(x+n,y-m-1) - S(x+n+d,y-m-1)|\end{aligned} \tag{4.8}$$

con $\bar{x} = x \mod (2n+1)$, e l'Eq. 4.2 si riduce a:

$$\begin{aligned}SAD(x,y,d) = \\ &SAD(x,y-1,d) + \Delta_r(x-1,y,d) - \Delta_p(\bar{x},d) + \\ &|D(x+n,y+m) - S(x+n+d,y+m)| - \\ &|D(x+n,y-m-1) - S(x+n+d,y-m-1)|\end{aligned} \tag{4.9}$$

In base a quanto esposto finora, la procedura impiegata nel calcolo della *Disparity Space Image* prevede che durante l'elaborazione si debbano gestire quattro differenti categorie di punti:

- il primo punto della prima riga, per cui i valori di correlazione si calcolano secondo l'Eq. 4.1;
- i punti successivi della prima riga, per cui vale la seguente relazione:

$$\begin{aligned}SAD(x,y,d) = SAD(x-1,y,d) + \\ &\sum_{i=-m}^{m} |D(x+n,y+i) - S(x+n+d,y+i)| - \\ &\sum_{i=-m}^{m} |D(x-n-1,y+i) - S(x-n-1+d,y+i)|\end{aligned} \tag{4.10}$$

- il primo punto di una generica riga all'interno dell'immagine, per cui vale l'Eq. 4.2;
- un generico punto all'interno dell'immagine, per cui vale l'Eq. 4.9.

A questo punto si può affrontare il problema di limitare la dimensione degli intervalli di ricerca del miglior abbinamento possibile ad un determinato insieme di valori di disparità, anche negativi, per meglio far fronte alle esigenze di un sistema come quello del TerraMax.

Si definisce una matrice *search_ranges* di dimensione $height \times 2$, imponendo che

$$search\_ranges(y,0) \geq search\_ranges(y-1,0) \, \forall y \in [\,1, height\,[\tag{4.11}$$

in modo da sfruttare la disparità del terreno –che tende a decrescere con la distanza– come punto di partenza per la ricerca; questa scelta ha inoltre il vantaggio di ridurre il numero di casi particolari da gestire.

Per prima cosa, va notato che gli intervalli di ricerca si riducono quando la finestra di riferimento si trova in prossimità dei bordi sinistro e destro dell'immagine:

$$\begin{cases} d_{min}(x,y) = \max(n-x, search\_ranges(y,0)) \\ d_{max}(x,y) = \min((width-n-x, search\_ranges(y,1)) \end{cases} \tag{4.12}$$

dal momento che non è stato imposto alcun vincolo sulle caratteristiche dei valori assunti da $search\_ranges(y,1)$, può accadere che

$$search\_ranges(y,1) < search\_ranges(y+1,1)$$

In questo caso non è possibile calcolare in modo incrementale (Eq. 4.9) i valori di correlazione per i punti alle coordinate

$$(x, y+1, d) | d \in [\,d_{max}(x,y), d_{max}(x,y+1)\,]$$

in quanto i rispettivi valori di $SAD(x,y,d)$ non vengono calcolati durante il ciclo precedente. Un metodo per ovviare a questo inconveniente è quello di calcolare comunque $SAD(x,y,d)$ anche per $d \in [\,d_{max}(x,y), d_{max}(x,y+1)\,]$, in modo da renderli

disponibili per la riga $y+1$. Nel fare ciò bisogna prestare attenzione al fatto che, a sua volta, anche la riga y dipende dalla riga $y-1$, e che dunque l'Eq. 4.9 potrebbe non essere valida (cosa che accade quando $d_{max}(x,y-1) < d_{max}(x,y+1)$): nel caso in esame si è scelto di utilizzare sempre l'Eq. 4.10, che è indipendente dai valori di $SAD(x,y-1,d)$, senza controllare se i valori di $SAD(x,y,d)$ si possano calcolare in modo del tutto incrementale (ovvero se $d_{max}(x,y-1) >= d_{max}(x,y+1)$), dal momento che all'atto pratico ciò non si verifica mai, in quanto l'utilizzo tipico prevede valori di $d_{max}(x,y)$ crescenti all'aumentare di y; così facendo si è semplificata –senza perdita di generalità– la logica dell'algoritmo, a vantaggio della velocità di esecuzione negli scenari più comuni.

Ammettere valori negativi per $search\_ranges(y,0)$ permette di garantire una certa tolleranza nel caso di errori in eccesso nella stima di questo valore (nel caso del TerraMax ricavato a partire dalla *V-Disparity image* [13], e dunque affetto da rumore), ma introduce un fenomeno di cui bisogna tener conto: se $search\_ranges(y,0) = \bar{d} < 0$ i primi $|\bar{d}|$ punti della riga non possono essere calcolati utilizzando l'Eq. 4.9, dal momento che non è disponibile il termine $\Delta_r(x-1,y,d) \forall x \in [\,1,\,|\bar{d}|\,[$.

Questo accade perché secondo l'Eq. 4.12 si ha che

$$\max(n-x,\bar{d}) = n-x$$

essendo

$$n-x > \bar{d} \forall x \in [\,1,\,|\bar{d}|\,[\text{ con } n>0, \bar{d}<0$$

con la conseguenza che

$$\max(n-x,\bar{d}) > \max(n-x-1,\bar{d}) \forall x \in [\,1,\,|\bar{d}|\,[$$

ovvero

$$d_{min}(x,y) > d_{min}(x-1,y)$$

e per la precisione

$$d_{min}(x,y) = d_{min}(x-1,y)+1$$

Anche in questo caso la soluzione adottata è quella di rinunciare al calcolo incrementale, ricorrendo all'Eq. 4.1.

4.2 Filtraggi

Durante il calcolo dell'immagine di disparità si incontrano alcune regioni che più di altre sono propense a dar luogo ad abbinamenti errati, portando dunque ad una non corretta stima del valore di disparità associato. L'algoritmo per la generazione della *Disparity Space Image* presentato in questa tesi è di tipo "locale", in quanto si basa soltanto su un "intorno" (in questo caso, una finestra $2m+1 \times 2n+1$) del punto in esame, e come tale è molto sensibile alla quantità di informazione presente nella finestra impiegata. Aree prive di *texture* danno risultati scadenti, in quanto i punteggi associati ai diversi valori di disparità risultano molto simili tra loro: in questi casi, è preferibile scartare il risultato piuttosto che rischiare di fornirne uno errato. Una metrica che permette di identificare queste zone è la varianza dei valori della finestra: valori troppo bassi indicano *pixel* con poco contenuto informativo, e dunque da scartare. Nel caso le immagini in ingresso siano state preelaborate con qualche tipo di filtro derivativo è anche possibile impiegare una metrica ancora più semplice: la somma dei valori dei *pixel* appartenenti alla finestra in esame, che, ancora una volta, deve superare una certa soglia per considerare attendibile il risultato. Entrambe le soluzioni presentate possono essere implementate con uno schema incrementale simile a quello visto per il calcolo dei valori di $SAD(x,y,d)$.

Prendendo come definizioni di media e varianza rispettivamente

$$\begin{aligned} \mu(x,y) &= \frac{\sum_{i=-m}^{m}\sum_{j=-n}^{n} D(x+j,y+i)}{(2m+1)\times(2n+1)} \\ &= \frac{1}{N} S_1(x,y) \end{aligned} \tag{4.13}$$

e

$$\begin{aligned} \sigma^2(x,y) &= \frac{\sum_{i=-m}^{m}\sum_{j=-n}^{n} D(x+j,y+i)^2}{(2m+1)\times(2n+1)} - \mu^2(x,y) \\ &= \frac{1}{N} S_2(x,y) - \mu^2(x,y) \end{aligned} \tag{4.14}$$

si ha che

$$\begin{aligned} \mu(x,y) = \frac{1}{N}(&S_1(x,y-1) + \Delta S_{1r}(x-1,y) - \Delta S_{1p}(\bar{x}) + \\ &D(x+n,y+m) - D(x+n,y-m-1)) \end{aligned} \tag{4.15}$$

con $\bar{x} = x \mod (2n+1)$, e

$$\begin{aligned}\sigma^2(x,y) &= \frac{1}{N}(S_2(x,y-1)+\Delta S_{2r}(x-1,y)-\Delta S_{2p}(\bar{x})+ \\ &\quad D(x+n,y+m)^2-D(x+n,y-m-1)^2)- \\ &\quad \mu(x,y)\end{aligned} \tag{4.16}$$

sempre con $\bar{x} = x \mod (2n+1)$. E' interessante notare come i valori di S_1 via via calcolati utilizzando l'Eq. 4.15 possano essere riutilizzati per la normalizzazione dei punteggi ottenuti (Algo. 1).

Per verificare la consistenza dei risultati ottenuti si può introdurre, con un costo computazionale limitato, l'algoritmo proposto in [35] denominato *Single Matching Phase* (SMP), che prevede di scartare l'abbinamento col punteggio più basso quando si rileva che

$$\bar{x}_1 + DSI(\bar{x}_1, y) = \bar{x}_2 + DSI(\bar{x}_2, y)$$

ossia nel caso in cui due finestre di riferimento (immagine destra) risultino mappate sulla stessa finestra di destinazione (immagine sinistra); tale evenienza, di per sé priva di significato fisico, può comunque verificarsi a seguito della presenza di condizioni di illuminazione avverse, rumore o occlusioni. L'analisi della distribuzione dei punteggi ottenuti da una finestra per diversi valori di disparità permette di irrobustire ulteriormente i risultati, andando ad individuare comportamenti che di solito portano ad abbinamenti sbagliati.

Per prima cosa si individuano il minimo della funzione di correlazione ed altri 3 pseudo-minimi, ottenuti confrontando tra loro i valori a gruppi di 4 elementi per volta. Si procede poi col calcolo delle quantità di interesse [35]:

$$\Delta d = \sum_{i=1}^{3} |d(i) - d_{min}| \tag{4.17}$$

$$\Delta SAD = \sum_{i=1}^{3} (SAD(i) - SAD_{min}) \tag{4.18}$$

Δd viene detta *sharpness*, e rappresenta una misura della dispersione dei minimi locali attorno al minimo globale: valori alti indicano picchi dispersi, e dunque *match*

potenzialmente ambigui. Nel caso si verifichi questa evenienza, si procede misurando la *distinctiveness* associata al punto, ossia il valore $\Delta SAD/SAD_{min}$: più è alto, più il picco risulta marcato, e dunque affidabile. L'ultimo controllo effettuato riguarda il numero minimo di elementi su cui viene effettuata la minimizzazione: risultati ottenuti da funzioni di correlazione riguardanti un intervallo di valori limitato sono infatti da scartare perché affetti da rumore. Il valore di disparità ottenuto viene quindi raffinato approssimando localmente la funzione di correlazione con una curva del secondo ordine, di cui viene restituito il minimo:

$$d = \frac{s_0 - s_2}{2(s_0 + s_2 - 2s_1)} + d_1 \tag{4.19}$$

dove

$$\begin{cases} d_0 = d_1 - 1 \\ d_1 = d_{min} \\ d_2 = d_1 + 1 \end{cases} \quad \text{e} \quad \begin{cases} s_0 = score(d_0) \\ s_1 = score(d_1) \\ s_2 = score(d_2) \end{cases} \tag{4.20}$$

4.3 Implementazione

L'algoritmo fin qui descritto è stato sviluppato con l'obiettivo di ridurre al minimo il numero di operazioni necessarie al calcolo della *Disparity Space Image*; il peso computazionale finale dipende però in buona parte anche da come l'algoritmo viene implementato, e da come vengono sfruttate le risorse *hardware* a disposizione. L'approccio scelto è quello di realizzare un sistema il più possibile modulare, per poterlo adattare con minimo sforzo a diverse configurazioni. Il linguaggio utilizzato è il C++, con design basato su *policy classes* [4], unito a sezioni in *assembly* per sfruttare al meglio le istruzioni SIMD presenti in vari processori, che permettono, per un determinato *pixel*, di calcolare in contemporanea il punteggio associato a più valori di disparità. La disponibilità di sistemi con più *core* permette inoltre di ripartire l'elaborazione su più *thread* paralleli (ciascuno operante su una differente porzione orizzontale dell'immagine), riducendo in maniera consistente i tempi di esecuzione.

Viene ora presentata l'implementazione relativa al caso di immagini monocromatiche, 8 bpp (*bit per pixel*), per un processore che supporti il *set* di istruzioni SSSE3.

L'Algo. 2 descrive la procedura che permette di ricavare il valore di $SAD(x,y,d)$ per un generico punto all'interno dell'immagine: pur non comprendendo la gestione degli effetti dovuti ai bordi dell'immagine, si tratta della porzione di codice che viene eseguita più di frequente, e dunque la più rilevante dal punto di vista dell'ottimizzazione delle prestazioni. Per ognuna delle funzioni principali è riportato il codice *assembly* SIMD corrispondente[1].

```
function load(uint8_t* data, xmm_reg)
copies 8 bytes from memory into xmm_reg
in: mem[hg|fe|dc|ba]
out: xmm_reg[00|00|00|00|hg|fe|dc|ba]

movq data, xmm_reg
```

```
function store(xmm_reg, uint16_t* data)
copies 8 16-bit words from xmm_reg into (aligned) memory
in: xmm_reg[hh|gg|ff|ee|dd|cc|bb|aa]
out: mem[hh|gg|ff|ee|dd|cc|bb|aa]

movdqa data, xmm_reg
```

```
function unpack(xmm_reg)
expands lower 8 bytes of xmm_reg to 16-bit words
in: xmm_reg[00|00|00|00|hg|fe|dc|ba]
out: xmm_reg[0h|0g|0f|0e|0d|0c|0b|0a]
```

[1] La notazione utilizzata per le parti in *assembly* è la AT&T: `<opcode> <sorgente> <destinazione>`. Come *layout* per i registri `xmm` si è assunto il seguente: `[127:112|111:96|95:80|79:64|63:48|47:32|31:16|15:0]`

```
punpcklbw xmm_reg, xmm_reg ;xmm_reg[64:127] ← xmm_reg[0:63]
psllw 8, xmm_reg ;0-extend 8 bit left shift, 16-bit blocks
psrldq 1, xmm_reg ;0-pad 1 byte right shift, 128-bit blocks
```

function **match**(xmm_reg1, xmm_reg2, xmm_reg3, xmm_reg4)
computes the B term as per Eq. 4.7 and 4.9 for 8 disparities at a time
in:
xmm_reg1[0r_{07}|0r_{06}|0r_{05}|0r_{04}|0r_{03}|0r_{02}|0r_{01}|0r_{00}]
xmm_reg2[0r_{15}|0r_{14}|0r_{13}|0r_{12}|0r_{11}|0r_{10}|0r_{09}|0r_{08}]
xmm_reg3[0l_{07}|0l_{06}|0l_{05}|0l_{04}|0l_{03}|0l_{02}|0l_{01}|0l_{00}]
xmm_reg4[0l_{15}|0l_{14}|0l_{13}|0l_{12}|0l_{11}|0l_{10}|0l_{09}|0l_{08}]
out:
xmm_reg4[b_7b_7|b_6b_6|b_5b_5|b_4b_4|b_3b_3|b_2b_2|b_1b_1|b_0b_0]

```
psubw xmm_reg1, xmm_reg3 ;subtract, 16-bit blocks
psubw xmm_reg2, xmm_reg4
pabsw xmm_reg3, xmm_reg3 ;absolute value, 16-bit blocks
pabsw xmm_reg4, xmm_reg4
psubw xmm_reg3, xmm_reg4 ;subtract, 16-bit blocks
```

Algorithm 2: Calcolo della somiglianza tra finestre per il *pixel* (x,y) secondo l'Eq. 4.9

$dx\_beta[15]$;
$dx\_gamma[15]$;
for $i \leftarrow 0$ **to** 15 **do**
  $dx\_beta[i] \leftarrow D[x+n, y-m-1]$;
  $dx\_gamma[i] \leftarrow D[x+n, y+m]$;
end
`load(`*dx_beta, xmm*0`)`;
`load(`dx_beta+8*, xmm*1`)`;
`load(`*dx_gamma, xmm*2`)`;
`load(`$dx\_gamma+8$*, xmm*3`)`;
`unpack(`*xmm*0`)`;
`unpack(`*xmm*1`)`;
`unpack(`*xmm*2`)`;
`unpack(`*xmm*3`)`;
$iterations \leftarrow (d_{max} - d_{min})/16$;
$b[iterations * 16]$;
for $i \leftarrow 0$ **to** $iterations$ **do**
  `load(address(`$S[x+n+d_{min}+i*16, y-m-1]$`)`, *xmm*4`)`;
  `load(address(`$S[x+n+d_{min}+i*16+8, y-m-1]$`)`, *xmm*5`)`;
  `load(address(`$S[x+n+d_{min}+i*16, y+m]$`)`, *xmm*6`)`;
  `load(address(`$S[x+n+d_{min}+i*16+8, y+m]$`)`, *xmm*7`)`;
  `unpack(`*xmm*4`)`;
  `unpack(`*xmm*5`)`;
  `unpack(`*xmm*6`)`;
  `unpack(`*xmm*7`)`;
  `match(`*xmm*0, *xmm*2, *xmm*4, *xmm*6`)`;
  `match(`*xmm*1, *xmm*3, *xmm*5, *xmm*7`)`;
  `store(`*xmm*6, b`)`;
  `store(`*xmm*7, $b+8$`)`;
end
$i \leftarrow 0$;
for $d \leftarrow d_{min}$ **to** d_{max} **do**
  $\bar{x} \leftarrow x \mod (2win\_w+1)$;
  $\Delta_r[x][y][d] \leftarrow \Delta_r[x-1][y][d] - \Delta_p[\bar{x}][y][d] + b[i]$;
  $\Delta_p[\bar{x}][y][d] \leftarrow b[i]$;
  $i \leftarrow i+1$;
  $SAD[x][y][d] \leftarrow SAD[x][y-1][d] + \Delta_r[x][y][d]$;
end

Capitolo 5

L'algoritmo di rilevamento degli ostacoli

Nel Cap. 4 si è visto come, a partire da una coppia di immagini stereoscopiche, sia possibile ricavare una ricostruzione tridimensionale della porzione di mondo inquadrata; in questo capitolo verrà trattato il riconoscimento delle zone che si configurano come ostacoli da evitare. L'algoritmo, già introdotto nel Cap. 3 prevede l'utilizzo, in sequenza, di una serie di filtri:

ground rimuove la maggior parte dei punti che si trovano sul terreno

distance rimuove i punti troppo vicini o troppo lontani dal mezzo

layers raggruppa i punti con disparità compatibili

compactness valuta la compattezza dei gruppi di punti

column aggregation aumenta il valore di gruppi di punti adiacenti

fusione e *flood-fill* espande i semi derivanti dalla fusione dei punti sopravvissuti al filtraggio con quelli prodotti dai *laserscanner*

wrap calcola un inviluppo poligonale convesso dei gruppi di punti risultanti

5.1 *Ground filter* e *distance filter*

Per rimuovere gran parte dei punti appartenenti al terreno si può sfruttare il fatto che questo è caratterizzato da valori di disparità crescenti all'aumentare del numero di riga. Questo

approccio, già utilizzato in [13], è stato adattato allo scenario della *Urban Challenge*, osservando che l'asfalto risulta più omogeneo rispetto agli sterrati tipici della *Grand Challenge*, fatta eccezione per la segnaletica orizzontale, che risulta una *feature* molto marcata, e pressoché sempre presente all'interno della mappa di disparità. Per questo motivo si è scelto di ricercare la presenza di gradienti di disparità lungo linee dirette verso l'orizzonte, e non più verticali: in questo modo si va a rimuovere dalla mappa di disparità quasi del tutto il contributo dovuto ad elementi orizzontali. Una ulteriore elaborazione elimina i punti la cui coordinata Z (secondo il sistema di riferimento indicato in Fig. 5.1) non supera un certo valore, in considerazione del fatto che gli ambienti da affrontare risultano in gran parte pianeggianti. I vari passaggi fin qui descritti sono visibili in Fig. 5.2.

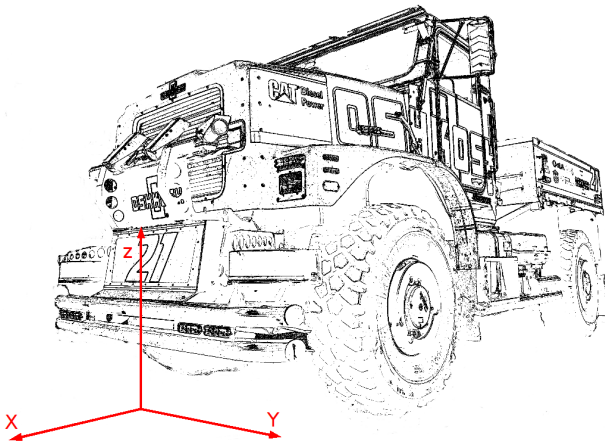
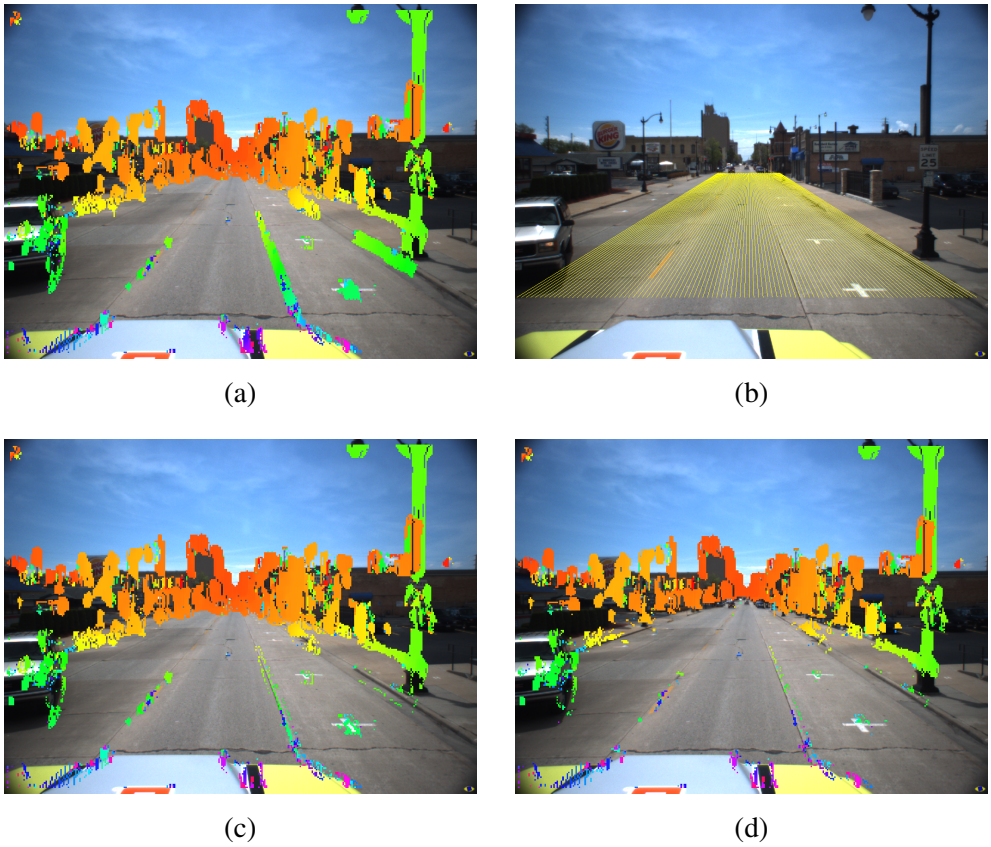


Figura 5.1: Sistema di riferimento per il veicolo autonomo TerraMax

Il filtro sulla distanza opera in modo da segnalare come non validi gli elementi che risultino troppo vicini o troppo lontani (coordinata X) dal veicolo, in modo da garantire che tutte le operazioni successive risultino relative alla sola area di interesse.



(a) (b)

(c) (d)

Figura 5.2: (a) Immagine acquisita dalla telecamera destra del sistema trinoculare del TerraMax, con sovrimpressa la mappa di disparità corrispondente (i punti di colore più caldo si trovano a distanze maggiori). (b) Le linee lungo cui si ricercano dei gradienti nei valori di disparità. (c) Punti sopravvissuti al primo filtraggio del terreno. (d) Punti che soddisfano anche la condizione sul valore della coordinata Z.

5.2 Costruzione dei *layer*

Per poter riconoscere oggetti differenti che risultino incolonnati per effetto della prospettiva si procede, in ogni colonna, a suddividere i punti in gruppi i cui valori di disparità siano omogenei tra loro, con i risultati esemplificati in Fig. 5.3.

Per ogni colonna dell'immagine viene dunque calcolato un istogramma dei valori di disparità, avendo cura di utilizzare una quantizzazione che generi insiemi significativi di punti. Definito *max_layers* il numero massimo di gruppi che si intende gestire, si estraggono dall'istogramma al più *max_layers* valori di disparità, in ordine decrescente di frequenza. I valori estratti dai vari istogrammi calcolati vanno via via a riempire due matrici, da utilizzare in seguito per gli ulteriori filtraggi previsti: la prima, denominata $column\_disparities$, e di dimensione $max\_layers \times width$, contiene alla posizione (i, j) il valore di disparità del *layer* i della colonna j; la seconda, denominata *pixels_layers*, e di dimensione $height \times width$, contiene alla posizione (i, j) il *layer* cui il punto a quelle coordinate appartiene. Viene inoltre generata una terza matrice, denominata *column_average_distances*, di dimensione $max\_layers \times width$, in cui alla posizione (i, j) viene inserita la distanza media dal veicolo dei punti che compongono il gruppo.

5.3 *Compactness filter*

Utilizzando la matrice *pixels_layers* appena costruita e l'immagine di disparità in ingresso si va ad applicare, ad ogni colonna e per ciascun gruppo di punti al suo interno, l'algoritmo di filtraggio presentato in [16]. La prima operazione consiste nel calcolo del baricentro dell'insieme di *pixel* in esame; a partire da questo, muovendosi verso l'alto e verso il basso lungo la colonna, si attribuisce un bonus in presenza di sequenze di punti appartenenti al *layer*, ed una penalità nel caso si rilevino delle discontinuità. Il punteggio complessivo viene infine confrontato con una soglia: nel caso non risulti sufficiente il gruppo viene rimosso, altrimenti il valore viene inserito in una matrice denominata *column_scores*, di dimensione $max\_layers \times width$, alla posizione (i, j), con i e j pari rispettivamente al *layer* e alla colonna correnti.

Di tutti i gruppi risultati validi si va ad evidenziare la parte predominante, predisponendoli così all'utilizzo come seme nella successiva operazione di *flood-fill*, descritta nella Sez. 5.5. Per isolare la porzione più caratteristica di ciascuno si va ad identificare la sequenza più lunga di *pixel* contigui, sostituendone il valore di disparità con quello del baricentro del *layer*; tutti gli altri punti vengono invece marcati come non validi.

5.4 *Column aggregation*

La presenza di insiemi di colonne contigue con valori di disparità simili è fortemente correlata all'effettiva esistenza di un ostacolo, e dunque è opportuno andare a ricercare questo tipo di

caratteristica all'interno dell'immagine. Per prima cosa va notato che con l'aumentare della distanza le dimensioni in *pixel* degli oggetti all'interno dell'immagine diminuisce, e dunque l'area di ricerca va circoscritta in base alla distanza media del gruppo dal veicolo:

- fino a 20 m: due colonne a sinistra e due a destra;
- da 20 a 50 m: una colonna a sinistra ed una a destra;
- oltre 50 m: nessun tentativo di aggregazione.

Per ogni *layer* si va a cercare nella relativa zona d'interesse la presenza di altri gruppi con disparità compatibile e, nel caso questi esistano, si va a sommarne il punteggio (ricavato da *column_scores*) a quello del *layer* in esame. Il risultato viene poi inserito nella matrice *aggregated_column_scores* le cui caratteristiche rispecchiano quelle di *column_scores*.

Dopo questa ultima operazione di filtraggio si calcola per ogni gruppo un punteggio finale, e lo si confronta con una soglia, andando a selezionare solo le regioni che risultano sufficientemente affidabili:

$$\begin{aligned} score(l,x) \quad = \quad & aggregated\_column\_scores(l,x) * \\ & column\_average\_distances(l,x) \\ & \forall l \in [0, max\_layers[\text{ e } \forall x \in [0, width[\end{aligned} \tag{5.1}$$

5.5 Fusione e *flood-fill*

I gruppi di punti sopravvissuti ai filtraggi fin qui presentati corrispondono con un alto grado di probabilità alla presenza di ostacoli effettivi; il limite del tipo di informazione che forniscono è la fedeltà con cui vengono rilevate le forme degli oggetti identificati, dal momento che i punti ritenuti sufficientemente affidabili sono in genere pochi. Per ottenere una ricostruzione più dettagliata si ricorre ad un algoritmo di *flood-fill*, che va ad espandere ricorsivamente gli 8-vicini di ciascun seme, fino ad inglobare tutti gli elementi contigui della *Disparity Space Image* di partenza che abbiano valori di disparità compatibili, secondo il criterio

$$\frac{|DSI(x,y) - d_{seed}|}{DSI(x,y) * d_{seed}} < t_{exp} \tag{5.2}$$

I punti *raw* provenienti dai *laserscanner* frontali presentano caratteristiche comparabili a quelli ottenuti mediante filtraggio della *Disparity Space Image*, a patto di rimuovere quelli dovuti a riflessioni sul terreno (ad esempio utilizzando l'algoritmo presentato in [14]); per questa ragione è possibile impiegarli per arricchire l'insieme dei semi da utilizzare per la fase

di *flood-fill*. Questa tecnica di fusione permette di irrobustire i risultati, senza però rendere l'algoritmo di visione dipendente dalla presenza e dal corretto funzionamento degli altri sensori, dal momento che l'espansione viene effettuata tenendo come riferimento i valori della *Disparity Space Image*.

Durante la fase di espansione bisogna fare attenzione a non andare ad includere il terreno su cui poggia ciascun ostacolo: esiste infatti sempre una porzione del suolo i cui valori di disparità sono per forza di cose molto simili a quelli della base dell'oggetto in esame, trovandosi entrambi alla stessa distanza dal veicolo. Per ovviare a questo inconveniente è stato introdotto un ulteriore criterio che vincola l'inclusione dei punti:

$$\frac{DSI(x,y)}{ground\_disp(y)} < t_{gnd} \tag{5.3}$$

dovendo espandere il punto alle coordinate $(\bar{x},\bar{y})$, tale controllo viene effettuato solo per i *pixel* alle posizioni $(\bar{x}-1,\bar{y})$, $(\bar{x}+1,\bar{y})$, $(\bar{x}-1,\bar{y}+1)$, $(\bar{x},\bar{y}+1)$ e $(\bar{x}+1,\bar{y}+1)$, dal momento che i movimenti verso l'alto non risultano pressoché mai problematici.

5.6 Wrapping

L'operazione successiva consiste nel trasformare i punti risultanti dall'espansione presentata nella Sez. 5.5 in oggetti gestibili dal sistema di *path planning*: questa componente opera su poligoni bidimensionali i cui bordi descrivono, in coordinate mondo e secondo una visuale dall'alto, l'ingombro degli ostacoli presenti attorno al mezzo.

L'algoritmo utilizzato prevede per prima cosa di andare a raggruppare in maniera ricorsiva tutti i punti che soddisfano il criterio

$$\begin{cases} (x_1-x_0)^2+(y_1-y_0)^2 < t_{xy}^2 \\ |z_1-z_0| < t_z \end{cases} \tag{5.4}$$

Di ciascun gruppo viene quindi calcolato il *convex hull*[1], utilizzato un algoritmo denominato *gift wrapping*, descritto in [20], di complessità $O(nh)$, con n pari al numero di punti in esame e h al numero di punti che appartengono al bordo.

Per poter ricavare direzione e velocità di movimento di ogni oggetto si ricorre ad una procedura di *tracking* di alto livello, inizialmente demandata al *path planner*, ed in seguito

[1] Il *convex hull* (o inviluppo complesso) dell'insieme I è costituito dall'intersezione di tutti gli insiemi convessi che contengono I

trasferita al sistema di visione per far fronte ai ritmi di sviluppo molto serrati imposti da una competizione come la *Urban Challenge*.

Nello sviluppo di questo algoritmo si è dovuto tener conto del fatto che la posizione del mezzo è espressa in coordinate UTM[2] –relative ad un punto fisso del mondo– e il suo orientamento come angolo rispetto al Polo Nord, mentre le coordinate degli ostacoli rilevati sono espresse rispetto ad un sistema di riferimento solidale al veicolo: occorre quindi compensare il movimento intercorso tra gli istanti di tempo $t-1$ e t per rendere confrontabili tra loro i valori rilevati.

Dette (x_t, y_t, h_t) e $(x_{t-1}, y_{t-1}, h_{t-1})$ posizione e *heading* del TerraMax agli istanti di tempo t e $t-1$, (p_x, p_y) le coordinate di un generico punto calcolate all'istante $t-1$, e (p'_x, p'_y) quelle dello stesso punto, rispetto al sistema di riferimento solidale al veicolo all'istante t, si pone:

$$\begin{cases} \Delta_x = x_t - x_{t-1} \\ \Delta_y = y_t - y_{t-1} \\ \Delta_h = h_t - h_{t-1} \end{cases} \tag{5.5}$$

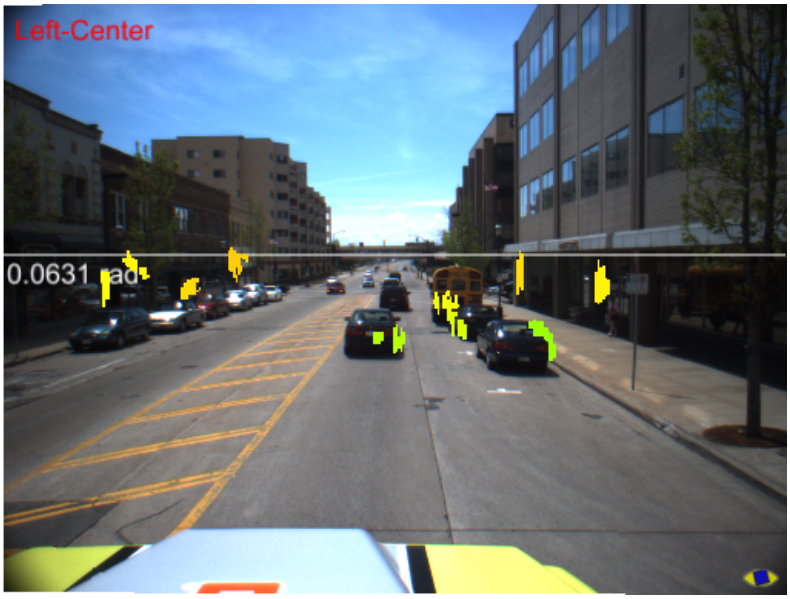
si definiscono inoltre t_x e t_y come

$$\begin{cases} t_x = -\Delta_x \cos h_t - \Delta_y \sin h_t \\ t_y = -\Delta_x \sin h_t + \Delta_y \cos h_t \end{cases} \tag{5.6}$$

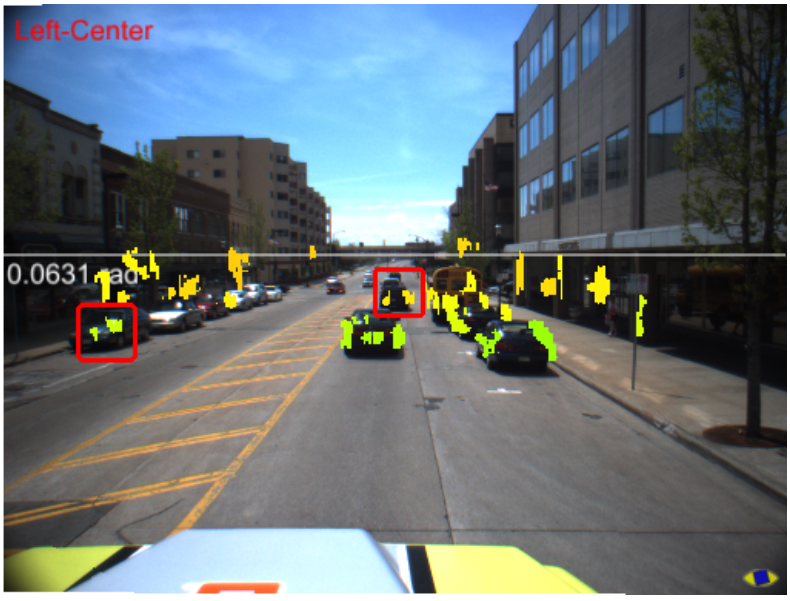
la trasformazione desiderata risulta infine essere

$$\begin{cases} p'_x = p_x \cos \Delta_h - p_y \sin \Delta_h + t_x \\ p'_y = p_x \sin \Delta_h + p_y \cos \Delta_h + t_y \end{cases} \tag{5.7}$$

[2] *Universal Transverse Mercator*, descritte in [34]



(a)



(b)

Figura 5.3: (a) Ostacoli rilevati utilizzando un solo *layer*. (b) Ostacoli rilevati utilizzando tre *layer*: i riquadri rossi ne evidenziano alcuni che vengono rilevati in modo corretto nonostante appaiano incolonnati per effetto della prospettiva.

Capitolo 6

Conclusioni

Gli algoritmi qui presentati hanno dato al TerraMax un sensore di ostacoli affidabile, al punto da poter far fronte al mancato funzionamento dei *laserscanner*, verificatosi in una delle prove dell'NQE[1], durante le fasi conclusive della *DARPA Urban Challenge*. Il contributo alla procedura di riconoscimento della segnaletica orizzontale è stato inoltre considerevole: oltre ad una riduzione del carico computazionale complessivo, ottenuta mettendo in comune le fasi di pre-elaborazione di *obstacle* e *lane detection*, si sono potute migliorare le prestazioni di quest'ultimo andando a mascherare le aree dell'immagine occupate da ostacoli, riuscendo così ad evitare i falsi rilevamenti dovuti ad elementi come *guard-rail*, paletti, lampioni e parti di carrozzeria di altri veicoli.

Il metodo di calcolo dell'immagine di disparità proposto si è dimostrato molto flessibile, ed è stato possibile utilizzarlo in contesti anche molto differenti: un esempio è l'applicazione ad una coppia di immagini IPM, eseguendo i confronti su regioni di dimensioni considerevoli, allo scopo di migliorare la rimozione della componente dovuta al terreno quando si vuole andare ad individuare oggetti presenti nelle immediate vicinanze di un veicolo con telecamere molto distorcenti [10].

L'idea di utilizzare un approccio di tipo incrementale che tuttavia consideri la componente dovuta al terreno sta anche portando alla realizzazione di algoritmi ibridi, in cui si sacrifica in parte l'indipendenza dalle dimensioni delle finestre di *matching*, rilassando però al contempo il vincolo di calcolare i valori di correlazione per tutti i *pixel*, anche quelli relativi a zone prive di contenuto informativo: in questo modo diventa possibile ridurre in modo consistente

[1] *National Qualification Event*, fase eliminatoria che ha preceduto la competizione finale

i tempi di elaborazione nel caso di immagini in ingresso di tipo derivativo, da cui si voglia andare a ricavare una DSI sparsa utilizzando intorni limitati di ciascun *pixel* per i confronti.

Un campo di indagine interessante è anche quello di realizzare implementazioni che sfruttino, in maniera trasparente al programmatore, risorse di calcolo di tipo differente rispetto alle tradizionali CPU, come le schede grafiche sempre più potenti che si stanno diffondendo sul mercato *consumer*; l'architettura *software* utilizzata permette infatti di realizzare un'estensione di questo tipo in maniera naturale, sfruttando poi per la parte di più basso livello una delle diverse librerie per il calcolo di tipo GPGPU[2] esistenti.

[2] *General-Purpose computing on Graphics Processing Units*

Parte II

Lane detection su piattaforma *embedded*

Capitolo 7

Le componenti *hardware*

Nel Cap. 1 sono state presentate alcune linee guida in materia di sicurezza stradale che prevedono l'introduzione su larga scala di sistemi avanzati di ausilio alla guida a bordo dei veicoli, di cui il *lane detector* presentato in questa tesi è un esempio. Per realizzare dispositivi di questo genere occorre far fronte non solo alla complessità legata allo sviluppo di algoritmi sofisticati ed affidabili, ma anche alle caratteristiche del mercato *automotive*, che richiede prodotti dal costo contenuto, ma comunque in grado di funzionare anche in condizioni ambientali molto sfavorevoli.

Questa combinazione di requisiti rende problematico l'impiego di un *Personal Computer*, che per costo, dimensioni e condizioni operative mal si adatta a questo genere di utilizzo; volendo creare un prodotto appetibile risulta più opportuno affidarsi ad un sistema di tipo *embedded* progettato ad hoc: nel caso in esame le caratteristiche essenziali sono un sensore con risoluzione e dinamica adeguate, un processore adatto all'elaborazione di immagini, e la possibilità di fornire i risultati all'esterno in maniera modulare. È bene inoltre richiamare l'attenzione sul fatto che che le attuali esigenze di installazione a bordo del veicolo impongono l'impiego di una sola telecamera: questo tipo di soluzione permette sia di ridurre notevolmente i costi di integrazione nel caso in cui il sistema venga fornito come *optional* dal costruttore, sia di realizzare un dispositivo molto più compatto nel caso in cui la distribuzione avvenga attraverso canali di tipo *aftermarket*.

7.1 Processore

Il processore selezionato è un Texas Instruments[1] TMS320DM642-600[2] (Fig. 7.1) a 600 MHz, con 16 KB di L1 *cache* e fino a 64 MB di SDRAM. L'unità aritmetica è di tipo *fixed point*, e la capacità di calcolo raggiunge i 4800 MIPS. Questo DSP permette diversi tipi di configurazione dal punto di vista dell'acquisizione video, in quanto è possibile sia collegare i sensori direttamente alle sue Video Port, che demandare questa operazione ad una FPGA esterna; con la prima soluzione si ottiene un unico dispositivo molto compatto, in grado di provvedere sia all'acquisizione dei dati che alla loro elaborazione, mentre la seconda è da preferire quando si ha la necessità di installare le telecamere ad una certa distanza: in questo modo è infatti possibile utilizzare per la trasmissione delle immagini protocolli adatti allo scopo (come FireWire ed LVDS). Il TMS320DM642-600 è in grado di gestire direttamente sia connessioni Ethernet 10/100 che seriali, cosa che semplifica l'interfacciamento con i dispositivi esterni a cui si inviano i risultati dell'algoritmo. Le temperature di impiego sono comprese nel *range* 0 – 90 °C, con un prezzo che si aggira sui 50 $.

Figura 7.1: Il processore Texas Instruments TM320DM642.

[1] http://www.ti.com/

[2] Le specifiche sono disponibili all'indirizzo http://www.ti.com/lit/gpn/tms320dm642

7.2 Sensore video

Per il comparto video si è optato per un Aptina[3](ex Micron) MT9V022, visibile in Fig. 7.2; questo sensore è un CMOS da 1/3″, disponibile sia in versione monocromatica che a colori con *pattern* di Bayer, ed è caratterizzato da una discreta sensibilità nello spettro del vicino infrarosso[4], come si può osservare in Fig. 7.3: questa peculiarità lo rende particolarmente adatto all'impiego in condizioni di scarsa illuminazione. La risoluzione massima è di 752×480 punti, alla quale si possono acquisire immagini alla velocità di 60 fps; ciascun *pixel* ha dimensione 6 μm×6 μm, e la dinamica è superiore a 55 dB. Le temperature a cui può operare sono quelle comprese nel *range* -40 – 85 °C, mentre il prezzo è di circa 16 $.

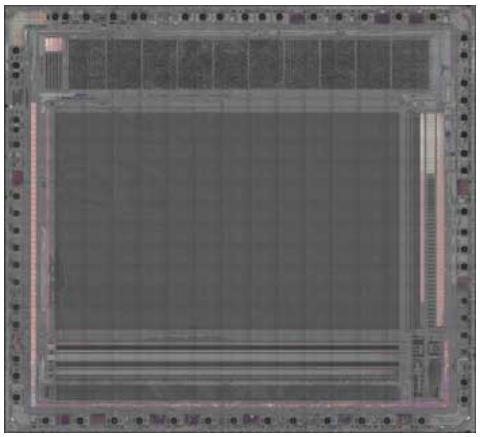


Figura 7.2: Il sensore CMOS Aptina MT9V022.

7.3 Prototipo

Lo sviluppo di un progetto del genere richiede un investimento considerevole sia in termini economici che di tempo, che sono giustificati solo se l'algoritmo è in grado di fornire le prestazioni previste con le risorse *hardware* a disposizione: è dunque opportuno partire dalla

[3] http://www.aptina.com

[4] Appartengono a questa categoria le radiazioni con lunghezza d'onda compresa tra i 2500 e i 750 nm

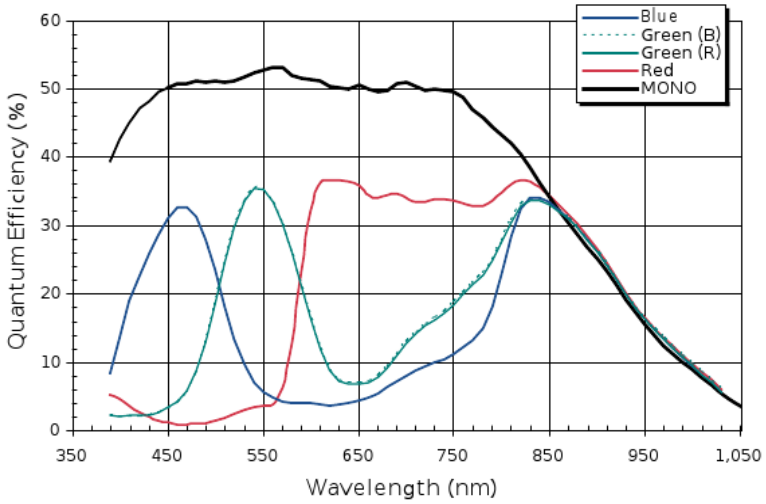


Figura 7.3: Efficienza spettrale dell'MT9V022.

realizzazione di un prototipo costituito per quanto possibile dagli stessi componenti *hardware* e *software* da impiegare nella versione finale, seppur con un minor grado di integrazione.

La scelta è stata quella di utilizzare un dispositivo Valde[5] VS1501, equipaggiato col DSP TMS320DM642-600 descritto poco sopra, 64 MB di SDRAM, due ingressi FireWire-400 indipendenti, una porta Ethernet 10/100, una seriale, due GP-I/O, ed una uscita video composita (Fig. 7.4), e di connettere a questa unità una telecamera FireWire Point Grey Research FireFly MV, visibile in Fig. 7.5, che monta il sensore Aptina MT9V022 ed è dotata di microlenti con focale 6,4 mm. Questa configurazione risulta molto efficace in fase di sviluppo per diversi motivi:

- permette di acquisire in modo semplice sequenze di immagini da utilizzare in fase di stesura e validazione degli algoritmi: il sensore per tipologia e posizionamento rispecchia fedelmente le caratteristiche del prodotto finale, mentre la connessione FireWire consente l'impiego di un sistema di registrazione delle immagini tradizionale basato su *Personal Computer* [5];
- facilita le fasi di *debug* e ottimizzazione, dal momento che è possibile caricare il codice

[5] htt://www.valdesystems.com

da eseguire via JTAG[6] (dunque senza operazioni di sovrascrittura della EEPROM), e la porta Ethernet permette la lettura di immagini registrate su cui andare ad eseguire l'algoritmo: i risultati possono così essere confrontati passo a passo con quelli della controparte per *Personal Computer*, la cui correttezza è più facile da verificare;

- l'uscita video integrata consente l'utilizzo di uno schermo esterno per valutare in maniera qualitativa i risultati prodotti nel corso di *test* condotti sul campo, mentre via rete è possibile sia provvedere alla loro registrazione, che intervenire sui parametri di funzionamento dell'algoritmo fino a trovare quelli ottimali.

La memoria a bordo del dispositivo permette infine di registrare brevi sequenze di immagini (circa 10 s) e di trasferirle via rete ad un *Personal Computer*: tale caratteristica permette sia una diagnostica minimale che l'acquisizione dei dati necessari alla calibrazione del sistema, che può essere a questo punto effettuata *off-line*.

[6] *Joint Test Action Group*, nome con cui è più comunemente noto lo standard IEEE 1149.1, inizialmente creato per il collaudo dei circuiti stampati e oggi molto usato anche per il *debug* dei sistemi *embedded*

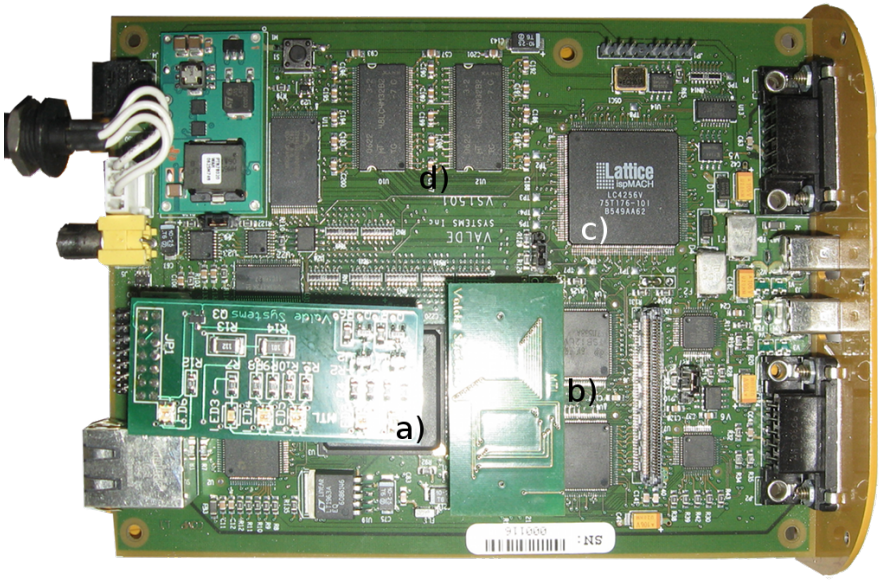


Figura 7.4: I principali componenti presenti all'interno del Valde VS1501 – a) DSP Texas Instruments TMS320DM642-600; b) *Controller* FireWire; c) CPLD Lattice 4265V per gestire il trasferimento delle immagini in memoria; d) SDRAM.

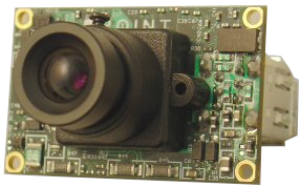


Figura 7.5: Telecamera Point Grey Research FireFly MV.

Capitolo 8

L'algoritmo

Dei vincoli di tipo tecnologico ed economico presentati nel Cap. 7 uno dei più stringenti in fase di progettazione dell'algoritmo è quello di avere a disposizione un'unica telecamera: questo tipo di soluzione complica sia la stima dei parametri estrinseci istantanei della telecamera (ed in particolare dell'angolo di beccheggio, o *pitch*, che varia in maniera sensibile durante la marcia), che la corretta discriminazione tra segnaletica orizzontale e altri tipi di oggetti (come ad esempio *guard-rail* e paletti), che possono talvolta presentare caratteristiche compatibili. Questo tipo di considerazioni hanno portato ad organizzare l'elaborazione in due fasi successive: in un primo momenti si va a rilevare il *pitch* del veicolo, utilizzando un algoritmo derivato da quello presentato in [6], ed in seguito si procede al riconoscimento delle linee che delimitano le corsie, con una versione semplificata dell'algoritmo sviluppato nell'ambito della *Urban Challenge* e presentato in [11].

8.1 *Pitch detection*

In letteratura esistono diversi di metodi che permettono di ricavare dalle immagini provenienti da una sola telecamera il valore di beccheggio istantaneo del veicolo. A questi si aggiungono vari sistemi di stabilizzazione dell'immagine, che permettono di continuare ad utilizzare i parametri di calibrazione estrinseci della telecamera con un errore contenuto; alcuni algoritmi di stabilizzazione sono presentati nell'appendice C.

Nel caso in esame si può osservare che gli ambienti in cui è utile conoscere con precisione il *pitch* del veicolo sono proprio quelli in cui è presente la segnaletica orizzontale: tale caratteristica permette di ottenere buone indicazioni sulla posizione del punto di fuga

all'interno dell'immagine, e da questo le informazioni di calibrazione desiderate. La prima operazione effettuata consiste nell'applicare un filtro derivativo di Sobel verticale con segno, in modo da isolare i bordi più marcati, coi risultati visibili in Fig. 8.1.

Il passo successivo consiste nel generare, all'interno di un'area di ricerca definita nella parte bassa dell'immagine, e visibile in Fig. 8.2, una serie di regioni connesse; ciascuna regione viene costruita a partire da uno dei punti appartenenti ad un bordo verticale sufficientemente forte non ancora classificato, che viene espanso secondo le direzioni indicate in Fig. 8.3. La procedura di etichettatura è codificata in modo iterativo, e non ricorsivo, perché questo approccio, pur essendo meno immediato da implementare, ha il vantaggio di non utilizzare lo *stack*.

Dall'insieme dei *cluster* generati vengono rimossi quelli costituiti da un numero di punti troppo piccolo, per poi andare a calcolare il baricentro per ciascuno dei rimanenti. L'immagine viene a questo punto divisa in settori, e i *cluster* vengono raggruppati in base al settore di appartenenza del proprio baricentro: dette (x_c, y_c) le coordinate del centro dell'immagine, (x_b, y_b) quelle del baricentro, e α l'ampiezza, in radianti, di ciascun settore, l'indice corrispondente sarà dato da:

$$\left\lfloor \frac{\arctan\left(\dfrac{y_b - y_c}{x_b - x_c}\right)}{\alpha} \right\rfloor \tag{8.1}$$

All'interno di ciascun settore i *cluster* vengono divisi in due famiglie, in base al segno dei *pixel* che li compongono; da ognuno di questi gruppi di *cluster* viene estratta, tramite un procedimento di regressione lineare ai minimi quadrati, l'equazione della retta che meglio li approssima: detto N il numero complessivo di punti di coordinate (x_i, y_i) nel gruppo di *cluster* in esame, e posti

$$\begin{cases} S_{xx}^2 = \sum_{i=1}^{N} x_i^2 - \dfrac{1}{N}\left(\sum_{i=1}^{N} x_i\right)^2 \\ S_{xy}^2 = \sum_{i=1}^{N} x_i y_i - \dfrac{1}{N}\left(\sum_{i=1}^{N} x_i y_i\right)^2 \\ S_{yy}^2 = \sum_{i=1}^{N} y_i^2 - \dfrac{1}{N}\left(\sum_{i=1}^{N} y_i\right)^2 \end{cases} \tag{8.2}$$

la retta r cercata è quella di equazione:

$$r)\ ax + by + c = 0 \tag{8.3}$$

con

$$\begin{cases} a = -N * S_{yy}^2 \\ b = N * S_{xy}^2 \\ c = S_{yy}^2 \sum_{i=1}^{N} x_i - S_{xy}^2 \sum_{i=1}^{N} y_i \end{cases} \tag{8.4}$$

mentre l'errore di approssimazione e vale

$$e = \frac{1}{N-2} \left(S_{xx}^2 - \frac{S_{xy}^{2\,2}}{S_{yy}^2} \right) \tag{8.5}$$

Per ciascun gruppo di *cluster* si costruisce il *bounding box*[1] corrispondente, e se ne calcola la lunghezza (al quadrato, per ragioni di efficienza) della diagonale l^2. Ogni linea viene a questo punto giudicata valida solo se:

- $e < t_e$
- $l^2 > t_{l^2}$
- r interseca una regione di interesse rettangolare entro cui dovrà andare a cadere il punto di fuga stimato (vedi Fig. 8.2);
- $\left| \pi - \arctan\left(\frac{b}{a}\right) \right| > t_v$ con a e b ricavati dall'Eq. 8.4; in questo modo vengono rimosse tutte le rette troppo verticali.

Le rette selezionate vengono ordinate in base al loro peso (pari al numero di punti complessivo dei *cluster* da cui sono state ricavate), e solo le $\bar{n}$ migliori vengono mantenute; l'operazione viene effettuata utilizzando la variante dell'algoritmo di *selection sort* descritta nell'Algo. 3. Questo metodo è stato scelto perché ha la proprietà di portare, dopo i iterazioni, i primi i elementi nella posizione corretta: in questo modo sono sufficienti $\bar{n}$ cicli per ottenere il risultato desiderato.

Le $\bar{n}$ rette migliori vengono a questo punto intersecate a coppie, e si va quindi a valutare se

$$\alpha_{min} < \alpha < \alpha_{max}$$

dove α è l'angolo con cui l'intersezione avviene; questo controllo può essere effettuato in maniera efficiente utilizzando il metodo di seguito esposto.

[1] è il più piccolo rettangolo che contiene tutti i punti in esame

Algorithm 3: *Partial selection sort*

```
for i ← 0 to n̄ − 1 do
    min_pos = i;
    min_val = elements[i];
    for j ← i + 1 to elements_num do
        if min_val > elements[j] then
            min_pos = j;
            min_val = elements[j];
        end
    end
    Swap(elements[i], elements[min_pos]);
end
```

Detti $(a_0, b_0, c0)$ i parametri della retta r_0 e $(a_1, b_1, c1)$ quelli della retta r_1, si ha che $\alpha_{min} < \alpha$ quando è verificata una delle seguenti condizioni:

$$\begin{cases} \cos\alpha_{min} < 0 \\ a_0a_1 + b_0b_1 < 0 \\ (a_0^2 + b_0^2)(a_1^2 + b_1^2)\cos^2\alpha_{min} < (a_0a_1 + b_0b_1)^2 \end{cases}$$

$$\begin{cases} \cos\alpha_{min} \geq 0 \\ a_0a_1 + b_0b_1 < 0 \end{cases} \tag{8.6}$$

$$\begin{cases} \cos\alpha_{min} \geq 0 \\ a_0a_1 + b_0b_1 > 0 \\ (a_0^2 + b_0^2)(a_1^2 + b_1^2)\cos^2\alpha_{min} > (a_0a_1 + b_0b_1)^2 \end{cases}$$

mentre $\alpha < \alpha_{max}$ quando:

$$\begin{cases} \cos\alpha_{max} < 0 \\ a_0a_1 + b_0b_1 < 0 \\ (a_0^2+b_0^2)(a_1^2+b_1^2)\cos^2\alpha_{max} > (a_0a_1+b_0b_1)^2 \end{cases}$$

$$\begin{cases} \cos\alpha_{max} < 0 \\ a_0a_1 + b_0b_1 \geq 0 \end{cases} \tag{8.7}$$

$$\begin{cases} \cos\alpha_{max} > 0 \\ a_0a_1 + b_0b_1 \geq 0 \\ (a_0^2+b_0^2)(a_1^2+b_1^2)\cos^2\alpha_{max} < (a_0a_1+b_0b_1)^2 \end{cases}$$

come si nota occorrono solo confronti, mentre il calcolo dei coefficienti $\cos\alpha_{min}$ e $\cos\alpha_{max}$ può essere effettuato *offline*. Nel caso che le condizioni riportate risultino soddisfatte si procede al calcolo del punto d'intersezione, le cui coordinate valgono:

$$\begin{cases} X = \dfrac{c_0b_1 - b_0c_1}{b_0a_0 - b_1a_1} \\ \\ Y = \dfrac{a_0c_1 - a_1c_0}{b_0a_0 - b_1a_1} \end{cases} \quad \text{se} \quad b_0a_0 - b_1a_1 \neq 0 \tag{8.8}$$

mentre il valore di confidenza associato alla linea r_0 sarà incrementato della quantità

$$w = \min(N_o, N_1) \tag{8.9}$$

dove N_0 ed N_1 sono il numero di punti presenti nei *cluster* che hanno dato origine rispettivamente ad r_0 ed r_1; questa operazione fa sì che le intersezioni prodotte da rette ricavate con pochi punti (e quindi in genere più rumorose) abbiano peso minore nella determinazione della posizione del punto di fuga.

Terminata la generazione delle intersezioni si procede al calcolo delle coordinate del punto di fuga come:

$$\begin{cases} f_x = \dfrac{\sum_{i=1}^{\bar{n}} X_i w_i}{\sum_{i=1}^{\bar{n}} w_i} \\ \\ f_y = \dfrac{\sum_{i=1}^{\bar{n}} Y_i w_i}{\sum_{i=1}^{\bar{n}} w_i} \end{cases} \tag{8.10}$$

tale valore viene accettato solo se $\sum_{i=1}^{\bar{n}} w_i > t_w$, $\sigma_X^2 < t_{\sigma_X^2}$ e $\sigma_Y^2 < t_{\sigma_Y^2}$.

Nel caso in cui il punto (f_x, f_y) sia accettabile il valore ϑ del *pitch* del veicolo viene calcolato come:

$$\vartheta = \arctan\left[\left(1 - \frac{f_y}{v_0}\right)\tan\left(\frac{\beta}{2}\right)\right] \tag{8.11}$$

dove v_0 è il *principal point* dell'immagine, e β l'apertura verticale espressa in radianti.

Se l'algoritmo fin qui esposto non è in grado di determinare la posizione del punto di fuga –ad esempio per la mancanza di un numero sufficiente di elementi rettilinei utili, o la presenza di elementi di disturbo, come veicoli molto vicini– si procede all'inseguimento dell'orizzonte a partire dall'ultimo valore utile di (f_x, f_y). Questa operazione viene effettuata costruendo un istogramma dei bordi orizzontali su una finestra centrata attorno all'orizzonte (visibile in Fig. 8.2), che viene poi traslato in verticale per tutti i valori di *offset* compresi in $[-\Delta, \Delta]$ e confrontato con quello relativo al fotogramma precedente, utilizzando come metrica la somma delle differenze in modulo dei rispettivi valori. Il valore di *offset* per cui la differenza risulta minima viene infine utilizzato per il calcolo di ϑ, sempre secondo l'Eq. 8.11.

In Fig. 8.4 sono visibili alcune delle componenti dell'algoritmo di *pitch detection* fin qui descritte: le linee dirette verso il punto di fuga, il valore stimato, e l'istogramma dei bordi orizzontali.

8.2 *Lane detection*

Per quanto riguarda l'algoritmo monoculare di rilevamento delle linee di demarcazione della carreggiata la scelta è caduta su quello sviluppato per la *Urban Challenge*, presentato nell'App. B.

Questa soluzione, adottata sia per la qualità dei risultati prodotti che l'affidabilità dimostrata, ha comunque richiesto un certo lavoro di adattamento per renderne i tempi di esecuzione sul DSP scelto compatibili con uno scenario di utilizzo *soft-realtime*. I principali punti di intervento sono stati l'eliminazione delle funzionalità meno interessanti, e la velocizzazione della trasformazione prospettica inversa, che è risultata essere l'operazione più costosa in termini di carico computazionale; un esempio del risultato ottenuto è visibile in Fig. 8.5

Nel caso del TerraMax, progettato tenendo conto del codice della strada in vigore negli Stati Uniti d'America, risulta importante rilevare la presenza di linee di colore giallo, in quanto queste separano corsie in cui i veicoli marciano in senso opposto: in questo modo diventa ad esempio più semplice pianificare in modo adeguato le manovre di sorpasso. Questo genere di problematica però non si applica al caso europeo, mercato a cui si rivolge il dispositivo qui presentato, e presenta due notevoli svantaggi:

- richiede l'uso di una telecamera a colori;
- riduce di molto le prestazioni del sistema (di fatto le dimezza).

La ragione del requisito elencato nel primo punto è evidente, mentre il costo associato in termini di prestazioni potrebbe esserlo meno: questo genere di telecamere infatti sfrutta un *Color Filter Array*[2] per ricavare l'informazione relativa al colore, a scapito della sensibilità, che risulta ridotta in modo non trascurabile sia nello spettro del visibile che nel vicino infrarosso, come si può vedere dai grafici riportati in Fig. 7.3.

Il calo di prestazioni indicato nel secondo punto è comportato dal fatto che tutta l'elaborazione prevista per l'estrazione delle linee bianche deve essere replicata su un'immagine in cui viene messa in evidenza la componente gialla, a cui si somma il tempo necessario alla generazione dell'immagine stessa: queste operazioni, ed in particolare il calcolo di un'*Inverse Perspective Mapping* aggiuntiva, porta il tempo di esecuzione ben oltre i 100 ms che costituiscono il limite oltre al quale un dispositivo come quello presentato in questa tesi inizia a perdere di efficacia.

Queste osservazioni hanno portato alla decisione di rimuovere il supporto al rilevamento delle linee gialle, che portano a problemi e limitazioni considerevoli, avendo per di più in Europa un'area di impiego decisamente limitata (per lo più costituita da cantieri e corsie degli autobus).

Un'altra caratteristica a cui si è scelto per il momento di rinunciare per ragioni di efficienza (ma che potrebbe trovare spazio nel caso diventi indispensabile supportarla) è la capacità di gestire in modo specifico le linee doppie. Questa abilità presuppone sia l'applicazione di un filtraggio di basso livello che vada ad evidenziare all'interno dell'immagine i *pattern* di tipo DLDLD[3], che un aumento di complessità (seppur contenuto) nella parte di alto livello deputata al riconoscimento dell'effettiva presenza delle linee.

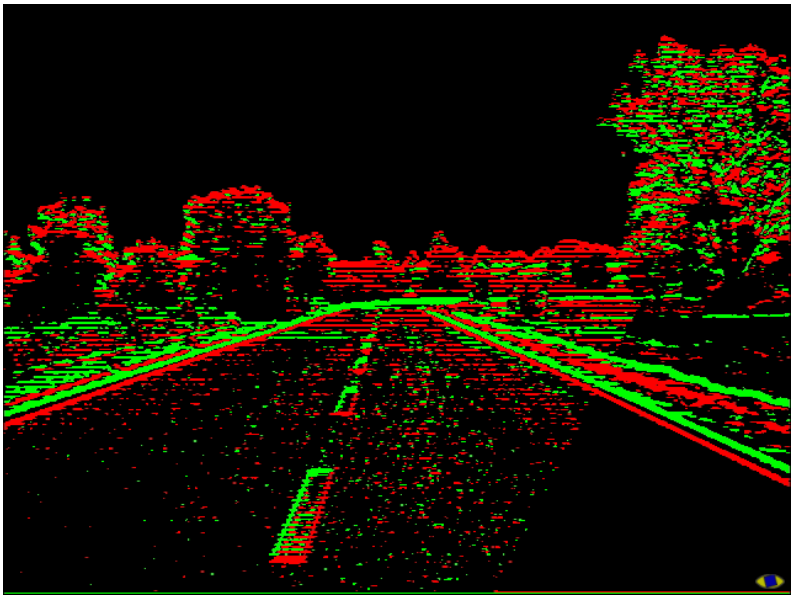
Come si è ricordato poco sopra l'operazione di rimozione della prospettiva è molto onerosa, e per questa ragione si è scelto di realizzarne una versione approssimata, che permette prestazioni nettamente migliori senza degradare in maniera eccessiva la qualità dell'immagine risultante.

L'approccio utilizzato nel caso del TerraMax prevede di andare a calcolare, per ogni punto dell'immagine IPM risultante, le coordinate del corrispettivo punto dell'immagine sorgente; nella versione approssimata, invece, il calcolo viene effettuato soltanto per un punto

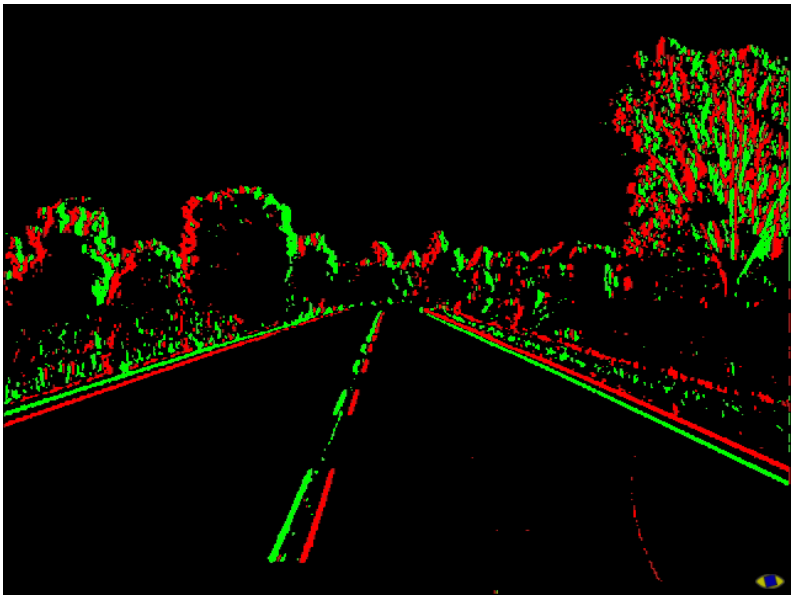
[2] mosaico di filtri colorati disposti sopra il sensore per catturare l'informazione relativa al colore

[3] Dark-Light-Dark-Light-Dark, ovvero un succedersi di punti scuri, chiari, scuri, chiari ed ancora scuri ad una distanza fissata all'interno di una stessa linea dell'immagine IPM

ogni 16: i rimanenti valori vengono ottenuti per interpolazione lineare. Questo accorgimento permette una riduzione dei tempi di calcolo nell'ordine di 8/10 volte, dal momento che il calcolo completo di ciascun punto richiede l'esecuzione di una divisione *floating point*, che non è supportata in modo nativo dal processore a disposizione, e viene dunque emulata via *software*.



(a)



(b)

Figura 8.1: Il risultato ottenuto applicando il filtro di Sobel; in verde sono rappresentate le transizioni scuro-chiaro, mentre in rosso quelle chiaro-scuro per i bordi a) orizzontali e b) verticali.

Figura 8.2: Regioni di interesse: in rosso –per la generazione dei *cluster*; in verde –per la validazione del punto di fuga; in blu –per la generazione dell'istogramma utilizzato nella fase di *tracking*.

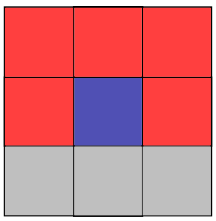


Figura 8.3: Maschera per la creazione dei *cluster*: in blu, il *pixel* di riferimento; in rosso, i punti che si tenta di andare ad inglobare.

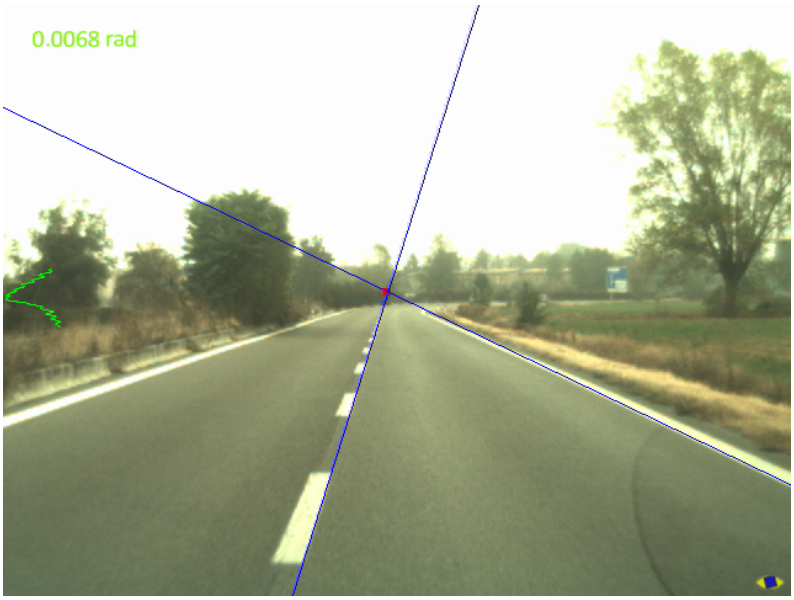


Figura 8.4: Componenti del *pitch detector*: in blu, le linee dirette verso il punto di fuga; in rosso, il punto di fuga stimato; a sinistra, in verde l'istogramma dei bordi orizzontali attuale, ed in rosso quello relativo al fotogramma precedente.

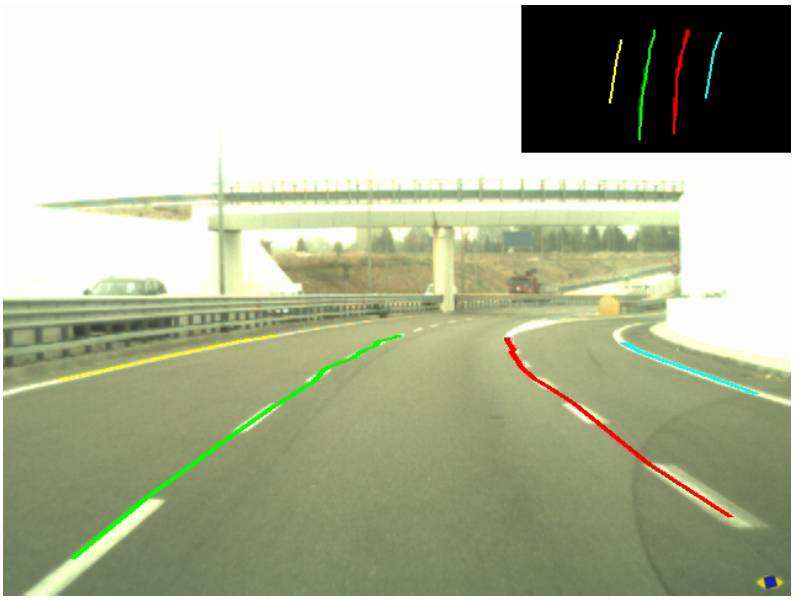


Figura 8.5: Il risultato prodotto dal *lane detector*. I colori codificano la posizione delle linee rilevate rispetto al veicolo: in verde, la prima alla sinistra; in rosso, la prima alla destra; in giallo, la seconda alla sinistra; in azzurro, la seconda alla destra.

Capitolo 9

Conclusioni

Il prototipo realizzato ha mostrato che l'approccio scelto è senz'altro adeguato a scenari extraurbani, ma fornisce anche prestazioni accettabili in aree più trafficate e complesse, come i centri cittadini. Installazione ed utilizzo risultano già in questa prima fase molto semplici: dopo una prima fase di calibrazione, che sfrutta il dispositivo stesso per acquisire una piccola sequenza di immagini, il sistema non richiede altri interventi, ed è attivo subito dopo l'accensione. I risultati possono essere visualizzati collegando uno schermo, oppure acquisiti via rete sotto forma di una descrizione della geometria della strada ad intervalli di distanza fissati, come esemplificato in Fig. 9.1; questa scelta permette una più semplice codifica dei valori ottenuti, agevolandone la trasmissione con i protocolli tipici dell'ambito *automotive*, come ad esempio quello CAN[1].

Il sistema è in conclusione promettente, e le componenti *hardware* selezionate si sono rivelate adeguate: a questo punto risulta però necessario procedere ad una ulteriore ingegnerizzazione del prodotto, fino a rendere dimensioni e costi compatibili con le esigenze del mercato.

Resta in ogni caso spazio sia per alcuni miglioramenti decisivi all'usabilità, come la capacità di avere una calibrazione che avvenga in modo del tutto automatico, senza alcun intervento da parte dell'utente, che per interventi dall'esito più incerto, come lo sarebbe la sostituzione dell'algoritmo di *pitch detection* con un sensore fisico; in questo caso andrebbero valutate sia le prestazioni in termini di precisione di rilevamento e sensibilità al rumore introdotto dalle normali condizioni di marcia (con buche e fondi sconnessi che possono falsa-

[1] Controller Area Network

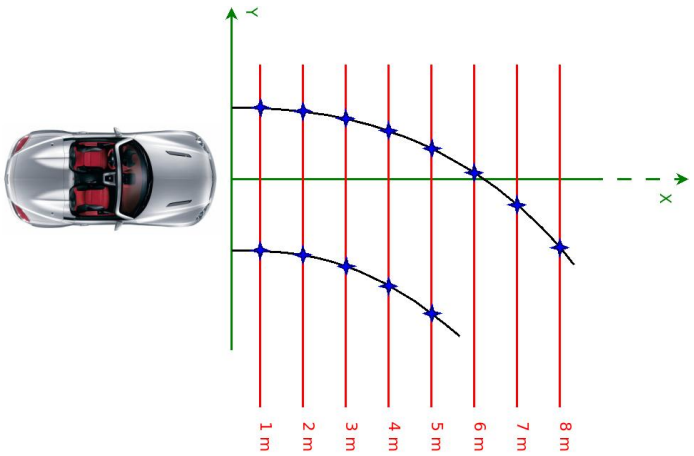


Figura 9.1: L'*output* prodotto dal sistema: per ciascuna delle linee individuate vengono restituiti i valori della coordinata Y in corrispondenza di valori di X fissati.

re i risultati), sia costi e vincoli di installazione, per determinare se una soluzione del genere possa risultare conveniente.

Sarebbe infine utile realizzare uno strumento che permetta di valutare in modo sistematico le prestazioni dell'algoritmo, per poter disporre una stima analitica dell'impatto di ciascuna modifica introdotta: utilizzando sequenze di immagini in cui la disposizione delle linee della carreggiata sia stata annotata manualmente diventa possibile misurare sia la percentuale dei riconoscimenti effettuati con successo sia della loro precisione.

Appendice A

Obstacle detection on the TerraMax autonomous vehicle

The cameras of the trinocular system are mounted on top of the cabin, with an asymmetrical configuration that generates three possible baselines (i.e. the distances between a pair of cameras). This solution, successfully tested during the DARPA Grand Challenge 2005 [13], provides a substantial degree of flexibility, satisfying the different perception needs which arise during autonomous navigation.

The processing begins with the conversion of the raw Bayer patterned frames acquired by the cameras to subsampled grayscale ones; these images are then rectified for stereo processing, correcting small hardware misalignment of the cameras, so that the corresponding epipolar lines become horizontal, and allowing more precise measurements. A special derivative filter [12] –useful to highlight ground features– is then applied to the images coming from the largest baseline and the resulting output is used to compute the so-called V-Disparity image [27]. The V-Disparity image is then analyzed to estimate the current vehicle pitch, as described in [13]. Fig. A.2 shows a sample V-Disparity image, with the detected ground correlation line highlighted in red, while the computed pitch value (in radiants) appears on the top-right corner.

The following step is the selection of one baseline out of the three available to be used for the generation of a Disparity Space Image: shorter baselines provide less noisy results, which are useful in low-speed manoeuvres, while longer ones allow a deeper perception range, thus permitting the detection of obstacles while they are still far away from the vehicle.

Disparity Space Image computation is carried out using a standard correlation-based approach, with SAD (sum of absolute differences) as the associated cost function, which allows very efficient implementations; the matching phase is performed in an incremental fashion, thus becoming independent of window size [23]. This algorithm has been improved by searching for matches along the epipolar line only for disparity values greater than the one expected for the ground (which varies along the image rows): this approach enhances the quality of the resulting map, with less spurious matches corresponding to disparities below the ground, but increases the overall complexity, since it generates a number of border cases that need to be carefully addressed. The resulting disparity map is shown in Fig. A.3.

Figura A.1: A sample frame acquired by the center camera.

Both the V-Disparity image and the Disparity Space Image are computed exploiting the parallelization opportunities offered by the target hardware: at coarse level, input images are splitted into stripes and processed in parallel by independet threads on different CPU cores, while at a more fine level the window matching algorithm exploits the available SIMD capabilities through the use of the MMX and SSE instruction sets, operating on multiple pixels and rows (V-Disparity image) or multiple disparity values (DSI) at the same time.

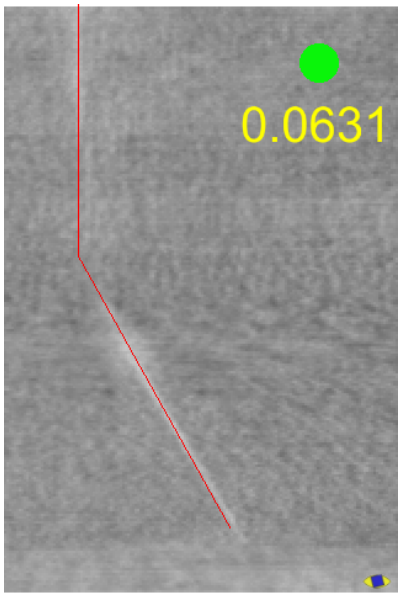


Figura A.2: V-Disparity image with pitch estimation.

Once the disparity map has been computed, along with the corresponding 3D world points, obstacle detection can take place. First, most of the elements belonging to the ground are removed, as contiguous pixels laying on the terrain surface have decreasing disparity values when analyzed from the bottom to the top of the image; remaining pixels are grouped column-wise into layers with similar disparities. One of the limitations of the original obstacle detection algorithm [16] (originally conceived for the DARPA Grand Challenge 2005, and used as a starting point for the development) was its inability to cope with multiple obstacles appearing in the same image column: in this case the only detected obstacles was the largest.

The use of multiple layers during the grouping process removes this limitation, allowing to cope with situations like the one depicted in Fig. A.4, where a number of queued cars become stacked together by the perspective projection.

The compactness of each group is evaluated in order to discard elements which are too spread vertically to represent an obstacle; surviving candidates receive a score proportional to the number of pixels in the group, and the score is increased if it is adjacent to groups with similar disparity. This value is then multiplied by the group average distance from the camera,

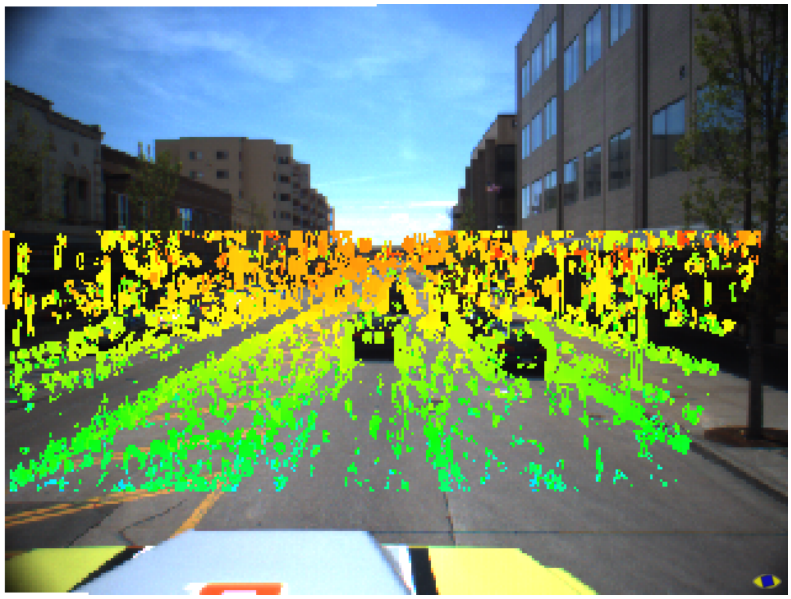
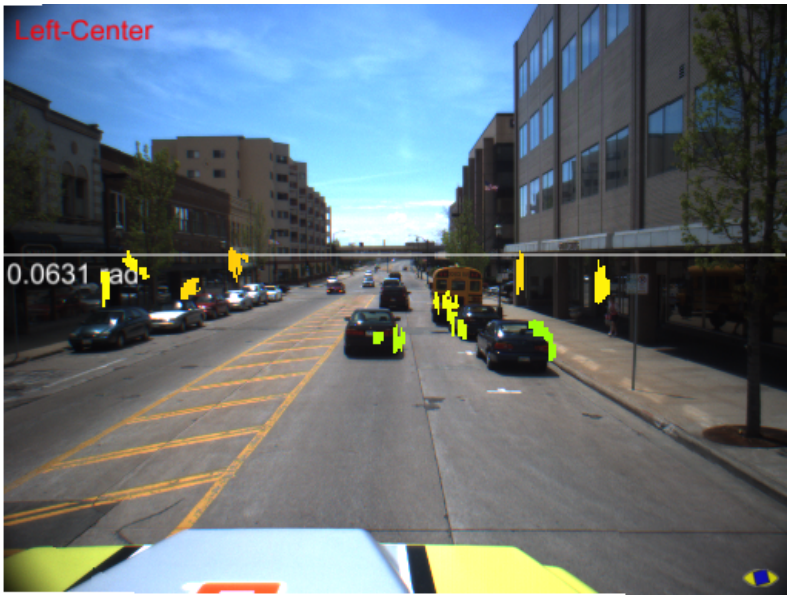


Figura A.3: Disparity Space Image.

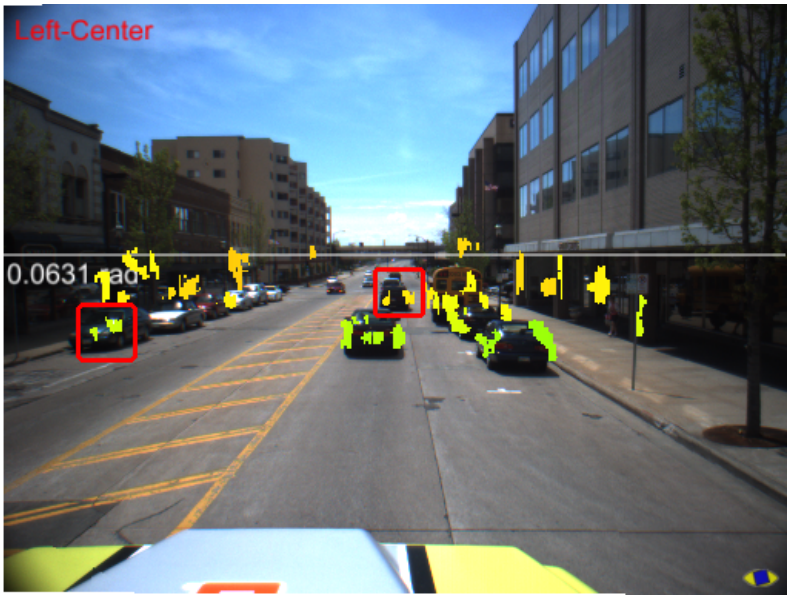
so that the scores of objects placed in different positions become comparable. At the end of this process, a threshold on the score is applied to select only the elements which represent actual obstacles for the truck.

To improve system detection rate in poorly textured areas a data fusion approach has been devised, exploiting information coming from both the cameras and the front LIDARs. The first step consists in filtering LIDAR data in order to remove the main ground components; next for each LIDAR world point the associated disparity value and image coordinates are computed using sensors calibration. The result is overlayed onto the previously computed Disparity Space Image, generating a richer starting set for a final flood-fill stage, where these seed values are expanded, merging close pixels with similar disparity values. It should be noted that despite being originated by completely different sensors, seeds can be fused together since after the respective filtering processes they provide comparable information: a few points corresponding with a high confidence degree to obstacles. The flood-fill step is performed onto the original (not filtered) Disparity Space Image using the computed seeds, and resulting clusters are filtered to remove spurious ones (those too small to represent any obstacle).

A final, high-level tracker assigns to each detected element id, age, and speed information.



(a)



(b)

Figura A.4: Detection improvements using multiple layers: (a) obstacles identified using a single layer, and (b) the same frame processed using 3 layers, with some of the differences circled in red.

Appendice B

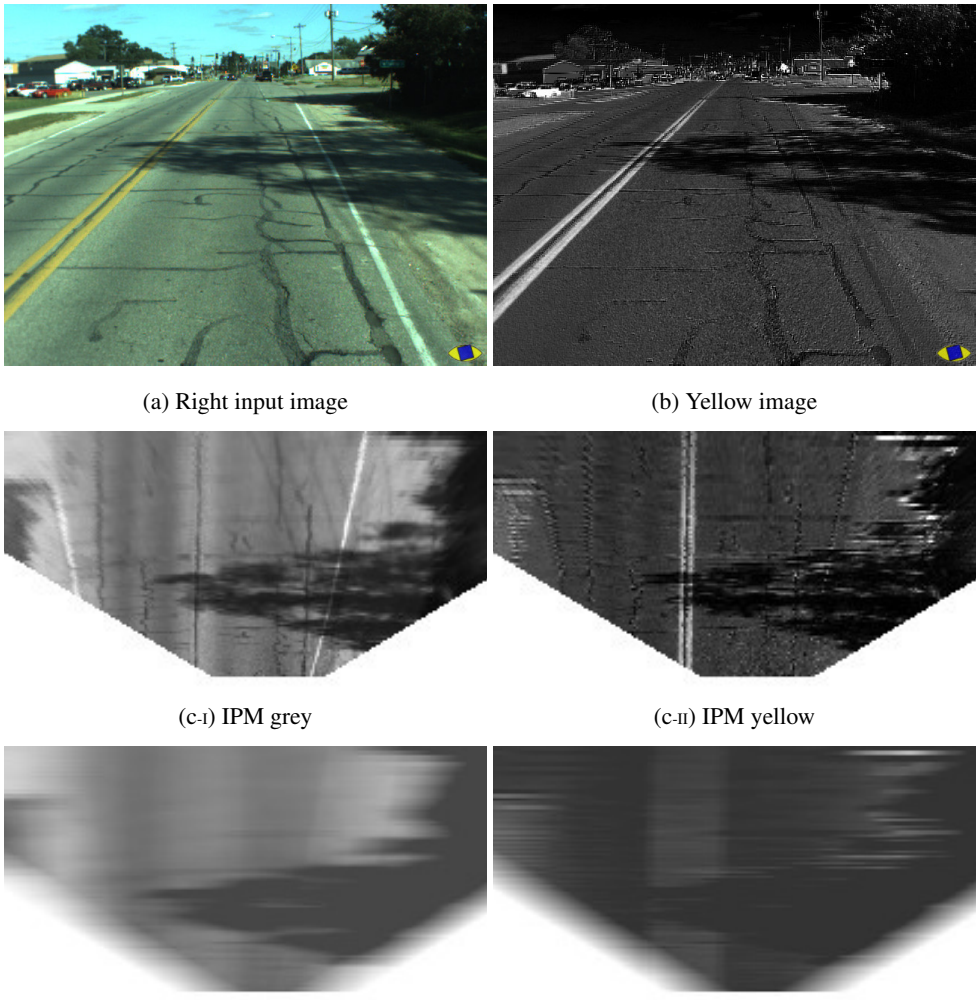
Lane detection on the TerraMax autonomous vehicle

Lane boundaries knowledge has been essential to allow the autonomous vehicles to safely navigate the roads of the Urban Challenge. Knowledge about the road structure permits to:

- follow roads whose a priori knowledge is incomplete (sparse waypoint following);
- correctly choose the lane to navigate in, without interfering with traffic in the opposite direction;
- switch to other lanes to avoid obstacles and during overtaking maneuvers.

The rules of the Urban Challenge [1] described:

- Three possible lane markings (lines) kinds:
  - Double Solid Yellow lines (coded as *SYSY* in the results), used to mark the edge between adjacent lanes with opposite traffic direction.
  - (Single) Solid White lines (*SW*), to mark road side boundaries; this line marking was absent in most of the course, substituted by the presence of a curb or a barrier.
  - Dashed White Lines (*DW*), to divide adjacent lanes with same direction of traffic.
- Three possible structures for a road segment:



(a) Right input image

(b) Yellow image

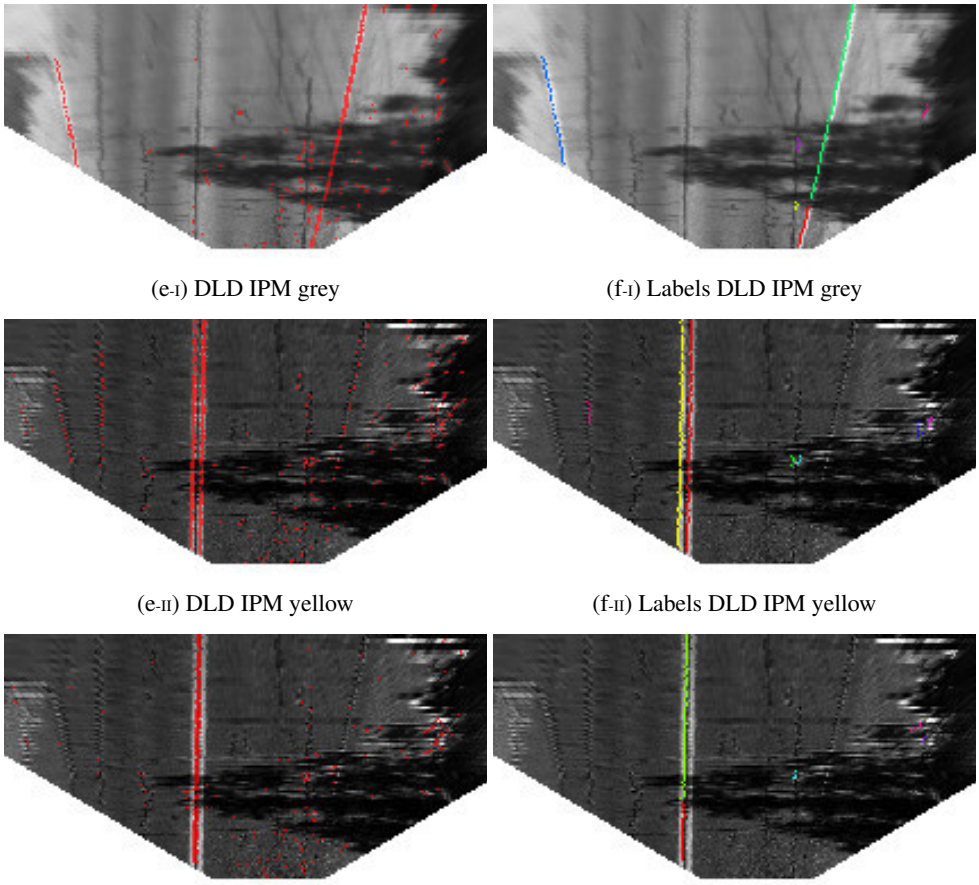
(c-I) IPM grey

(c-II) IPM yellow

(d-I) IPM grey averaged

(d-II) IPM yellow averaged

Figura B.1:



(e-I) DLD IPM grey

(f-I) Labels DLD IPM grey

(e-II) DLD IPM yellow

(f-II) Labels DLD IPM yellow

(e-III) DLDLD IPM yellow

(f-III) Labels DLDLD IPM yellow

Figura B.1: Low level processing of the lane detector for the trinocular system: a special case with diverging lane markings.

- One way road with two lanes: SW at the sides, DW in the middle.
- Two ways road with two lanes: SW at the sides, SYSY in the middle.
- Two ways road with four lanes, two for each traffic direction: SW at the sides, SYSY in the middle, DW to divide adjacent lanes with same direction of traffic.

A fourth line type, namely Single Solid Yellow (SY), to be used in replacement and with the same meaning of double yellow lines, was also utilized to ease the construction of the site visit tracks of the teams involved [2]. In designing TerraMax, it was decided to obtain information about road structure through vision based lane detection. At a first stage, the task of lane detection was assigned only to the trinocular system, also exploiting color, given the fact that it carries important information. In order to obtain an extended perception range, during lane detection the images were processed at full resolution (1024×768), obtaining color images through conversion of the Bayer input images. This conversion causes an additional computational cost, that cannot be delegated to the cameras because the system proved to suffer from the increased bandwidth request. Anyway this problem is mitigated by the fact that lane detection is performed using just one of the three input images.

From the input color image (Fig. B.1-a), two images are initially extracted:

- A brightness image (*grey image*), for white lines detection.
- An image encoding higher values for pixel whose hue is closer to yellow (*yellow image*, Fig. B.1-b), for yellow lines detection. Taking the 3 color components Red, Green, Blue, in the yellow image the pixels values are set as:

$$pixValue = \min\left(K\frac{\min(R,G)}{\max(B,B_{th})}, 255\right) \tag{B.1}$$

with K constant, and B_{th} a threshold used to avoid random values caused by noise in shadow regions.

The following low level processing is then computed in parallel on both images (Fig. B.1).

The algorithm looks for patterns eligible to be lane marking sections, i.e. for Dark-Light-Dark [31] (DLD) transitions. In order to have constant DLD patterns widths, allowing simplifications and computation load benefit, an Inverse Perspective Mapping is applied to the image (Fig. B.1-c). To accomplish this task, the instanteous pitch value obtained previously (Sect. A) is used; it should be noted that to obtain the pitch value no parallel lane markings assumption is made, so that cases of diverging lines, as the example of Fig. B.1, can be handled. The IPM in use maps a region 25 m wide, up to a 40 m maximum distance in a

333×120 pixels image: thus pixels do not represent square regions of the real world. This is made in order to obtain a trade off between DLD patterns intelligibility and computational costs.

DLD transitions for each image row are then extracted[1] (Fig. B.1-e); to binarize DLD values, a the algorithm uses a local threshold, proportional to the local brightness. This allows to detect DLD patterns also in shadow regions. The local brightness is estimated as the average value of pixels in a horizontal neighborhood. Note that computing average on an IPM image corresponds allows to average brightness over real world regions of constant area. In order to ease the detection of particularly close to each other double lines, a DLDLD pattern extraction is also performed (Fig. B.1-e-III).

In order to delete false DLD patterns generated by objects, which indeed are not laying on the ground, a mask, created using the output from stereo obstacle detection (Sect. A), is applied at this stage. The resulting data is then clustered in labels (Fig. B.1-f), discarding too short elements.

Length, position and orientation of labels are examined in order to obtain the output road model, joining compatible segments to get candidate lines. Since pixels of the IPM image do not represent square real world regions, at this stage label points are converted in real world coordinates, so that lengths, curvatures, and orientations of labels become directly usable. The road model is then built in three steps:

- Identification of a dominant line, named *mother line* (Fig. B.2-a, in purple), chosen considering:
  - length;
  - reliability (e.g. double lines are considered more reliable);
  - persistence over time.
- Classification of remaining segments as possible pieces of other up to 4 distinct lane boundaries, depending on the distance from the mother line.
- Creation of the lines at the side of the mother line, and construction of the output model (Fig. B.2-b and B.2-c), categorizing left (in green) and right (in red) edges of the lane that the vehicle is currently navigating, and boundaries of adjacent lanes.

Finally, the introduction of lines tracking lead to increased stability and reliability of the results, particularly in case of dashed lines.

[1] On the analogy of Sobel naming conventions, these features are considered *vertical*.

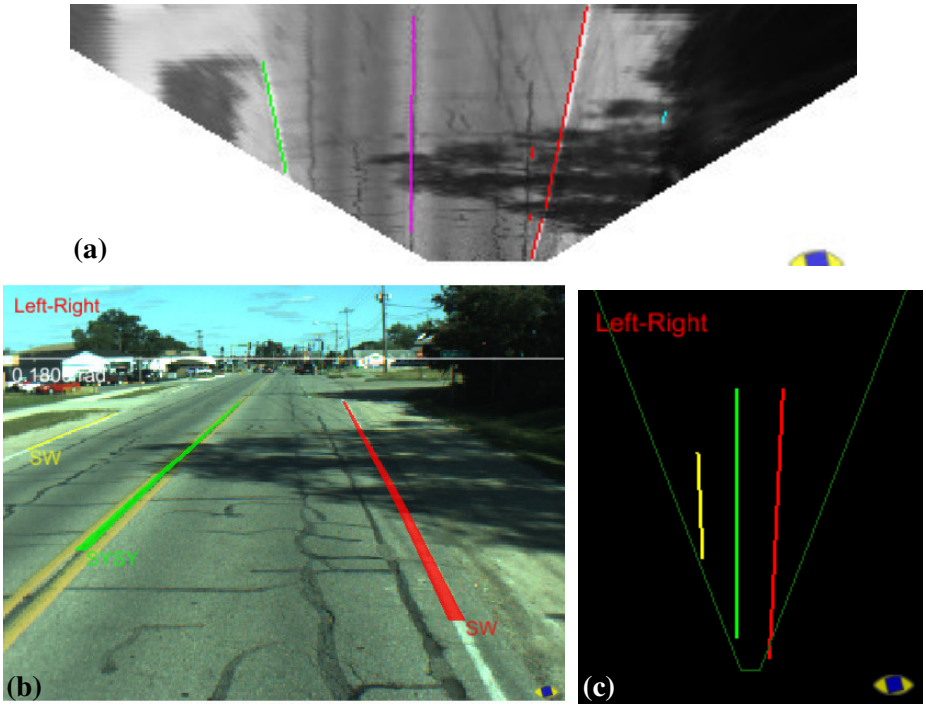


Figura B.2: (a) Categorization of labels based on compatibility with mother line (plotted in purple): compatible labels at the left (green), right (red), far left (yellow) or far right (cyan) of the mother line. (b-c) Final results plotted on input image and bird's eye view map. Green: close line at the left. Red: close line at the right. Yellow: far line at the left. SYSY: double solid yellow. SW: (single) solid white.

Appendice C

Evaluation of Monocular Image Stabilization Algorithms for Automotive Applications

Abstract

The performance of many computer vision applications, especially in the automotive field, can be greatly increased if camera oscillations induced by movements of the acquisition devices are corrected by a stabilization system. An effective stabilizer should cope with different oscillation frequencies and amplitude intervals, and work in a wide range of environments (such as urban, extra-urban or even unstructured ones). In this paper we will analyze three different approaches, based on signature, feature, and correlation tracking respectively, that have been devised to face these problems.

Introduction

Camera oscillations can be a serious problem in many computer vision applications, and this fact becomes particularly evident in the automotive field, where an acquisition system installed on a vehicle suffers from its movements. For instance, when driving on urban roads and highways, considerable pitch variations can be noticed, while in off-road and unstructured

environments every camera orientation angle can be largely affected by the road roughness. Such kind of movements cause continuous variations of the calibration parameters which many algorithms (like those employed in pedestrian detection [9] [8], obstacle detection [30] or lane detection [7]) use to associate the real 3D world position to the features they extract. Figure C.1 plots the error percentage that would affect distance estimations, if the pitch variation angle (with respect to the initial calibration) were neglected. Let p be the pixel corresponding to the projection of the real-world point $(d,0,0)$ in a camera with an instant pitch of $\theta' = \theta_{calib} + \Delta\theta$, and height h from the ground; if no stabilization is performed, the use of the sole calibration pitch value to compute its distance leads to:

$$d_{estim} = \frac{h}{tan(atan(\frac{h}{d}) + \Delta\theta)} \tag{C.1}$$

and thus to an error of:

$$error = \frac{|d_{estim} - d|}{d} \tag{C.2}$$

Many stabilization systems have been devised to cope with this problem, and while their goal is the same - i.e. to compensate for camera movements - the approaches used can be substantially different, depending on the application field they are targeted to.

State of the art in video stabilization

One way to reduce the acquisition system vibrations is to install it on an electromechanical stabilization platform [32]. This approach is quite effective in suppressing high frequency noise, but the performances are generally poor at lower rates, such as in the $1-25\,Hz$ interval (which unfortunately is the one where most of the noise involved with vehicle driving is concentrated in). Besides this, the device itself is usually quite cumbersome and expensive, so other types of stabilizers are more commonly employed. Another viable solution is to exploit the movement of a set of lenses to achieve the desired stabilization [17]. While there is the clear advantage of having no image quality loss, since this process directly acts on light incidence angle on the camera sensor, still it must be taken into account the fact that broad movements cannot be dealt with, as the correction scope is limited by physical factors; this approach seems therefore more suitable for other kinds of applications, such as in handheld cameras.

Digital image stabilization algorithms do also exist, and have shown to be capable of providing good results in automotive applications. The big advantage with this class of approaches is that no extra machinery is needed besides the acquisition device itself, since all

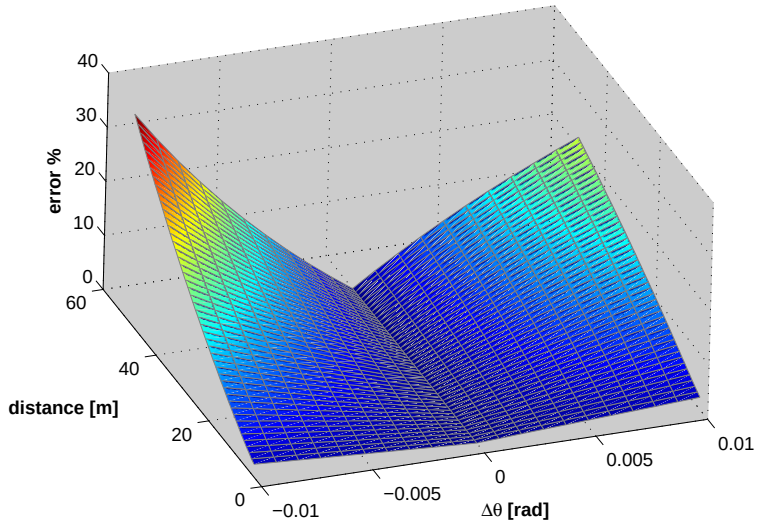


Figura C.1: Distance estimation error for a camera whose pitch has changed by $\Delta\theta$; camera height is 1.8 m. The surfrace is lacking symmetry because when the image gets shifted up by p pixels (that is, for negative values of $\Delta\theta$) real world points distance estimations are greatly increased, while the opposite movement makes the camera look farther, leading to smaller variations in the predicted values.

the desired corrections are performed within the image processing hardware, which can either be dedicated or general purpose (in which case the work is done at software level). This usually leads to a more compact and robust design, and possibly to lower costs in large scale production. The resulting system is also more customizable, as its behaviour can be modified to exactly fit the requirements of the application it is used in, while mechanical devices can hardly offer such flexibility. On the other hand these stabilizers have two main drawbacks: they cause information loss (only a limited region of the original image is available for further processing, and pixel interpolation might be used, thus reducing image quality), and introduce a certain delay.

A first example from this category of algorithms is stereovision, which can be exploited to estimate the vehicle pitch at the time of acquisition, as it has been shown in [12]. A pair of stereo images allows to estimate the ground coordinates with respect to the cameras at the

time of acquisiton, through the V-Disparity image approach, first presented in [26]. Once this information is known, and given the static camera orientation information, pitch detection becomes a trivial task. Stereovision has proven to be effective in both urban and unstructured environments, and represents a viable realtime solution whenever two or more synchronized cameras are available. The main problem arises from system calibration, whose accuracy becomes of paramount importance, especially with large baselines.

Algorithms based on motion vectors have also been proposed [25]. These algorithms try to determine global frame motion starting from local motion vectors, and deciding whether such movement has been produced by undesired oscillations or not. This approach has shown to be fast and effective, but it is quite hard to employ in the automotive field, where the presence of ego-motion leads to poor results.

In urban contexts lane markings can be successfully exploited to recover camera orientation [28]. Such kind of algorithms can produce accurate outputs; nonetheless, it must be noted that they make strong assumptions on environment structure, thus resulting not suitable for use in all those situations where their requirements are not met.

Algorithms

In this section we will analyze and compare three more algorithms, expressly designed to be employed in automotive applications.

Signature-based stabilization

A signature, extracted from each acquired image, can be successfully used to estimate undesired image shifts caused by vehicle pitch: in [15] the horizontal edges histogram is used to generate a signature, whose changes are tracked over time, as shown in figure C.2. This algorithm is able to correct shifts in the current acquired image up to 10% of its height, without introducing any frame delay in the processing pipeline; its reduced execution time makes it suitable for real time video applications. In figure C.3 is presented a sample frame and the derived signature.

While this algorithm has proven to be quite effective in many situations, still it must be observed that objects with a significant horizontal structure might induce it to apply unnecessary corrections, since they produce evident peaks inside the signature; slope changes are also hard to manage, as the signature extracted in such cases drifts in a preferred direction, making correct shift estimation impossible: this is an intrinsic weakness of vision-based sta-

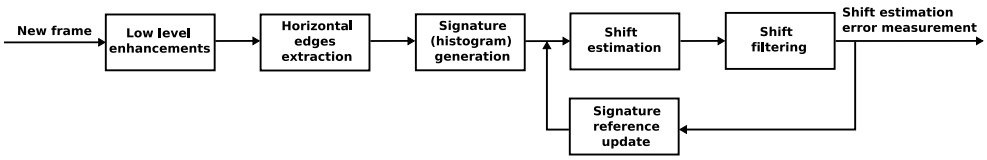
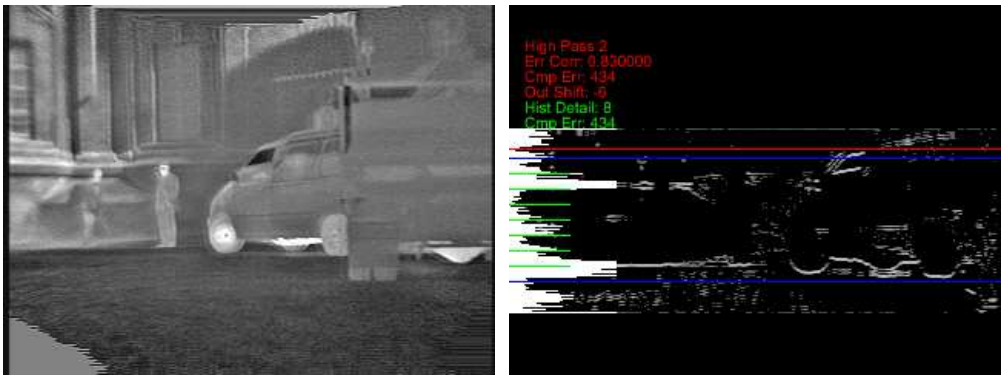


Figura C.2: Signature stabilization algorithm processing flowchart.



(a) (b)

Figura C.3: A sample processing step of the signature stabilization algorithm. (a) Far infrared input image. (b) Output: blue lines indicate the region of interest, while green lines indicate features detection areas; the intensity histogram related to edge image is depicted on the left-hand side, computed values are reported in the top-left hand side.

bilizers, which cannot know if such kind of oscillations should be ignored, unless inertial data is provided by other means. Best performances are obtained with input images grabbed from far infrared (FIR) cameras, as they have higher contrast if compared to visible ones, making the signature more easily trackable across frames.

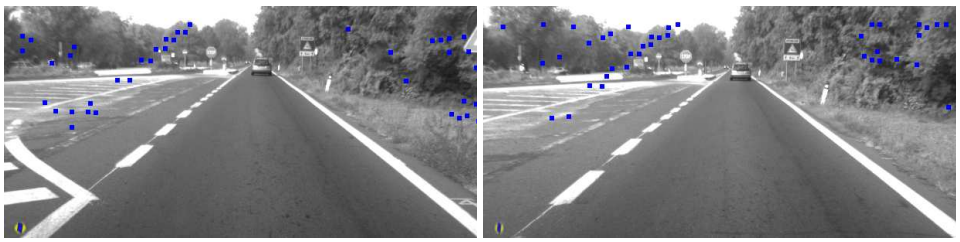
Features-based stabilization

The tracking of image features along the frames of a video sequence can also be employed for the detection of acquisition system movements [18]. A feature is a small region of the image characterized by some desired property, such as a high standard deviation in the spatial intensity profile, or a zero-crossing in the intensity function laplacian. Features have also been identified as lines, corners [29] or even through a wavelet-domain transform [22]. In [36] features are defined as image tiles which provide the best results with a given tracking algorithm; this approach is optimal by definition, and even if common sense would suggest that the former criteria should be preferred as they seem to be able to give reasonably good results, still the latter is more appropriate, as it makes no implicit assumption of independence between features localization and tracking.

Exploiting the features selection algorithm presented in [36] for automotive applications can lead to fairly good results, provided that some specifical issues related to this kind of use are correctly addressed. First, the cross-correlation between two frames out of a sequence of images acquired from a moving vehicle can be low, because of the speeds involved with driving: this means that an image cannot be taken as a reference for many frames. Another problem is that, when grabbing frames from a camera placed onto a vehicle, depth information is not neglectable, making the feature tracking task harder to perform than it would have been if the whole scene could be approximated to a planar surface: the points of two successive frames cannot be related through a simple rotoranslation. An example of how the described issues impact on features selection and tracking can be seen in figure C.4.

Processing time must also be kept under control, given the real-time nature of this class of systems. Following these considerations, the stabilization algorithm looks for features inside each image, avoiding an explicit tracking; instead it assumes that in two successive frames almost the same details are selected as features, and that they appear in similar columns. Figure C.5 shows that features pairing is performed considering the x-axis position of each feature, so that a correspondence is considered to be found when for two of them (one from the frame taken at time $T-1$ and one at time T) the abscissa difference is under a certain threshold.

This process creates a set of paired features, which is then used to estimate the current pitch, through the analysis of the vertical offset in each features pair. In a typical scenario many different values may appear, but usually one tends to prevail, thus the mode is used to determine the global frame shift, as in (C.3), where S is the set of all the admissible shifts,



(a) (b)

Figura C.4: Features tracking in a extraurban environment. (a) First acquired image. Selected features are marked with small blue squares (b) The following image. The set of selected features is quite different, due to the small amount of correctly identifiable details.

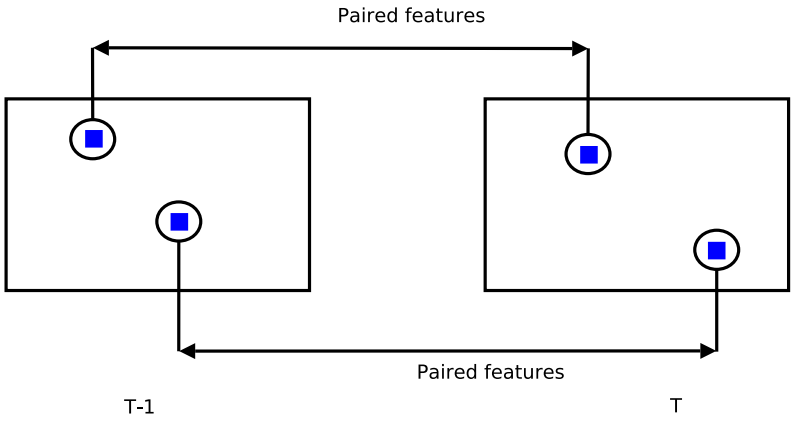


Figura C.5: Features pairing. The frames are acquired at times $T-1$ and T, then their details selected as features are paired according to their x-axis position

and $frequency(s)$ is the number of paired features with a vertical offset s.

$$shift = \bar{s} \in S \mid f = \max_{s \in S}(frequency(s)) \tag{C.3}$$

In some situations software stabilizers output can be incorrect, with effects spanning across several subsequent frames; a very simple way to cope with this problem is to insert in

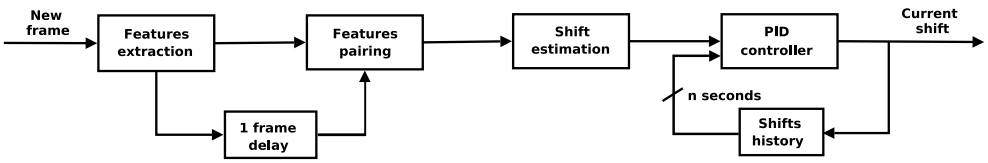


Figura C.6: Features-based stabilization algorithm processing flowchart.

the processing pipeline a PID controller, that brings the shift back to zero when its running sum becomes greater than a certain threshold (thus avoiding absolute value drifts), or when the estimated image offset remains constant for too many frames, as sometimes happens after a sharp bump, when the algorithm might stabilize its corrective action on a small - but still non-zero - output shift.

The complete features-based stabilization algorithm processing flowchart is the one depicted in figure C.6.

Urban environments are the ideal field of application for the described algorithm, as they are usually rich of easily-trackable details, while on extra-urban roads the performance is lower, given the presence of large plain-textured areas; sharp turns can also be hard to manage, since the acquired images content changes very quickly. Features-based stabilization has also another disadvantage: it is computationally expensive, with a processing time that heavily depends on the number of tracked features. While some fast implementations of features-based algorithms exist [22], in this case the analysis of a lower number of details seems more suitable to obtain a faster execution, albeit at the cost of slightly deteriorated results: a more reliable stabilization still needs higher time consumption. A good compromise for images with a resolution ranging from 750×400 up to 640×480 pixels in both urban and extra-urban contexts is of about 10 features per image.

Difference-based stabilization

The inherent complexity of features-based stabilization, with a performance bound to the environment characteristics and a high processing time, makes it clear that a simpler and more robust approach is necessary to achieve better results.

Starting from the usual consideration that the content of two successive frames doesn't normally change too abruptly, it follows that even in presence of significant camera oscillations it is still possible to find, for some vertical shift, a high correlation between the acquired images, as it has been shown in [29]. A good correlation criterion is to compute the sum of

absolute values of the differences among the corresponding pixels between a reference image (named r), taken at time $T-1$, and the current one (named c), taken at time T, shifted vertically by s pixels; this procedure is summarized in (C.4), where $d(s)$ denotes the difference at a given shift, $h\_min = \max(s,0)$, $h\_max = \min(height, height - s)$, and $n = h\_max - h\_min$:

$$d(s) = \frac{1}{n} \sum_{y=h\_min}^{h\_max} \sum_{x=0}^{width} |r[x][y-s] - c[x][y]| \tag{C.4}$$

The selected shift will be the one which minimizes the difference between the two images, thus producing the best possible match: this leads to (C.5), where S denotes the set of all admissible shift values:

$$shift = \bar{s} \in S \mid \bar{d} = \min_{s \in S}(d(s)) \tag{C.5}$$

The described operations are illustrated in figure C.7.

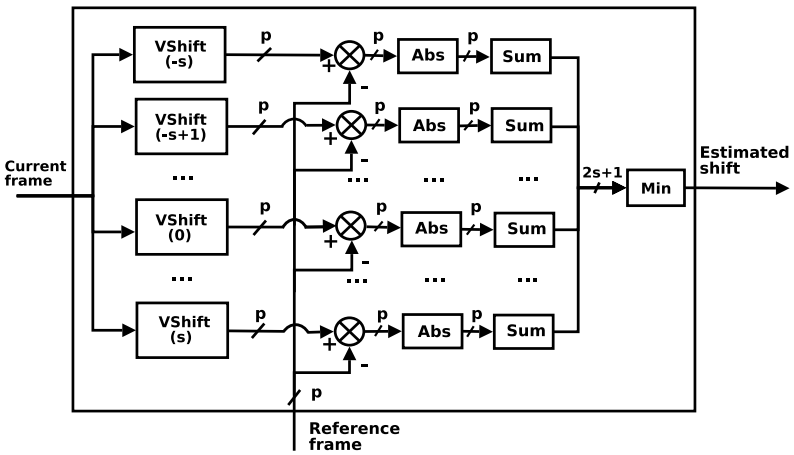
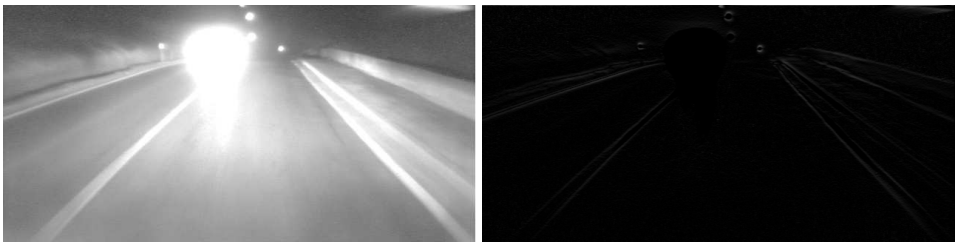


Figura C.7: Details on how the shift estimation is performed; p is the number of analyzed pixels, while s is the maximum offset applied.

It is possible to effectively compute the desired shift even applying some subsampling to the input images (for example processing one pixel every four), obtaining fast execution times, which make this algorithm suitable for real-time applications; results stability is granted, as it happens for the features-based stabilizer, by a PID controller.

An horizontal Sobel filter can be inserted at the very beginning of the processing pipeline to enhance the results in urban contexts, which are rich of details; this step is nevertheless optional, and can even cause some information loss in certain situations, such as the one depicted in figure C.8, where the frame has been grabbed at night: in these conditions the preprocessing step is useful to remove the visual artifacts introduced by lens flare, but it also completely eliminates the other car beams, that would have been an easy to match region.



(a) (b)

Figura C.8: Night stabilization. (a) An image acquired at night on an extraurban road. (b) The same image, after the application of a Sobel filter. Easily recognizable details, such as the car beams, have been removed by this kind of preprocessing.

A big advantage of the difference-based stabilizer over the features-based one is that the type of environment does not affect its performance, both in terms of results and processing time; moreover, no particular thresholds are needed to fine-tune its behaviour, and the only conditions that must be met in order to have it working correctly are:

- **good frames correlation**: sharp turns or crossing vehicles can seriously impair the stabilization outcomes, since they produce images with loose vertical correlation; these events can however be spotted out, as they are usually associated with good correlation for some horizontal shift. Moreover, the processing overhead of this sanity check is neglectable, since it is effective even when performed on very few pixel rows.
- **a few objects inside the field of view**: this ensures that enough information is available to perform the matching; large, textureless areas - such can be wide extraurban roads where no lane markings are present - can prevent the algorithm from working correctly.

This minimal set of requirements allows a wide range of applications, with both visible

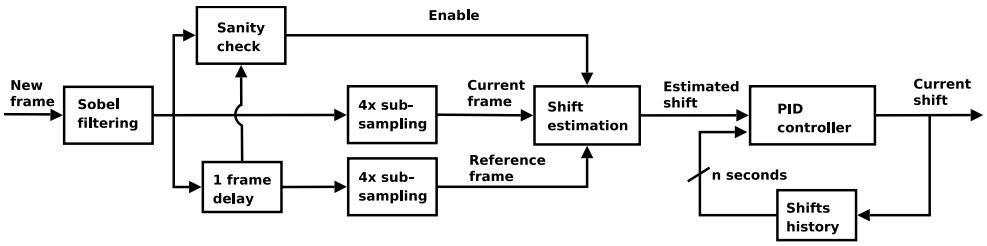


Figura C.9: Difference-based stabilization algorithm processing flowchart.

and infrared cameras; the complete processing flowchart for this algorithm is presented in figure C.9.

Results and Conclusions

The presented algorithms have been tested on sequences acquired both in urban and extra-urban environments, using three cameras simultaneously, in order to build a common evaluation set. The cameras were FIR, NIR and visible ones, with resolutions of 320×240, 768×288 and 640×480 pixels respectively. Figure C.10 contains a sample frame for each sequence, grabbed by the visible camera.

(a) (b) (c)

Figura C.10: Stabilization test sequences samples. The images have been acquired by a visible camera on (a) The close proximity of a bumper on a urban road. (b) A gravel road (c) A country road. Note the sign used in all the sequences as a reference for manual annotation

During the acquisitions, cameras have been set up with optical axes laying all on the same plane, parallel to the ground. A sign, mounted on a pole and aligned with the optical axes of all the cameras, has been used as a reference for performance evaluation: a perfect stabilization would keep it on the central row of the image, so the gap between its actual position and the ideal one has been manually determined for each frame. The results are presented in table C.1, where each cell contains error mean and variance for the given combination of camera and algorithm.

Tabella C.1: Stabilization algorithms performance

	Algorithm			
Camera	None μ, σ^2	Histograms μ, σ^2	Features μ, σ^2	Difference μ, σ^2
Urban Road				
FIR	-0.012, 75.91	-1.457, 26.72	0.931, 111.48	-0.815, 55.56
NIR	-5.337, 181.37	-8.723, 151.98	-8.224, 222.03	-4.733, 165.60
Visible	-2.608, 28.40	-5.026, 20.55	-3.487, 36.09	-6.173, 10.51
Gravel Road				
FIR	3.997, 43.31	-1.342, 30.84	2.838, 22.70	-0.245, 31.19
NIR	9.907, 92.08	1.604, 57.30	8.645, 101.37	4.040, 83.18
Visible	2.885, 12.66	1.474, 7.23	1.368, 21.32	1.493, 7.78
Country Road				
FIR	5.110, 57.36	6.530, 41.88	4.768, 62.96	4.453, 46.82
NIR	18.117, 149.99	19.078, 118.14	6.216, 222.00	15.513, 139.26
Visible	0.287, 10.28	-0.642, 5.62	4.967, 56.03	-0.693, 8.33

The signature-based algorithm and the features-based one have shown to provide the best results when employed to face specific problems, such as the stabilization of images in urban contexts; while the former can more easily manage frames grabbed with a FIR camera, the latter is more suitable to compensate the undesired oscillations in visible and NIR images. The difference-based stabilizer appears to be much more versatile, as it can handle both urban and

extra-urban environments without significant performance variations. Execution times for all the described algorithms have been compared on a commercial Pentium 4 2.8 GHz machine, equipped with 1 Gb RAM; the results are summarized in table C.2.

Tabella C.2: Execution times for the described algorithms on a Pentium 4 2.8 GHz machine

Algorithm	*Execution time* [ms]		
	Min	*Avg*	*Max*
Signature	19	20	23
Features	28	30	39
Difference	6	6	6

The most time-consuming algorithm is clearly the features-based one, which is also the less suitable for real time tasks, since an upper bound to its execution time cannot be easily determined; signature-based stabilization worst case execution time can instead be determined, albeit it can be quite different from the average one. Difference-based stabilization seems to be the best solution for real-time applications, since its running time is constant across frames, making it a good choice when it comes to deal with the constraints of the automotive field.

Bibliografia

[1] DARPA Urban Challenge Rules, 2007. Available http://www.darpa.mil/grandchallenge/rules.asp.

[2] DARPA Urban Challenge Technical FAQ, June 2007. Available http://www.darpa.mil/grandchallenge/faq.asp.

[3] Care - european road accident database, 2008. Available at http://ec.europa.eu/transport/roadsafety/road_safety_observatory/care_en.htm.

[4] A. Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison-Wesley Professional, Indianapolis, February 2001.

[5] M. Bertozzi, L. Bombini, A. Broggi, P. Cerri, P. Grisleri, and P. Zani. GOLD: A Complete Framework for Developing Artificial Vision Applications for Intelligent Vehicles. *IEEE Intelligent Systems*, 23(1):69–71, Jan.–Feb. 2008.

[6] M. Bertozzi, L. Bombini, P. Cerri, P. Medici, P. C. Antonello, and M. Miglietta. Obstacle Detection and Classification fusing Radar and Vision. In *Procs. IEEE Intelligent Vehicles Symposium 2008*, pages 608–613, Eindhoven, Netherlands, June 2008.

[7] M. Bertozzi and A. Broggi. GOLD: a Parallel Real-Time Stereo Vision System for Generic Obstacle and Lane Detection. *IEEE Trans. on Image Processing*, 7(1):62–81, Jan. 1998.

[8] M. Bertozzi, A. Broggi, R. Chapuis, F. Chausse, A. Fascioli, and A. Tibaldi. Shape-based pedestrian detection and localization. In *Procs. IEEE Intl. Conf. on Intelligent Transportation Systems 2003*, pages 328–333, Shangai, China, Oct. 2003.

[9] M. Bertozzi, A. Broggi, T. Graf, P. Grisleri, and M.-M. Meinecke. Pedestrian Detection in Infrared Images. In *Procs. IEEE Intelligent Vehicles Symposium 2003*, pages 662–667, Columbus, USA, June 2003.

[10] M. Bertozzi, A. Broggi, P. Medici, P. P. Porta, and A. Sjögren. Stereo Vision-Based Start-Inhibit for Heavy Goods Vehicles. In *Procs. IEEE Intelligent Vehicles Symposium 2006*, pages 350–355, Tokyo, Japan, June 2006.

[11] A. Broggi, A. Cappalunga, C. Caraffi, S. Cattani, S. Ghidoni, P. Grisleri, P. P. Porta, M. Posterli, P. Zani, and J. Beck. The passive sensing suite of the TerraMax autonomous vehicle. In *Procs. IEEE Intelligent Vehicles Symposium 2008*, Eindhoven, Netherlands, June 2008.

[12] A. Broggi, C. Caraffi, R. I. Fedriga, and P. Grisleri. Obstacle Detection with Stereo Vision for off-road Vehicle Navigation. In *Procs. Intl. IEEE Wks. on Machine Vision for Intelligent Vehicles*, San Diego, USA, June 2005.

[13] A. Broggi, C. Caraffi, P. P. Porta, and P. Zani. The Single Frame Stereo Vision System for Reliable Obstacle Detection used during the 2005 Darpa Grand Challenge on TerraMax. In *Procs. IEEE Intl. Conf. on Intelligent Transportation Systems 2006*, pages 745–752, Toronto, Canada, Sept. 2006.

[14] A. Broggi, S. Cattani, P. P. Porta, and P. Zani. A Laserscanner–Vision Fusion System Implemented on the TerraMax Autonomous Vehicle. In *Procs. IEEE Intl. Conf. on Intelligent Robots and Systems 2006*, Beijing, China, Oct. 2006.

[15] A. Broggi and P. Grisleri. A Software Video Stabilization System for Automotive oriented Applications. In *Procs. IEEE Vehicular Technology Conference*, Stockholm, Sweden, June 2005.

[16] C. Caraffi, S. Cattani, and P. Grisleri. Off-road Path and Obstacle Detection using Decision Networks and Stereo. *IEEE Trans. on Intelligent Transportation Systems*, 8(4):607–618, Dec. 2007.

[17] B. Cardani. Optical image stabilization for digital cameras. *IEEE Control System Magazine*, 26(2):21–22, Apr. 2006.

[18] A. Censi, A. Fusiello, and V. Roberto. Image stabilization by features tracking. In *Procs. Intl. Conf. on Image Analysis and Processing*, pages 665–667, Venice, Italy, Sept. 1999.

[19] E. Commission. Proposal for a regulation of the european parliament and of the council concerning type-approval requirements for the general safety of motor vehicles. Available at http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:52008PC0316:EN:NOT.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*, chapter 33.3, pages 949–955. McGraw-Hill Science/Engineering/Math, July 2001.

[21] COWI. Cost-benefit assessment and prioritisation of vehicle safety technologies. Technical Report TREN/A1/56-2004, European Commission Directorate General Energy and Transport, Jan. 2006.

[22] J. Derutin, F. Dias, L. Damez, and L. Allezard. SIMD, SMP and MIMD-DM parallel approaches for real-time 2D image stabilization. In *Procs. Intl. Workshop. on Computer Architecture for Machine Perception*, pages 73–80, Venice, Italy, July 2005.

[23] O. Faugeras, B. Hotz, H. Mathieu, T. Viéville, Z. Zhang, P. Fua, E. Théron, L. Moll, G. Berry, J. Vuillemin, P. Bertin, and C. Proy. Real-time correlation-based stereo : algorithm, implementations and applications. Technical Report 2013, INRIA, Aug. 1993.

[24] KfV, NTUA, SWOV, TRL. Annual statistical report 2008. Technical Report TREN/04/FP6TR/SI2.395465-506723, SafetyNet - European Road Safety Observatory, Jan. 2008.

[25] S. J. Ko, K. H. Lee, and S. H. Lee. Digital image stabilization algorithms based on bit-plane matching. *IEEE Trans. on Consumer Electronics*, 3:617–622, Aug. 1998.

[26] R. Labayrade and D. Aubert. A Single Framework for Vehicle Roll, Pitch, Yaw Estimation and Obstacles Detection by Stereovision. In *Procs. IEEE Intelligent Vehicles Symposium 2003*, pages 31–36, Columbus, USA, June 2003.

[27] R. Labayrade, D. Aubert, and J.-P. Tarel. Real Time Obstacle Detection in Stereo Vision on non Flat Road Geometry through "V-Disparity" Representation. In *Procs. IEEE Intelligent Vehicles Symposium 2002*, Paris, France, June 2002.

[28] Y.-M. Liang, H.-R. Tyan, S.-L. Chang, H.-Y. Liao, and S.-W. Chen. Video stabilization for a camcorder mounted on a moving vehicle. *IEEE Trans. on Vehicular Technology*, 53(6):1636–1648, Nov. 2004.

[29] L. Marcenaro, G. Vernazza, and C. Regazzoni. Image stabilization algorithms for video-surveillance applications. In *Procs. 3rd IEEE Intl. Conf. on Image Processing*, pages 349–352, Oct. 2001.

[30] S. Nedevschi, R. Danescu, R. Schmidt, and T. Graf. High accuracy stereovision system for far distance obstacle detection. In *Procs. IEEE Intelligent Vehicles Symposium 2004*, Parma, Italy, June 2004.

[31] S. Nedevschi, F. Oniga, R. Danescu, T. Graf, and R. Schmidt. Increased Accuracy Stereo Approach for 3D Lane Detection. In *IEEE Intelligent Vehicles Symposium*, pages 42–49, Tokyo, Japan, June 2006.

[32] J. Schiehlen and E. Dickmanns. Design and control of a camera platform for machine vision. In *Procs. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, pages 2058–2063, Sept. 1994.

[33] M. Schulze. Contribution of PReVENT to the safe cars of the future. In *13th ITS World Congress*, London, UK, Oct. 2006.

[34] J. P. Snyder and G. S. (U.S.). *Map projections–a working manual*. United States Government Printing Office, 1987.

[35] L. D. Stefano, M. Marchionni, S. Mattoccia, and G. Neri. A fast areabased stereo matching algorithm. *Image and Vision Computing*, 22:983–1005, 2004.

[36] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical report, Carnegie Mellon University, Pittsburgh, Pennsylvania, July 1991.

Ringraziamenti

Eccoci di nuovo all'ultima pagina, ma questa volta sarò breve, tanto già lo sapete quanto vi devo, e non c'è bisogno di tante parole per ricordarlo.

Il primo grazie va ad Alberto, che ha reso possibile tutto questo, e anche molto altro – sei un "capo" unico!

Grazie anche a tutti i ragazzi del laboratorio, e a quelli che dal laboratorio oramai se ne sono andati: con voi ho attraversato città e deserti fino ai confini del mondo conosciuto, ho scalato montagne, e vissuto giorni e notti che non dimenticherò mai. Più di ogni altra cosa, però, importa il fatto che ho trovato degli amici su cui contare.

Grazie infine a tutti quelli che da sempre mi sono vicini, e quanti ogni giorno entrano ed escono dalla mia vita: mi avete insegnato amore ed odio, e obbligato a conoscere me stesso e il mondo. A volte è dura, ma ne vale la pena se si vuole avere una storia da raccontare.