Università degli Studi di Genova

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Autolocalizzazione di robot mobile tramite riconoscimento di feature visive

Relatore:

Chiar.mo Prof. Renato Zaccaria

Correlatore:

Ing. Antonio Sgorbissa

Candidato: Simone Zanella

Anno Accademico: 2003/2004

Prefazione

La localizzazione è il processo che si occupa di determinare e tracciare la posizione di un robot nel proprio ambiente di lavoro. Tutti i robot mobili hanno bisogno di conoscere dove si trovino per poter operare autonomamente.

La percezione visiva rappresenta uno strumento molto utile per i compiti di localizzazione di un robot, poiché essa è in grado di influenzare significativamente le capacità dei sistemi robotizzati di interagire con l'ambiente circostante. Un robot, interpretando ciò che vede, valuta ed analizza il proprio funzionamento senza dover necessariamente entrare in contatto diretto con l'ambiente. Un sistema di visione è potenzialmente in grado di mettere in risalto una vasta quantità di errori, sistematici e non, che incorrono inevitabili durante la navigazione, contribuendo alle necessarie correzioni sul processo di localizzazione.

Obiettivo di questa tesi è stato la realizzazione di un sistema di autolocalizzazione di robot basato sul riconoscimento di *feature* visive. Le *feature* sono caratteristiche distintive e descrittive dell'ambiente. Un sistema di visione artificiale è in grado di interpretare la scena inquadrata da una telecamera ed estrarre da essa la posizione delle *feature* presenti. Il robot combina poi le informazioni ricevute, e tramite un Filtro di Kalman è in grado di correggere la stima della propria posizione allo scopo di localizzarsi correttamente.

Il lavoro oggetto di questa tesi è stato realizzato presso il Laboratorium, laboratorio di robotica mobile sito nel Dipartimento di Informatica, Sistemistica e Telematica dell'Università di Genova, applicando il sistema realizzato ad un robot autonomo, e

rendendo quest'ultimo in grado di autolocalizzarsi servendosi delle informazioni provenienti dalla visione artificiale.

Questa tesi è strutturata essenzialmente in due parti. Nella prima parte viene presentato il *background* tecnico alla base del progetto. La realizzazione di un sistema di navigazione robotica è sempre frutto dell'interazione tra diverse discipline, verranno così presentate e descritte le principali tecniche relative alla localizzazione robotica, all'estrazione di *features* da immagini digitali, e al filtraggio di Kalman. Nella seconda parte della tesi verrà invece descritto il sistema realizzato, presentando i risultati ottenuti nelle diverse prove sperimentali eseguite. Conclude la tesi un'appendice che approfondisce e completa le informazioni riguardanti il software di visione artificiale ed estrazione di *features* realizzato.

Ringraziamenti

Una tesi è sempre frutto di un intenso lavoro, durante il quale attraverso il supporto e la vicinanza di altre persone si riescono a superare le difficoltà e si trova la determinazione nel portare a termine il proprio progetto. Desidero quindi fare dei doverosi ringraziamenti a tutti coloro che hanno avuto un ruolo significativo nella realizzazione di questa tesi.

Vorrei innanzitutto ringraziare i miei relatori per avermi consentito di affrontare le problematiche relative a materie innovative e affascinanti quali sono l'elaborazione di immagini digitali e la robotica mobile.

Ringrazio anche tutto lo staff del Laboratorium che si è interessato al mio lavoro, fornendomi il supporto necessario a proseguire nello sviluppo del progetto. In particolare ringrazio l'Ing. Fulvio Mastrogiovanni per il costante impegno e aiuto fornitomi durante il lungo periodo di testing del sistema.

Desidero ringraziare anche i miei compagni di corso, in particolare Avio, Nico, Camix, Daniele e Beppe, con i quali ho condiviso gran parte della mia carriera universitaria, e siamo riusciti a superare insieme le difficoltà che ci si sono presentate.

Ringrazio anche tutti i miei amici che mi sono stati vicini, e con i quali ho trascorso la mia vita universitaria lontano da casa, in particolare Ale, Kiuzzo, Paolino, Fabio, Anna e Katia.

Infine ringrazio con affetto la mia famiglia, per avermi sostenuto e incoraggiato durante gli anni passati in Università.

Indice

Capitolo 1
Robotica e visione artificiale

1.1 – Robot autonomi

L'evoluzione della scienza robotica ha portato i robot moderni a raggiungere l'*autonomia* e la *mobilità*. La mobilità è la capacità di un robot di potersi muovere liberamente nel mondo. L'autonomia è la capacità di un robot di eseguire i propri compiti senza l'intervento umano. A seconda dei diversi gradi di autonomia e mobilità si possono identificare diverse categorie di robot: [COX]

- *non autonomi*: sono robot manovrati interamente tramite intervento umano. L'intelligenza di questi robot consiste nell'interpretare correttamente i comandi provenienti dall'esterno;

- *semi-autonomi*: sono robot che sono in grado di prendere decisioni autonome, ma hanno al tempo stesso la possibilità di essere manovrati dall'esterno;

- *completamente autonomi*: sono robot pienamente indipendenti, che non necessitano di alcun tipo di intervento esterno nella scelta del moto o delle azioni da compiere.

Il grado di autonomia in generale è strettamente legato all'utilizzo che viene fatto del robot. La non-autonomia non è un problema in ambito industriale, in cui il robot svolge spesso sempre la stessa azione, senza bisogno di dover effettuare delle scelte. In questi

casi il costo di realizzazione e di gestione del robot è decisamente minore rispetto a robot completamente autonomi. Per questi motivi si preferisce utilizzare robot autonomi solo per l'utilizzo in situazioni in cui non è possibile, o non è semplice, determinare le più appropriate sequenze di azioni da compiere per il raggiungimento di un obiettivo prefissato (*goal*). [COW]

Per conferire a un robot proprietà di autonomia e mobilità è generalmente necessario servirsi di strumenti e sistemi che consentano al robot di analizzare l'ambiente in cui lavora. Questa analisi avviene attraverso l'utilizzo di sensori, ovvero dispositivi in grado di rilevare segnali o condizioni fisiche. A fianco di sensori tipicamente tradizionali, quali ad esempio *sonar* ed *encoder*, ne vengono molto spesso impiegati di più moderni e complessi, in grado di fornire ai robot ulteriori e preziose informazioni sull'ambiente esterno.

Grazie al continuo miglioramento delle tecnologie e delle capacità di calcolo degli elaboratori, le tecniche di *visione artificiale* si sono gradualmente ritagliate un ruolo di notevole importanza tra i sistemi di percezione robotica. Le vaste potenzialità offerte dalla visione artificiale forniscono infatti molteplici possibilità di applicazione in robotica.

1.2 - La visione artificiale

La visione artificiale si pone l'obiettivo di automatizzare ed integrare una vasta gamma di processi e rappresentazioni tipiche della percezione visiva. [BAL]

La percezione visiva è essenzialmente un problema di elaborazione di informazioni a più livelli gerarchici, costituito da: [MAR]

- un *input* del sistema, ovvero l'immagine, trattata in forma matrice di valori relativi all'intensità luminosa dell'ambiente circostante;

- un primo stadio elaborativo di estrazione di informazioni elementari dall'immagine, che porta alla formazione del *raw primal sketch*, ovvero uno schema grezzo su cui compiere le analisi successive;

- un secondo stadio elaborativo, che completa ed espande l'informazione frammentaria ricavata dal *raw primal sketch*;

- uno stadio di elaborazione finale, che si astrae dal punto di vista della telecamera e cerca di interpretare la scena.

La natura gerarchica del problema è conseguenza della crescente complessità del processo, a partire dall'input fino alla generazione dell'output.

La complessità tecnica e il notevole carico computazionale richiesto dal problema sono stati affrontati dalla scienza robotica seguendo approcci differenti.

I primi studi di percezione robotica si fondavano sul principio della *percezione generalizzata*: il sistema di visione era concepito indipendente dal sistema generale di controllo e di gestione del robot; inoltre, era necessaria una ricostruzione 3D completa della scena per acquisire sufficienti informazioni da analizzare. Un tale approccio si rivelava però assolutamente inadatto a un utilizzo in applicazioni in tempo reale.

I limiti della percezione generalizzata indirizzarono la ricerca robotica verso la *percezione modulare*: il sistema di visione non è più indipendente dal resto del sistema, ma interagisce con

gli altri moduli del robot, scambiando e analizzando solo le informazioni strettamente necessarie. Questo approccio ha reso possibile l'utilizzo efficiente dei sistemi di visione nella robotica moderna.

1.3 – Robotica e visione

L'approccio della percezione modulare segue l'idea che l'integrazione di sensori di diverso tipo sia fondamentale per conferire a un robot proprietà di versatilità ed adattamento al proprio dominio di applicazione. Molto spesso si trovano infatti sistemi di visione artificiale affiancati a sistemi basati su sensori tradizionali. I sistemi di visione possiedono infatti il grande vantaggio di poter misurare l'ambiente senza dovervi entrare in stretto contatto.

In robotica i sistemi di visione cercano di riprodurre il senso della vista umana, e rendere il robot in grado di interpretare ciò che "vede". I sensori visivi acquisiscono una grande quantità di dati, rendendo i sistemi in grado di estrarre moltissime informazioni dall'ambiente circostante. Tuttavia, proprio la grande quantità di dati da processare, assieme alla complessità computazionale delle tecniche di estrazione di informazioni, gravano inesorabilmente sulle prestazioni generali dei sistemi stessi. E' stato il costante incremento della velocità degli elaboratori e il miglioramento delle tecnologie di acquisizione video a rendere possibile l'applicazione con successo dei risultati degli studi della scienza robotica, portando allo sviluppo di sistemi di visione realmente funzionanti ed efficienti nella navigazione in tempo reale.

L'interpretazione della scena circostante rende la visione artificiale uno strumento accurato per l'azionamento del robot.

Tipicamente i sistemi sensoriali e l'azionamento sono combinati secondo un modello ad *open loop*. Nel caso della visione, si segue il modello del "*prima vedo, poi mi muovo*". La visione artificiale è in grado di fornire a questo modello un controllo in *closed loop*, tramite un controllo detto di "*visual-feedback*", migliorando l'accuratezza dei risultati. Attraverso la fusione dei risultati ottenuti tramite l'elaborazione di immagini, la cinematica, la dinamica, la teoria del controllo e l'elaborazione di dati in *real time* si giunge alla creazione di architetture di *visual servoing*.

La robotica si serve delle architetture di *visual servoing* operando su quattro schemi principali : [WEI]

- *Dynamic look and move*: l'architettura di controllo fornisce i dati del sistema di visione come ingressi al *controller*, che stabilizza poi il robot attraverso un feedback sui giunti;

- *Direct visual servo*: viene utilizzato direttamente il sistema di visione per elaborare gli ingressi per i giunti, tramite un *visual servo controller*;

- *Position-based control*: attraverso la conoscenza dei modelli geometrici degli elementi del sistema, si stimano la posizioni del *target* rispetto allo strumento di acquisizione video utilizzando elementi distintivi (*features*) estratti dalle immagini acquisite;

- *Image-based control*: i controlli al robot sono elaborati direttamente dall'analisi delle *features* presenti nell'immagine.

In generale, la conoscenza del modello robot-ambiente consente di astrarre gradualmente i dati acquisiti dai sensori visivi, e ricavare

informazioni di grande qualità e accuratezza per il sistema di navigazione del robot.

Capitolo 2
La navigazione tramite landmark

2.1 – La navigazione robotica

La navigazione robotica si occupa di rendere un robot autonomo in grado di muoversi da una posizione a un'altra nell'ambiente che lo circonda.

Il problema generale della navigazione può essere formulato in tre semplici domande [LEO]:

- *Dove sono?* Il robot deve conoscere dove esso si trovi per poter prendere decisioni corrette riguardo il movimento futuro. E' compito della *localizzazione* trovare la risposta a questa domanda.

- *Dove devo andare?* Per potersi muovere con successo e rispettare eventuali vincoli il robot ha bisogno di conoscere la posizione da raggiungere (*goal*). E' compito della *Goal Recognition* trovare la risposta a questa domanda.

- *In che modo ci vado?* Una volta che il robot conosce la propria posizione e sa qual è il proprio *goal*, deve decidere anche in che modo raggiungerlo. E' compito del *Path Planning* trovare la risposta a questa domanda.

La navigazione non è un compito semplice per un robot. Sono diverse le difficoltà da affrontare in un problema di navigazione :

- *Complessità computazionale*. Sono numerosi i processi che gravano sulle prestazioni delle CPU: la capacità di calcolo richiesta dalle operazioni di analisi di immagini in *real time*, la visione artificiale, l'apprendimento del robot. Sebbene i processori moderni effettuino miliardi di operazioni al secondo, essi hanno comunque dei ritardi nello svolgere tutte le operazioni richieste dalla navigazione. Mantenere limitati i ritardi è requisito fondamentale per la navigazione in *real time*.

- *Riconoscimento di oggetti e landmark*. I robot necessitano di riconoscere nell'ambiente che li circonda oggetti e *landmark* per poter compiere le proprie azioni con successo. Il riconoscimento di landmark è complesso e grava pesantemente sulle capacità di calcolo delle CPU. In ambienti non noti a priori dal robot il riconoscimento porta inevitabilmente con se errori e incertezza.

- *Ostacoli e occlusioni*: la gestione del movimento in ambienti con ostacoli è un problema di pianificazione particolarmente complesso, soprattutto nel caso in cui gli ostacoli siano dinamici. Pianificare il percorso può essere reso ulteriormente problematico da eventuali occlusioni di oggetti o landmark necessari alla localizzazione.

- *Fusione sensoriale*. Il robot riceve diverse informazioni dai propri sensori. Queste informazioni devono successivamente essere combinate e confrontate. Sensori di tipo diverso possono dare informazioni discordanti su uno stesso ambiente, causando inconsistenza ed errori nel confronto delle misure.

Affrontare efficacemente la serie di problemi presentati fa parte dei compiti significativi della navigazione.

2.2 - Localizzazione

La localizzazione è il processo che si occupa di determinare e tracciare la posizione del robot nel proprio ambiente. Tutti i robot mobili hanno infatti bisogno di conoscere dove si trovino per poter operare in completa autonomia. La risposta alla domanda "*Dove sono?*" è quindi alla base della localizzazione robotica. [THR]

Formalmente il problema della localizzazione consiste nel determinare la posizione del robot nel sistema di riferimento assoluto, sulla base delle informazioni sensoriali ricevute.

Il problema della localizzazione è importante, poiché è una componente chiave del corretto funzionamento dei sistemi di navigazione. Senza conoscere la propria posizione rispetto all'ambiente esterno, un robot non è in grado di decidere cosa fare, e quindi non è in grado di essere autonomo. [COX]

Esistono diverse istanze del problema della localizzazione, caratterizzate da una crescente complessità [THF]:

- *Position Tracking*: in questo caso il robot conosce la propria posizione iniziale, e la localizzazione si occupa di mantenere traccia della posizione durante gli spostamenti del robot nell'ambiente. Le tecniche che risolvono questo problema sono dette *tecniche locali*.

- *Wake-Up Problem*: in questo caso il robot non ha idea della propria posizione iniziale. Compito della localizzazione è aiutare il robot a localizzarsi attraverso l'analisi dell'ambiente circostante. Il problema è più complesso del precedente, e il

robot potrebbe avere indicazioni e stime discordanti sulla propria posizione. Le tecniche che risolvono questo problema sono dette *tecniche globali*.

- *Kidnapped Robot Problem*: è il problema più complesso. Il robot è perfettamente localizzato, ma viene improvvisamente spostato in una nuova posizione senza esserne aggiornato. In realtà il *Wake-Up Problem* è un sottocaso del *Kidnapped Robot Problem*, in cui il robot è avvertito di essere stato spostato e di doversi rilocalizzare.

La localizzazione robotica affronta i tipi di problemi finora descritti attraverso sistemi di localizzazione relativa e sistemi di localizzazione assoluta.

2.2.1 - Sistemi di localizzazione relativa

I sistemi di localizzazione relativa (detti anche di *dead reckoning*) sono i più utilizzati e diffusi. Tali sistemi basano le stime della posizione sulla base di misure relative del moto del robot. Poiché ogni misura si basa sulle precedenti, nei sistemi di localizzazione relativa gli errori crescono con il tempo durante la fase di movimento del robot. Tuttavia la semplicità realizzativa e l'economicità li rendono preferibili a sistemi più complessi e accurati, e spesso si trovano accoppiati a sistemi di localizzazione assoluta.

Le posizioni relative del robot possono essere acquisite generalmente tramite odometria e navigazione inerziale. [BOR]

2.2.1.1 - Odometria

L'odometria (o *Ricostruzione Odometrica*) lavora integrando nel tempo l'informazione acquisita sul movimento. Attraverso l'impiego di *encoder* sulle ruote del robot, si è in grado di ricostruire con l'odometria la distanza percorsa e la direzione di navigazione. In genere ciò avviene misurando il numero di rivoluzioni delle ruote del robot nell'intervallo di tempo di movimento. Il pregio fondamentale dell'odometria è la semplicità, garantendo misure rapide e poco costose in termini computazionali. Tuttavia l'odometria è pesantemente afflitta da errori sistematici: l'attrito, lo slittamento delle ruote sul terreno, l'eventuale imprecisione degli *encoder*. In particolare, piccoli errori sull'orientazione del robot si ripercuotono come ampi errori sulla posizione. [BOF]

2.2.1.2 - Navigazione inerziale

Nella navigazione inerziale vengono utilizzati giroscopi e accelerometri per misurare velocità di rotazione e accelerazione. Le misure fornite hanno il pregio di essere dirette e di non richiedere informazioni esterne sulla dinamica del robot. Tuttavia, come nel caso dell'odometria, la navigazione inerziale è molto sensibile alle variazioni del terreno, e dovendo integrare i dati una o più volte nel tempo è affetta da errori che crescono con il tempo. [BOF]

2.2.2 – Sistemi di localizzazione assoluta

I sistemi di localizzazione assoluta calcolano la posizione assoluta del robot senza basarsi su misure precedenti. La posizione del

robot è infatti calcolata sulla misura corrente, e non viene stimata integrando misure passate come avviene nei sistemi di localizzazione relativa.

Il pregio di questi sistemi è naturalmente l'assenza di errori che crescono in base al tempo durante il movimento del robot.
I sistemi di localizzazione assoluta si dividono in metodi basati su landmark e metodi basati su mappe topologiche.

2.2.2.1 – Metodi basati su landmark

I landmark sono *features* (letteralmente "*caratteristiche*") dell'ambiente che il robot è in grado di rilevare e riconoscere. Attraverso l'analisi dei dati forniti dai sensori a bordo, un robot determina l'esistenza di un landmark ed esegue successivamente un confronto con informazioni a priori sulla posizione del landmark stesso. Attraverso tali informazioni il robot è in grado di localizzarsi con successo. [SIG]

I landmark utilizzati nel campo della robotica verranno trattati nella sezione 2.3 di questo capitolo, cui si rimanda per ulteriori informazioni.

2.2.2.2 – Metodi basati su mappe

I metodi di *Model Matching* e di confronto di mappe calcolano la posizione del robot in un modo diverso dai metodi precedenti. I sensori del robot cercano *features* geometriche distintive dell'ambiente esterno, e le confrontano con informazioni raccolte a priori sulla distribuzione di tali features nell'ambiente. Esistono due approcci diversi per rappresentare e confrontare mappe:

- *Mappe geometriche*: contengono la descrizione dell'ambiente in un sistema di coordinate globale.

- *Mappe topologiche*: l'ambiente è rappresentato tramite un'interconnessione di nodi (i luoghi dell'ambiente) e archi (le azioni da intraprendere per spostarsi da un nodo all'altro).

I metodi basati su mappe sono in generale molto precisi, tuttavia richiedono la disponibilità di una grande quantità di informazioni provenienti dai dati sensoriali, e necessitano di una capacità computazionale alquanto elevata. [SIG]

2.3 – I landmark

La navigazione tramite landmark è una delle tecniche messe a disposizione dalla localizzazione per rispondere alla domanda: "*Dove sono?*". [THR]

Praticamente tutti gli algoritmi e i sistemi di navigazione esistenti funzionano cercando ed estraendo insiemi di caratteristiche dall'analisi dei dati misurabili dal robot.

Gli approcci di navigazione tramite landmark si basano sulla verifica dei dati sensoriali del robot alla ricerca della presenza o assenza di landmark, allo scopo di risalire alla posizione del robot stesso. [THR]

In base al modo in cui interagiscono con il robot, i landmark si distinguono tra attivi e passivi:

2.3.1 – Landmark attivi

I landmark attivi, per lo più noti come boe (*beacons*), sono landmark in grado di inviare attivamente informazioni sulla

propria posizione al robot. In genere si tratta di apparecchiature radio. Il robot riceve il dato trasmesso dal landmark, dopodichè è in grado di calcolare la propria posizione. [DAN]

Esistono due metodi principali per calcolare la posizione assoluta del robot attraverso l'utilizzo di landmark attivi: la trilaterazione e la triangolazione.

- *Triangolazione*: utilizza le distanze e gli angoli relativi a tre o più landmark visibili dall'*heading* del robot. Attraverso l'analisi di tali dati il robot è in grado di calcolare la propria posizione e orientazione nel sistema di riferimento assoluto.

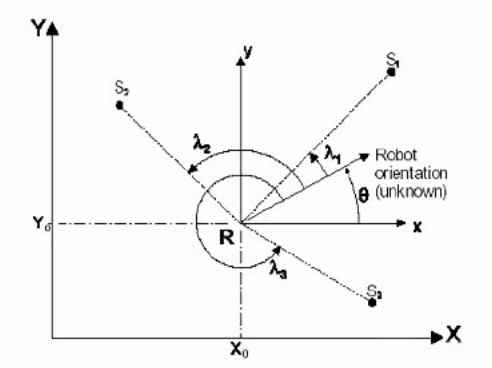


Fig. 1 - Esempio di calcolo della posizione del robot R con tecniche di triangolazione. Il metodo non funziona se i landmark sono allineati oppure se si trovano lungo la circonferenza che li unisce.

- *Trilaterazione*: attraverso l'analisi della sola distanza da almeno tre landmark, il robot è in grado di calcolare la propria posizione. In base alla distanza da un riferimento il robot sa di trovarsi su una circonferenza di raggio pari alla distanza dal landmark stesso. Intersecando almeno tre circonferenze il robot è in grado di localizzarsi con successo. Un tipico esempio di trilaterazione è il sistema GPS.

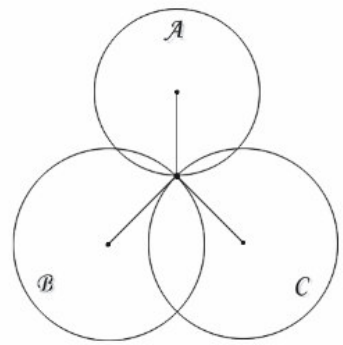


Fig. 2 - Esempio di calcolo della posizione del robot con tecniche di trilaterazione. Per identificare univocamente la posizione sono necessarie almeno tre circonferenze. Con due solo circonferenze si ottengono due possibili soluzioni, delle quali in genere è comunque possibile identificarne una come assurda.

Nei metodi finora descritti il robot necessita di conoscere a priori le posizioni dei landmark. Il successo di questi approcci è strettamente legato alla corretta ricezione dei segnali inviati dalle boe. Essendo apparecchiature di trasmissione, i landmark attivi vanno incontro a tutti i problemi tipici del settore: disturbi, diminuzione di potenza, riflessioni e rifrazioni dei segnali. Le boe inoltre non sono in grado di inviare il segnale in tutte le direzioni, cosicché durante la navigazione del robot alcuni landmark potrebbero risultare nascosti. [DAN]

2.3.2 – Landmark passivi

I landmark che non trasmettono la propria posizione sono detti landmark passivi. Il robot deve cercare autonomamente i landmark per localizzarsi.

I metodi di rilevazione dei landmark passivi variano a seconda del tipo di landmark scelto. E' quindi opportuno distinguere tra landmark artificiali e naturali [SIG]:

- *Landmark artificiali*: sono landmark creati appositamente per la localizzazione. In genere si utilizzano oggetti piani, dalle forme geometriche e dai colori facilmente riconoscibili. I landmark artificiali sono utilizzati con successo generalmente in ambienti molto standardizzati, quali ad esempio uffici e laboratori. Nonostante siano creati ad hoc dalla mano dell'uomo, i landmark artificiali soffrono comunque di notevoli problemi di riconoscimento. All'aumentare dell'angolazione e della distanza dal robot, infatti, i landmark tendono a diventare irriconoscibili, poiché lentamente ne cambia la forma e nascono problemi di illuminazione e visibilità. La capacità computazionale richiesta dal processo di riconoscimento risulta quindi decisamente elevata. [SIG]

- *Landmark naturali*: sono landmark che fanno parte dell'ambiente di lavoro del robot, e non devono essere costruiti appositamente. I landmark naturali soffrono degli stessi problemi dei landmark artificiali, tuttavia sono spesso preferibili a questi ultimi in quanto non è necessario modificare l'ambiente di lavoro del robot.

2.3.3 – Lavori precedenti

Il problema del riconoscimento di landmark è stato affrontato secondo approcci diversi. In generale non esiste un tipo di landmark o un algoritmo di riconoscimento migliore in assoluto, e ogni tecnica tende ad adattarsi particolarmente bene in un

determinato ambiente, e a comportarsi male in altri. Solo i metodi generalisti superano il problema della generalità di utilizzo, recando però con se una complessità e un carico computazionale decisamente elevati.

L'approccio più diffuso prevede l'utilizzo di landmark naturali già presenti nell'ambiente. In generale vengono scelti come landmark elementi tipici di ambienti di ufficio e di laboratorio: porte, finestre, spigoli. Il problema del riconoscimento è legato all'alta possibilità di occlusione degli oggetti. Proprio per superare il problema delle occlusioni si utilizzano spesso elementi dell'ambiente la cui possibilità di risultare nascosti è molto bassa. [LAU] ad esempio basa il proprio metodo di riconoscimento utilizzando come landmark le luci delle lampade di illuminazione dei corridoi, attraverso cui risulta facile costruire mappe topologiche e localizzare il robot.

Un altro approccio molto diffuso si basa sull'individuazione di *cluster* di colore (letteralmente "*agglomerati*"), tramite cui viene analizzata la distribuzione delle zone di uguale colore nell'ambiente. Le condizioni di luminosità giocano però anche in questo caso un ruolo decisivo nel corretto riconoscimento dei cluster da utilizzare come landmark. Approcci di questo tipo sono in genere di realizzazione relativamente semplice, e non gravano molto sulle prestazioni generali dei sistemi di navigazione. Tuttavia individuare cluster di colore richiede un'alta ripetitività e standardizzazione degli ambienti.

[TAK] e [GRO] sviluppano invece tecniche di riconoscimento basate sui contenuti delle immagini (*Visual Learning*). Attraverso l'analisi di una sequenza di immagini acquisite durante una fase di *learning* (letteralmente "*apprendimento*") viene elaborato un aspetto tipico dell'oggetto da riconoscere. Le informazioni raccolte vengono utilizzate durante la fase di riconoscimento dell'oggetto,

mediante comparazione di immagini. In generale i metodi di *Visual Learning* hanno lo svantaggio di richiedere una fase molto lunga di apprendimento (per l'acquisizione e la catalogazione delle immagini). [LAM]

[MAT] realizza invece una tecnica più elaborata attraverso l'uso di tecniche OCR per il riconoscimento di targhe e cartelli tipici degli ambienti di lavoro. Approcci del genere, nonostante l'eventuale accuratezza, soffrono tuttavia di scarsa generalità di utilizzo, e problemi di riconoscimento dovuti all'angolazione con cui vengono visti i landmark.

[THR] invece teorizza e approfondisce l'approccio statistico basato su reti neurali. In questo caso i landmark non sono predefiniti, ma vengono automaticamente scelti dal robot attraverso l'utilizzo di filtri adattivi e reti neurali. Il robot in pratica sceglie quali sono i landmark più significativi dell'ambiente in base al proprio punto di vista. Tale scelta rende molto robusto e rigoroso il riconoscimento dei landmark. La controparte dei metodi basati su reti neurali è però la necessaria e lunga fase di apprendimento da parte del robot, la notevole complessità di implementazione, e l'alto carico computazionale. [THR]

Capitolo 3
Estrazione di features da immagini

3.1 – Tecniche di elaborazione di immagini

La *Computer Vision* si occupa dello sviluppo di sistemi in grado di interpretare il contenuto di scene naturali. Rispetto alle capacità di calcolo dei moderni calcolatori, i sistemi di visione attuali sono da un lato straordinariamente precisi nell'estrazione di informazioni qualitative dalle immagini, dall'altro scarsamente accurati per quanto riguarda le analisi quantitative.[LUC]

L'estrazione di *features* svolge un ruolo fondamentale nel portarci alla comprensione quantitativa delle immagini. Una *feature* è definita come una funzione di una o più misure, intese come valori di una proprietà quantificabile dell'immagine, calcolata in modo da quantificare alcune delle caratteristiche significative di un oggetto. [CAS]

Le immagini digitali acquisite dai sistemi di visione quasi sempre non sono perfette: la prospettiva, l'illuminazione, le deformazioni, e più in generale il rumore rendono necessario un'elaborazione preliminare dei dati da analizzare. [JIA]

3.1.1 – Pre-processing

Il *pre-processing* è l'elaborazione iniziale cui viene sottoposta un'immagine allo scopo di semplificare la successiva estrazione di features. Il *pre-processing* è particolarmente indicato per

migliorare il contrasto, ridurre il rumore, e separare gli oggetti dallo sfondo.

Il *pre-processing* si realizza generalmente con l'analisi di istogrammi oppure con tecniche di filtraggio:

- *Istogramma*: indica per un immagine a toni di grigio il numero di pixel per ciascun livello di grigio. Attraverso l'analisi dell'istogramma si ottengono importanti informazioni sulla luminosità e sul contrasto. Operando successivamente sull'immagine con un'opportuna operazione di *mapping* (letteralmente "*mappatura*") si rende possibile schiarire, scurire, evidenziare o nascondere oggetti, equalizzare l'istogramma. Il riconoscimento di features catturate in diverse condizioni d'illuminazione ha alla base tecniche di mapping di equalizzazione.

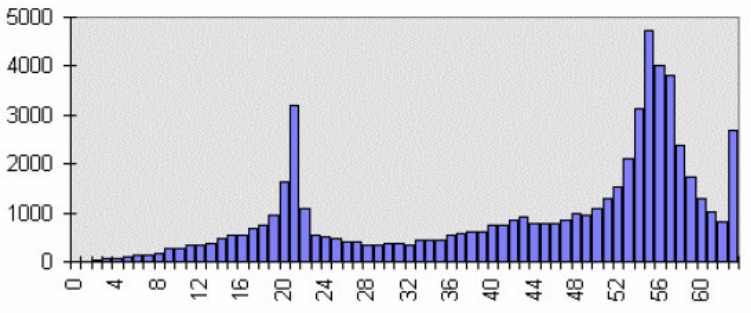


Fig. 3 - Esempio di istogramma ricavato dall'analisi di un'immagine generica. L'asse delle ordinate indica la quantità di pixel, lasse delle ascisse indica il livello di grigio. Si possono facilmente individuare e aree di massimo e minimo.

- *Filtri digitali*: sono maschere di pesi che indicano come modificare ogni pixel dell'immagine sulla base dei pixel vicini.

L'intensità di ogni pixel viene modificata in base alla convoluzione con il filtro scelto:

$$\mathbf{I}[i,j]=\sum_{i_c=1}^{M_1}\sum_{j_c=1}^{M_2}\mathbf{I}\left[i-i_c+\left(\frac{M_1}{2}+1\right),j-j_c+\left(\frac{M_2}{2}+1\right)\right]\cdot\mathbf{F}[i_c,j_c]$$

dove $[i,j]$ è il punto dell'immagine I in cui è applicato il filtro F definito sulla griglia $M_1 \times M_2$.

La complessità computazionale della convoluzione è elevata: data un'immagine di $N \times N$ pixel e un filtro di $M \times M$ pixel, richiede $N^2 \times M^2$ somme e altrettanti prodotti. In generale si cerca di calcolare più efficientemente la convoluzione tramite tecniche di linearizzazione (precalcolando gli offset dei soli pixel dell'immagine diversi da zero), o, se possibile, con tecniche di separazione dei filtri (applicando due filtri monodimensionali in serie anziché un filtro bidimensionale).

$$\mathbf{I}[i,j]=\sum_{i_c=1}^{M_1}\left\{\sum_{j_c=1}^{M_2}\mathbf{I}\left[i-i_c+\left(\frac{M_1}{2}+1\right),j-j_c+\left(\frac{M_2}{2}+1\right)\right]\cdot\mathbf{F}_2[j_c]\right\}\cdot\mathbf{F}_1[i_c]$$

Formula generale di separazione di un filtro digitale.

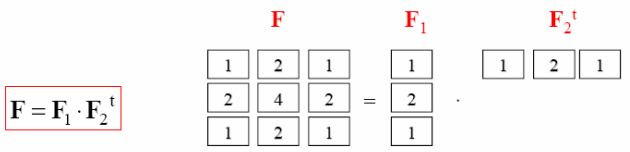


Fig. 4 - Esempio di separazione di un generico filtro F. Il risultato del filtro F applicato ad un'immagine è lo stesso di quello che da la composizione dei filtri F_1 e F_2.

Principalmente vengono utilizzati filtri di *sharpening* e di *smoothing*.

I filtri di *smoothing* producono una sfocatura variabile, in grado di nascondere piccole imperfezioni e brusche variazioni di luminosità dell'immagine.

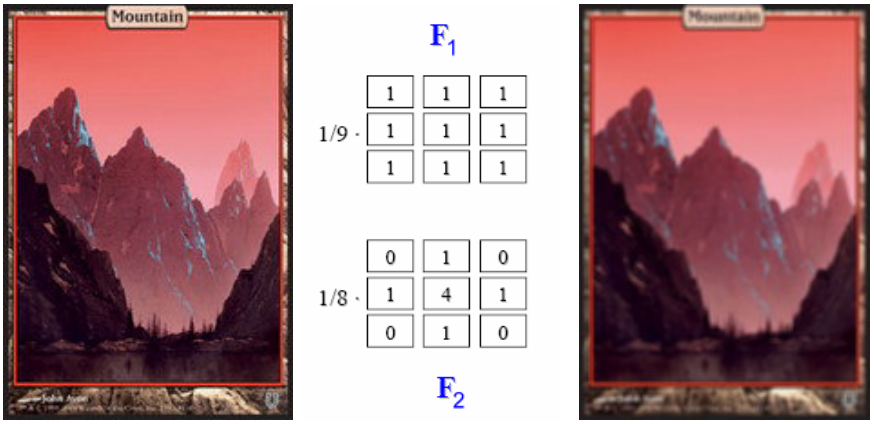


Fig. 5 - Esempi di filtri di smoothing. L'immagine di destra rappresenta l'output del filtro F_1 applicato all'immagine di sinistra.

I filtri di *sharpening* evidenziano i dettagli fini dell'immagine e le brusche variazioni di luminosità (spesso concentrate nei contorni), sommando la risposta del filtro all'immagine originale.

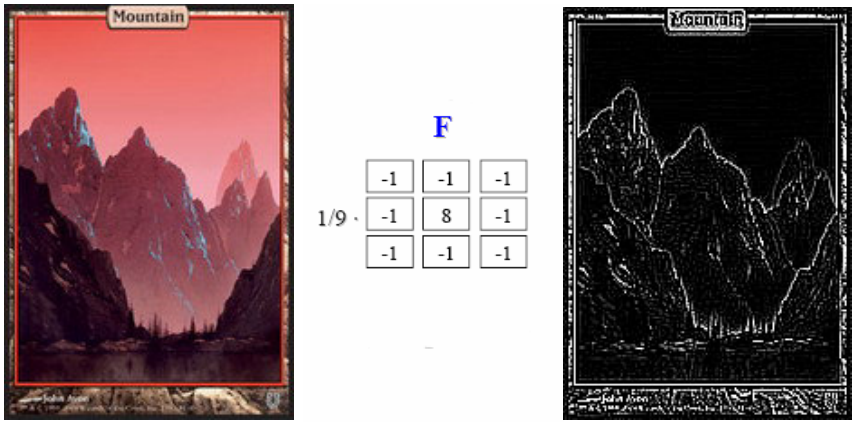


Fig. 6 - Esempio di filtro di sharpening. Attraverso l'applicazione del filtro F si produce l'output mostrato nella figura di destra.

3.1.2 – Estrazione dei contorni

I contorni degli oggetti di un'immagine sono invarianti alle condizioni di illuminazione e rumore, e sono indispensabili per la modellizzazione geometrica della forma. [CAN]

Le principali tecniche di estrazione di contorni si basano sull'analisi del gradiente dell'immagine. Il gradiente è il vettore contenente le derivate parziali dell'immagine lungo le direzioni principali:

$$\nabla \mathbf{I}[x,y] = \left[\frac{\partial \mathbf{I}[x,y]}{\partial x}, \frac{\partial \mathbf{I}[x,y]}{\partial y} \right]$$

La derivata di un segnale identifica la variabilità di un segnale, nel nostro caso bidimensionale. In corrispondenza di forti variazioni locali, la derivata assume valore elevato. Laddove invece il segnale è costante, la derivata è nulla.

Il calcolo delle derivate parziali viene effettuato tramite la definizione di rapporto incrementale:

$$\frac{\partial \mathbf{I}[x,y]}{\partial x} = \lim_{\Delta x \to 0} \frac{\mathbf{I}[x,y] - \mathbf{I}[x+\Delta x, y]}{\Delta x} \Rightarrow \mathbf{I}[x,y] - \mathbf{I}[x+1,y]$$

$$\frac{\partial \mathbf{I}[x,y]}{\partial y} = \lim_{\Delta y \to 0} \frac{\mathbf{I}[x,y] - \mathbf{I}[x,y+\Delta y]}{\Delta y} \Rightarrow \mathbf{I}[x,y] - \mathbf{I}[x,y+1]$$

Tuttavia, a causa di problemi di simmetria rispetto al punto di applicazione, e di robustezza al rumore, il calcolo delle derivate parziali viene generalmente effettuato mediante una coppia di filtri digitali: [MEI]

- *Filtro di Prewitt*: è un operatore direzionale, combina la rilevazione dei contorni con il calcolo di una media locale direzionale.

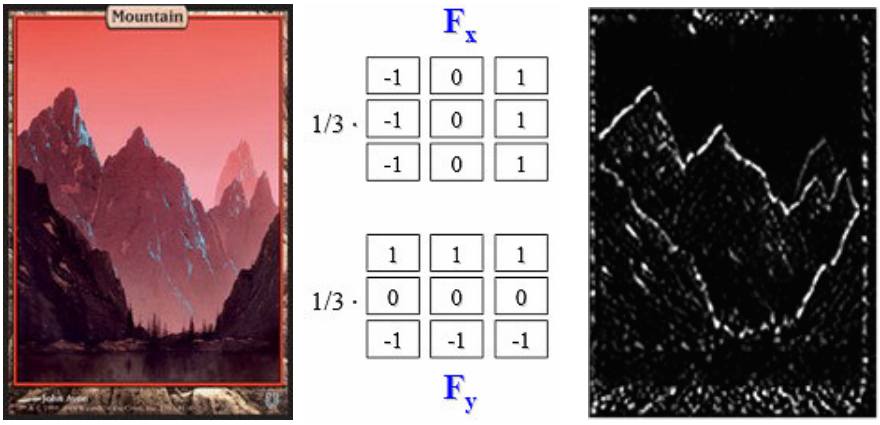


Fig. 7 - Esempio di applicazione del filtro di Prewitt.

- *Filtro di Sobel*: è un operatore direzionale simile al filtro di Prewitt, in cui però la media locale è pesata in funzione della distanza dei pixel coinvolti nella computazione dal pixel il cui valore è oggetto di calcolo.

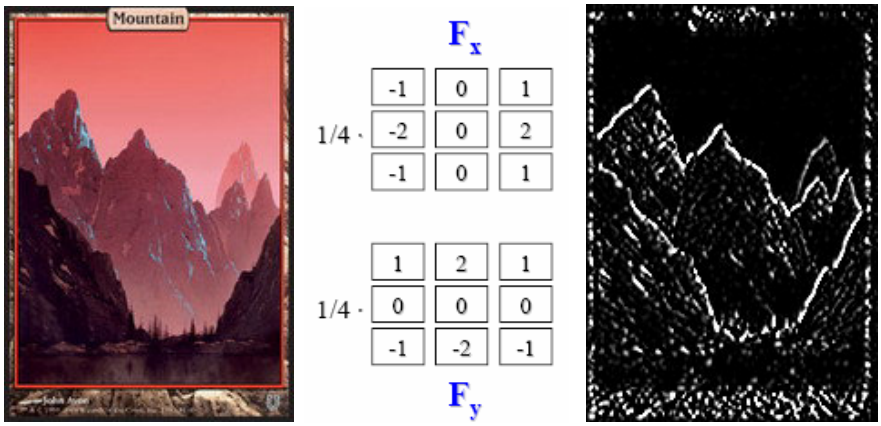


Fig. 8 - Esempio di applicazione del filtro di Sobel.

I filtri di Prewitt e Sobel producono immagini in cui i contorni sono ben visibili ma non direttamente utilizzabili, a causa di

frammentazioni degli edge, e di segmenti spuri generati a causa del rumore. Le tecniche di estrazione dei contorni sono alla base di quasi tutte le tecniche di riconoscimento di features, e sono descritti nel paragrafo 3.2.3.1, dedicato all'*Edge Detection*.

3.1.3 – Segmentazione

Obiettivo della segmentazione è separare gli oggetti di interesse dallo sfondo dell'immagine. [JOH] Per segmentare un'immagine si ricorre in generale ad approcci di binarizzazione tramite soglia (*thresholding*) o di *region growing*:

- *Binarizzazione tramite soglia globale*: è una tecnica di soglia che si utilizza nel caso vi sia un unico oggetto di interesse, caratterizzato da sfondo uniforme. La tecnica è semplice, ma non è in grado di gestire sfondi di intensità variabile, e non è adatta a segmentare oggetti non uniformi. [HRS]

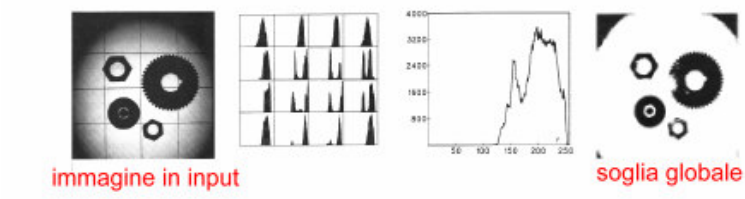


Fig. 9 - Output di un'immagine binarizzata tramite soglia globale. Attraverso l'analisi degli istogrammi di ogni regione in cui è suddivisa l'immagine si giunge all'output di destra, in cui sono isolati gli oggetti presenti nell'immagine. Non essendo lo sfondo perfettamente uniforme sono rimasti alcuni elementi a bordo immagine.

- *Binarizzazione tramite soglia locale*: questa tecnica si applica quando lo sfondo è caratterizzato da variazioni graduali di luminosità da una parte all'altra dell'immagine. L'immagine

viene suddivisa in regioni, all'interno di ciascuna delle quali viene calcolato l'istogramma, e si determina la soglia locale per la segmentazione. [HRS]

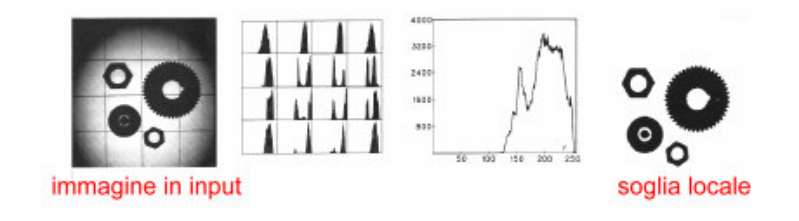


Fig. 10 - Output di un'immagine binarizzata tramite soglia locale. Attraverso l'analisi degli istogrammi di ogni regione in cui è suddivisa l'immagine si giunge all'output di destra, in cui sono isolati gli oggetti presenti nell'immagine.

- *Region growning*: questa tecnica viene utilizzata nel caso in cui vi siano due o più oggetti da segmentare, oppure quando gli oggetti e lo sfondo non sono uniformi. Le tecniche di *region growning* identificano dei "semi" (costituiti da uno o più pixel) per ogni regione dell'immagine, e accrescono tali regioni aggiungendo pixel sulla base di un criterio di assegnazione predefinito, nel rispetto di vincoli di connessione. Durante tali operazioni le diverse regioni vengono fuse (*merging*) o separate (*splitting*), isolando gli oggetti. [HRS]

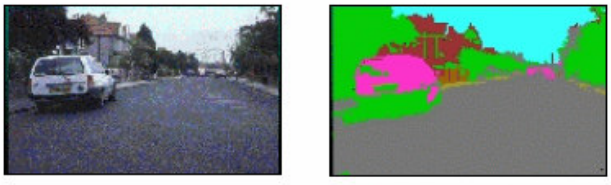


Fig. 11 - Esempio di region growning: l'immagine a sinistra produce l'output di destra, in cui sono evidenziate da diversi colori le regioni individuate.

3.2 - Tecniche di estrazione di features

Features appartenenti a una medesima tipologia presentano in generale metodi molto simili di estrazione. Nonostante non vi sia una classificazione rigorosa dei vari tipi di features, si è comunque soliti distinguere tra *General Features* e *Domain Feature*. [LEI]

3.2.1 - General Features

Sono features indipendenti dal contesto dell'immagine. In questa categoria si possono individuare diverse tecniche di estrazione, tra cui si impongono tre approcci principali:

- *Pixel Level Features* : sono features calcolate attraverso l'analisi di ogni singolo pixel dell'immagine. Esempio classico è l'individuazione di features tramite la ricerca di cluster di colore nelle immagini acquisite.

- *Local Features* : sono features calcolate attraverso la segmentazione dell'immagine tramite algoritmi di individuazione dei contorni. Si prestano bene all'utilizzo in ambienti chiusi caratterizzati da elementi distintivi ripetitivi, quali ad esempio porte e finestre.

- *Global Features* : sono features calcolate sull'intera immagine attraverso l'analisi di distribuzioni matematiche e statistiche. Le tecniche di estrazione di *Global Features* sono molto accurate e precise, ma al tempo stesso complesse da implementare e lente nell'esecuzione.

Le *General Features* sono dette anche di Basso Livello, poichè sono estraibili direttamente dall'immagine. [LEI]

3.2.2 - Domain Features

Sono features relative al contesto dell'immagine e dell'applicazione considerata. Proprio per la loro particolarità, esistono moltissime tecniche di riconoscimento sperimentali. I principali campi di ricerca sono generalmente legati al riconoscimento di facce e all'analisi di impronte digitali.

Le *Domain Features* sono dette anche di Alto Livello, poichè per il loro riconoscimento è necessaria l'analisi di features di Basso Livello. [LEI]

3.2.3 – Estrazione di features di Basso Livello

Le tecniche di estrazione di features di basso livello sono delle *primitive* tra le tecniche di estrazione di features stesse, in quanto ad esse ricorrono anche quelle di estrazione di features alto livello. [CAS]

3.2.3.1 – Edge Detection

E' la tecnica di estrazione di features più importante e utilizzata, soprattutto per il riconoscimento di features di alto livello.

Un *edge* (spigolo) è una zona di immagine in cui vi è una discontinuità del gradiente in una sola direzione, ovvero è un margine tra due zone differenti dell'immagine stessa. [CAN]

Esistono tre principali approcci per la rilevazione degli *edge*:

- *First Order Derivative*: analizza la derivata del primo ordine dell'immagine. L'idea alla base di questa tecnica è di identificare gli *edge* nei massimi locali della derivata. Attraverso la convoluzione dell'immagine con una maschera di *Edge Operator* si individuano i massimi del gradiente:

Simple Operator

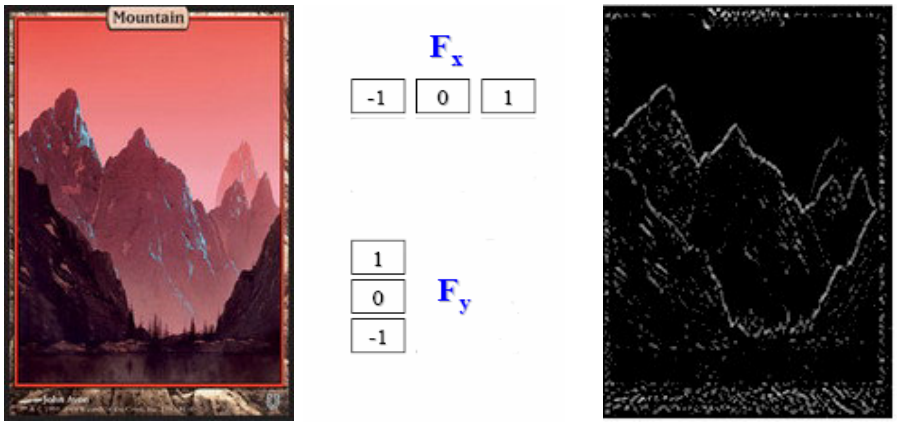


Fig. 12 - Esempio di applicazione del Simple Operator.

Robert's Cross Operator

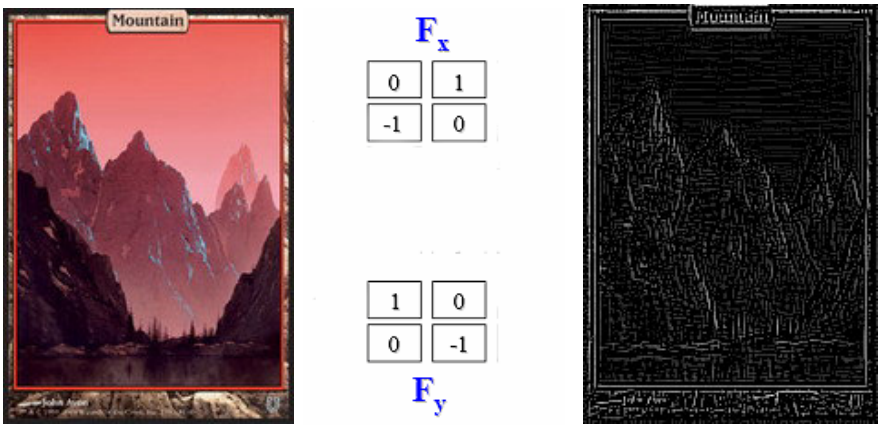
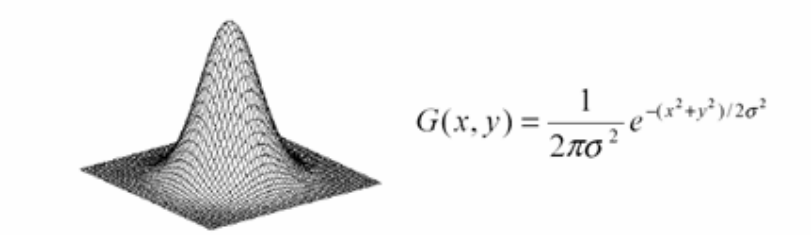


Fig. 13 - Esempio di applicazione del Robert's Cross Operator.

Fanno parte di questa categoria anche gli operatori di Sobel e Prewitt, trattati nel cap. 3.2.1. I diversi filtri differiscono in accuratezza e complessità computazionale, e vanno scelti in

base alle caratteristiche dell'immagine da analizzare. Attraverso un'operazione di soglia successiva al filtraggio viene poi realizzata la mappa degli *edge* dell'immagine. [MAR]

- *Second Order Derivative*: analizza la derivata seconda dell'immagine. Si basa sul presupposto che i massimi della derivata prima corrispondano agli "*zero-crossing*" della derivata seconda. Si analizza l'immagine tramite il *Laplacian of Gaussian (LoG)*, in cui viene calcolato l'*edge* attraverso l'analisi del Laplaciano dell'immagine pre-trattata con un filtro Gaussiano [MAR]:



$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

Fig. 14 - Definizione di filtro gaussiano. Si tratta di un filtro di smoothing in cui gli elementi dell'immagine sono pesati secondo pesi gaussiani. Nella pratica il filtro esegue una media locale, pesata in funzione della distanza dal centro, dei valori dei pixel di un intorno simmetrico di dimensioni variabili in base al parametro σ, deviazione standard di G(x,y).

Fig. 15 - Esempio di applicazione del filtro gaussiano.

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Definizione di Laplaciano.

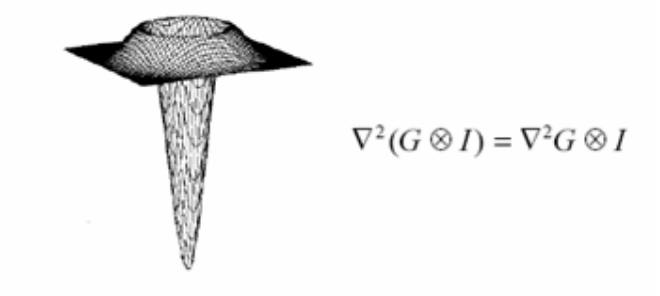


Fig. 16 - Definizione di Laplacian of Gaussian. La componente G effettua lo smoothing, la componente differenziale ∇^2 realizza la rilevazione degli edge.

Fig. 17 - Esempio di applicazione del Laplacian of Gaussian per l'individuazione di edge in un'immagine generica.

- *Canny Edge Detector*: analizza l'intensità del gradiente di un'immagine pre-trattata con un filtro di *smoothing* (generalmente un filtro gaussiano). La gestione dei punti di massimo assoluto e relativo è trattata con l'analisi di un

neighbourhood (intorno) di ogni pixel dell'immagine. La dimensione del *neighbourhood* è variabile arbitrariamente a seconda dell'accuratezza desiderata. Un'operazione di soglia finale estrae gli *edge* [CAN].

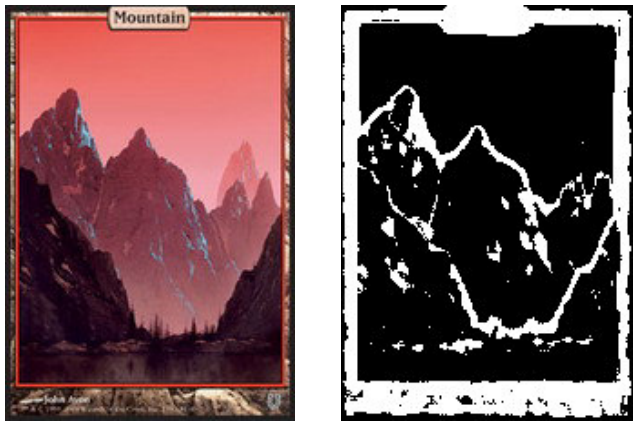


Fig. 18 - Esempio di applicazione del Canny Edge Detector, con neighbourhood 9 × 9 e l'impiego di un filtro gaussiano a $\sigma = 2$. Il risultato è abbastanza grezzo, ma settando opportunamente le soglie, il neighbourhood e i parametri del filtro è possibile ottenere risultati migliori.

3.2.3.2 – Textures

Sono features caratterizzate da zone di omogeneità derivante non solo dal colore. Sono una proprietà innata di praticamente tutte le superfici. I principali approcci al riconoscimento di *textures* si basano sull'analisi diretta delle zone di omogeneità delle immagini (*analisi di basso livello*), o sul *matching* (letteralmente "*confronto*") tra immagine acquisita e pattern di riferimento (*analisi di alto livello*). [HRA]

3.2.3.3 – Corner

I *corners* (angoli) sono punti estremamente caratteristici delle immagini bidimensionali. I *corners* sono facilmente rappresentabili matematicamente, poiché non corrispondono necessariamente ad alcun oggetto geometrico della scena osservata. [HRA]

In genere si parla di "*T*", "*Y*" e "*X junctions*", a seconda della forma assunta dal corner.

Formalizzando, un corner è una zona dell'immagine in cui si verifica una discontinuità del gradiente in due diverse direzioni. [HAR]

Il riconoscimento di corner avviene in generale tramite due diversi approcci:

- *Edge Based*: si basa sull'estrazione degli *edge* dell'immagine. Attraverso un *matching* tra le informazioni acquisite vengono individuati i corner. Questo genere di metodi risente tuttavia della qualità della segmentazione dell'immagine acquisita

- *Raw Based*: si basa sull'analisi diretta delle variazioni di intensità in immagini in scala di grigio. I corner sono indipendenti alle variazioni di luminosità, si può quindi giungere all'identificazione attraverso una semplice analisi dell'intensità del gradiente dell'immagine acquisita. I metodi *raw based* tendono spesso però a rilevare falsi corner, e necessitano di un'ulteriore analisi dei *neighbourhood* (letteralmente "*intorni*") di ogni pixel dell'immagine per garantire l'accuratezza dei risultati.

3.2.4 – Estrazione di features di Alto Livello

Le tecniche di riconoscimento di features di alto livello vengono utilizzate principalmente per analisi di oggetti e forme non semplici (spesso artificiali) presenti nelle immagini. Attraverso l'utilizzo combinato delle tecniche di estrazione di features di basso livello si possono ottenere sistemi di riconoscimento per qualsiasi tipo di oggetto. L'ampio campo di azione di queste tecniche è tuttavia gravato dall'altissima complessità computazionale, che in generale esula tali tecniche dalla possibilità di utilizzo in sistemi *real time*. [LEI]

3.2.4.1 - Point Patterns

Sono tecniche di estrazione basate su cluster di punti. Ogni immagine computerizzata è rappresentabile per punti tramite pixel. Le tecniche di *Point Patterns* si basano sul raggruppamento di punti simili in base a vincoli predefiniti. L'esempio più diffuso di *Point Patterns* è il riconoscimento di cluster di colore.

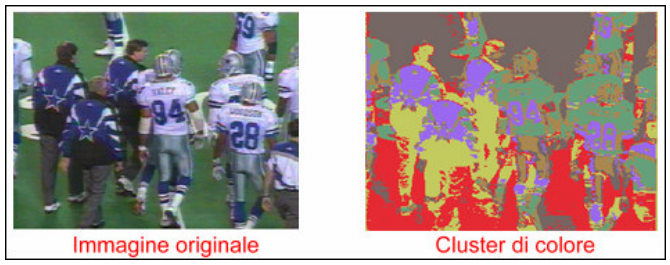


Fig. 19 - Esempio di individuazione di cluster di colore.

Un altro metodo molto diffuso tra le tecniche di *Point Patterns* é l'utilizzo della *Trasformata di Hough*.

La Trasformata di Hough permette il riconoscimento di configurazioni globali presenti in una immagine (segmenti, curve,

forme prestabilite), attraverso la trasformazione dei punti costituenti l'immagine in punti di un nuovo spazio detto spazio dei parametri. La tecnica si applica generalmente ad immagini binarie, e si basa sulla validazione di ipotesi tramite cui, definita la curva che si intende cercare nell'immagine, si calcolano i parametri di tutte le curve che potrebbero passare per ogni punto, incrementando le celle di uno spazio *n*-dimensionale (con *n* numero dei parametri) che corrispondono alle varie curve. Si ottiene in questo modo una funzione di accumulazione definita nello spazio dei parametri, i cui massimi rappresentano le curve che hanno probabilità elevata di essere presenti nell'immagine. La Trasformata di Hough non risente generalmente del rumore, in quanto eventuali lacune presenti nella retta di partenza o la presenza di punti spuri non influenzano significativamente la funzione di accumulazione. [LEI]

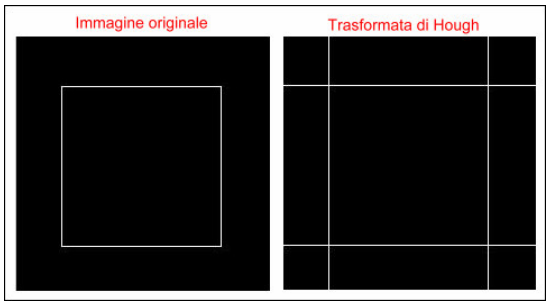


Fig. 20 - Esempio di ricerca di rette con la trasformata di Hough.

3.2.4.2 - Shape Description

Sono tecniche che si basano sull'estrazione di *edge* dalle immagini. Vengono utilizzate funzioni matematiche e statistiche per riconoscere forme predefinite nella disposizione degli *edge* nell'immagine. [LEI]

3.2.4.3 - Contour Segmentation

E' sicuramente uno degli approcci più complessi a livello computazionale, basato sul *linking* dei pixel di *edge* allo scopo di ricostruire forme di oggetti, anche parzialmente occlusi. [LEI]

3.3 – Corner detection

I corner sono features privilegiate nell'analisi di immagini digitali e nella localizzazione robotica. L'importanza del corner sta essenzialmente nella propria invarianza rispetto ai cambiamenti di illuminazione. Questa caratteristica fa sì che il moto del robot non risulti ambiguo nelle vicinanze di un corner. [HAR] Rispetto ad un *edge*, il moto è sicuramente ambiguo senza l'analisi di ulteriori riferimenti. Utilizzare il corner come feature consente quindi un risparmio computazionale durante la fase di riconoscimento.

Un'altra qualità apprezzabile del corner è la possibilità di estrazione da immagini in toni di grigio. [HAR] La quantità di dati da processare è un fattore di grande importanza in campi quali la navigazione e localizzazione robotica, i cui requisiti di *real time processing* e la gran mole di operazioni gravano sulla CPU e di conseguenza sulle prestazioni generali del sistema. Poter lavorare su immagini in toni di grigio consente di analizzare un terzo dei dati contenuti in un'immagine acquisita da una telecamera (vedere Appendice C).

I corner, inoltre, essendo una feature di basso livello possono essere estratti direttamente dall'immagine senza la necessità di operazioni intermedie o di individuare altre features.

I due principali algoritmi di estrazione di corner dalle immagini digitali, entrambi di tipo *raw based*, sono stati realizzati da Harris e da Kanade, Lucas e Tomasi. [CSE]

3.3.1 - Estrattore di Harris

L'estrattore di Harris, noto come *Harris Corner Detector* [HAR], si basa sulla definizione di *Local Structure Matrix*:

$$C_{str} = \omega_G(r;\sigma) * \begin{bmatrix} f_x^2 & f_x f_y \\ f_x f_y & f_y^2 \end{bmatrix}$$

La matrice C_{str} è formata dai prodotti dalle derivate parziali lungo le due direzioni dell'immagine, calcolati in ogni punto. Tale matrice è poi convoluita con un filtro Gaussiano ω_g. Talvolta però si preferisce sostituire il filtro Gaussiano con altri filtri meno pesanti a livello computazionale.

La matrice C_{str} è per definizione simmetrica e definita positiva. E' quindi possibile diagonalizzare la matrice nella forma:

$$C_{str} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

$$\lambda_1 \geq \lambda_2 \geq 0$$

in cui gli autovalori λ_1 e λ_2 sono non negativi.
Attraverso l'analisi punto per punto di entrambi gli autovalori della matrice C_{str} possiamo fare alcune considerazioni sull'immagine:

- *l'immagine è uniforme*: se gli autovalori sono entrambi nulli, allora l'immagine è perfettamente uniforme in quel punto;

- *il punto appartiene a un edge*: se l'autovalore λ_1 è positivo e l'autovalore λ_2 è nullo, il punto appartiene a un edge, e l'autovettore associato a λ_1 è perpendicolare all'edge;

- *il punto appartiene a un corner*: se entrambi gli autovalori sono positivi non nulli, allora il punto appartiene a un corner;

Si osserva inoltre che poiché un corner è l'incontro di due edge, in tale zona dell'immagine il valore dell'autovalore λ_2 sarà elevato.
Sulla base delle osservazioni precedenti Harris definisce una funzione in grado di misurare l'intensità di ogni corner:

$$H(x,y) = \det C_{str} - \alpha (\text{trace } C_{str})^2$$

$$\det A = \prod_i \lambda_i \ , \ trace \ A = \sum_i \lambda_i$$

da cui si ottiene:

$$H = \lambda_1 \lambda_2 - a \ (\lambda_1 + \lambda_2)^2$$

dove a è un parametro e $H \geq 0$ se $0 \leq a \leq 0.25$.
Un corner è rilevato quando;

$$H(x,y) \geq H_{thr}$$

dove H_{thr} è un parametro di soglia che determina la robustezza del corner. Per risolvere il problema degli angoli deboli nelle vicinanze degli angoli robusti in genere la matrice C_{str} viene calcolata su un *neighbourhood* $D \times D$ di ogni punto, in modo da tener traccia nel corso dell'algoritmo dei corner robusti. Non esiste una regola per scegliere tale intorno, in generale $2 < D < 10$.

Per quanto riguarda i valori delle due variabili di soglia H_{thr} e a, si scelgono valori:

- $H_{thr} \geq 0$, $a > 0.10$ per avere un estrattore molto preciso, che rilevi pochi corner
- $H_{thr} \geq 0$, $a < 0.10$ per rilevare molti corner.

L'impostazione delle due variabili va fatta per via euristica, sulla base di risultati sperimentali. Attraverso un'eventuale operazione di soglia finale si scelgono infine i corner più robusti (generalmente attraverso l'analisi di un istogramma).

3.3.2 - Estrattore KLT

Kanade, Lucas e Tomasi (KLT) propongono un algoritmo di estrazione che si fonda sull'idea originale di Harris. [CSE]

Data la matrice C_{str}, si costruiscono per ogni punto i *neighbourhood* D × D su cui verificare l'intensità dell'autovalore λ_2 della matrice, identificando con λ_2 il minore tra i due autovalori:

$$\lambda_1 \geq \lambda_2 \geq 0$$

Se $\lambda_2 > \lambda_{thr}$, dove λ_{thr} è un valore di soglia, allora ci troviamo in presenza di un corner.

Come per Harris, anche in questo caso il valore di λ_{thr} va stimato sulla base di analisi sperimentali, ed è possibile scegliere i corner più robusti attraverso un'ulteriore operazione di soglia.

3.3.3 - Confronto tra i due metodi

Entrambi i metodi sono accomunati dall'analisi della matrice C_{str} in cerca delle zone dell'immagine in cui le variazioni in entrambe le direzioni sono alte.

La differenza principale risulta nella scelta delle soglie. In Harris la soglia è implicita attraverso l'uso della funzione H. In KLT la soglia è determinata esplicitamente dall'analisi di C_{str}. Ciò rende l'uso dell'estrattore KLT più vicino alla percezione umana degli angoli, mentre Harris consente una ripetitività più rigida dei corner attraverso ambienti con forti variazioni di angolazione e illuminazione.

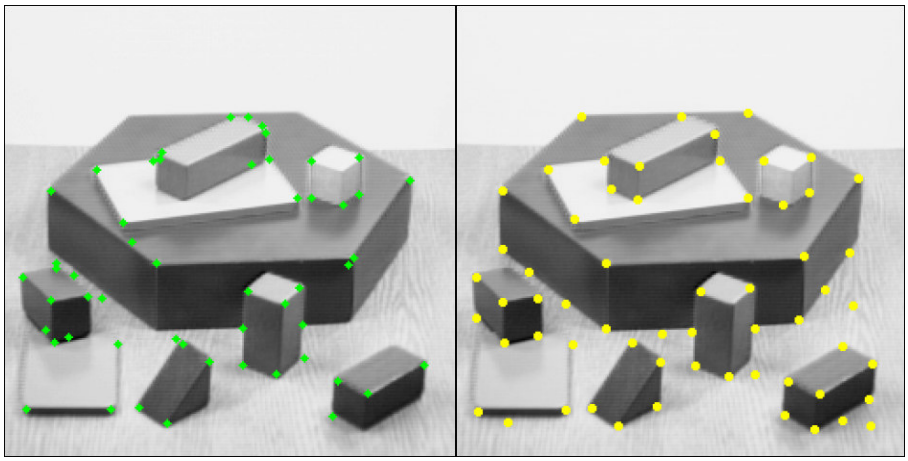


Fig. 21 - Confronto tra i corner rilevati dai due algoritmi. Nell'immagine a sinistra sono evidenziati i corner riconosciuti da Harris, a destra quelli riconosciuti da KLT.

3.3.4 – Lavori Precedenti

Uno degli approcci più diffusi nella ricerca di corner con tecniche *edge based* si fonda sull'analisi della curvatura degli edge. Data un'immagine, vengono calcolati in essa gli edge, dopodichè

attraverso l'analisi delle curvature é possibile risalire alla disposizione dei corner. [FRE]

Simile é l'approccio basato sulla *similaritá*. Vengono create delle funzioni locali di autocorrelazione che analizzano gli edge in entrambe le direzioni, ed in corrispondenza di un corner la funzione riporta un massimo. [COO]

Metodi come quelli di [FRE] e [COO] sono abbastanza efficaci per corner generici, ma incontrano grandi difficoltà nel riconoscimento di *junctions*. [ASA]

Un approccio sostanzialmente diverso é quello basato su *patch*. L'immagine (trattata in toni di grigio) é suddivisa in zone dette *patch*, e ciascuna *patch* é a sua volta divisa in due "regioni" tramite segmentazione. L'idea di fondo é che solo due regioni possano formare un corner. Attraverso l'analisi dell'intensità di grigio nelle due regioni di una *patch* si verifica la presenza o assenza di corner. [LIU] Nonostante gli ottimi risultati su immagini semplici, anche l'approccio su *patch* non é in grado di rilevare *junctions*, e soffre significativamente il rumore.

In generale approcci come quelli di [FRE], [COO] e [LIU] sono tutti strettamente legati alla possibilità di segmentare correttamente l'immagine, e ciò li rende inadatti all'analisi di immagini acquisite da strumenti in cui il rumore é inevitabile. Questo è uno dei motivi per i quali sono le tecniche *raw-based* ad essere preferite nelle applicazioni di visione artificiale.

Un approccio diffuso tra le tecniche *raw-based* è quello del *template matching*, in cui viene calcolata la similarità di ogni sotto-parte dell'immagine rispetto a un modello predefinito del corner da individuare. [RAN] sviluppa un corner detector basandosi sulla descrizione matematica del corner, individuando corner simmetrici rispetto all'asse orizzontale dell'immagine: il detector è il prodotto del seno nella direzione orizzontale X e

l'esponenziale nella direzione verticale Y in una parte della maschera, e il prodotto dei due seni in entrambe le direzioni nella parte restante di maschera. L'algoritmo è generalizzato per rilevare corner di qualsiasi ampiezza e orientazione. L'algoritmo è estremamente complesso e di gran peso computazionale, tuttavia grazie alla separabilità delle maschere di filtraggio è possibile migliorarne le prestazioni.

[MOR] introduce invece l'idea dei *punti di interesse*: si individuano le zone dell'immagine in cui vi sono alte variazioni d'intensità nella scala di grigio verso tutte le direzioni. Attraverso tecniche di autocorrelazione e soglia vengono evidenziati i punti di massimo. L'algoritmo è molto efficace, tuttavia dipende in modo decisamente significativo dal rumore.

Un ulteriore approccio molto usato è basato sull'utilizzo del solo gradiente dell'immagine. L'algoritmo più noto e diffuso è il SUSAN (*Smallest Univalve Segment Assimilatine Nucleus*) [SMI]: viene generata una maschera di filtraggio circolare su ogni punto dell'immagine, il quale viene confrontato in intensità con i punti vicini, se la similarità è bassa l'algoritmo identificherà il punto come un minimo, in cui sarà probabile trovare un corner. L'algoritmo si comporta bene anche in condizioni di rumore elevato.

Sono stati infine realizzati algoritmi e tecniche ibride, in cui vengono impiegate diverse tecniche assieme. [SIN] Tuttavia praticamente tutte queste tecniche risultano molto sensibili al rumore, e sono caratterizzate da un carico computazionale troppo elevato per un utilizzo efficace in applicazioni *real time*.

Capitolo 4
Localizzazione e filtro di Kalman

4.1 – Il filtro di Kalman

Per *stato* di un sistema generalmente si intende un vettore x formato da n variabili rappresentative di proprietà significative del sistema. Lo stato rappresenta la posizione del robot, costituita dalle coordinate x ed y, e dall'orientazione θ.

Stimare lo stato del robot può essere difficoltoso, perché le variabili di stato possono essere affette da rumore, e non direttamente osservabili. Le misure raccolte dal robot necessitano perciò di essere analizzate e combinate tra loro per poter fornire una stima plausibile della posizione del robot stesso.

Il Filtro di Kalman è un algoritmo ricorsivo impiegato per analizzare i dati provenienti da un sistema dinamico. Il Filtro di Kalman è il miglior stimatore possibile dello stato per un'ampia classe di sistemi caratterizzati da incertezza [KAL]. Ideato nel 1960 da Richard Kalman, il filtro ha riscosso grande successo ed è stato ampiamente utilizzato per via delle sue proprietà fondamentali:

- *il filtro è ricorsivo*: non è necessario memorizzare ed elaborare tutti i dati ad ogni passo di esecuzione del filtro. Le informazioni raccolte nei passi precedenti sono infatti incorporate nell'ultimo risultato del filtro;

- *il filtro è ottimo*: si dimostra che il Filtro di Kalman è uno stimatore ottimo, sotto opportune ipotesi. Ciò è reso

possibile dal fatto che il filtro utilizza tutte le informazioni che raccoglie. A prescindere dall'accuratezza e dalla precisione dei dati raccolti, il filtro li combina tutti per ottenere la migliore stima possibile per le variabili di interesse. Il filtro incorpora infatti informazioni sulla dinamica del sistema, sulle distribuzioni statistiche dei rumori di sistema e di misura, sull'incertezza del modello dinamico, e sulle condizioni iniziali delle variabili di interesse.

Il Filtro di Kalman è essenzialmente uno stimatore dello stato che opera su un meccanismo di *predizione - correzione*: il filtro calcola una stima dello stato, correggendo una predizione basata sulla dinamica del sistema con una misura del sistema. [MAY]

In virtù delle proprie caratteristiche, il Filtro di Kalman ha trovato utilizzo soprattutto nell'ambito del controllo e della predizione di sistemi dinamici lineari.

L'idea principale alla base del Filtro di Kalman è che esso calcoli lo stato *reale* del sistema. In particolare, il Filtro di Kalman stima lo stato e fornisce una misura di quanto sia certo che lo stato predetto sia effettivamente lo stato reale. La stima è resa difficoltosa dalla possibilità per lo stato di cambiare nel tempo e dai rumori. Inoltre le variabili di stato possono non essere osservabili, e i sensori che consentono di ottenere misure sullo stato sono anch'essi affetti da rumore.

4.1.1 – La stima

Formalmente il Filtro di Kalman calcola la probabilità condizionata di trovarsi nello stato x_k, date le misure $z_1, \ldots, z_k$. Si

definisce *stima* proprio la probabilità condizionata di trovarsi nello stato x_k date le misure $z_1, \dots, z_k$:

$$P(x_k \mid z_1,\dots,z_k)$$

Tale definizione di stima può essere separata in due stime distinte, servendosi della regola di Bayes, il teorema della probabilità totale e l'assunzione di Markov [NEG] :

$$Bel^-(x_k) = P(x_k \mid z_1,\dots,z_{k-1}) = \int_{\Xi} P(x_k \mid x_{k-1})\, Bel^+(x_{k-1})\, dx_{k-1}$$

$$Bel^+(x_k) = P(x_k \mid z_1,\dots,z_k) = \frac{P(z_k \mid x_k) Bel^-(x_k)}{P(x_k \mid z_1,\dots,z_k)}$$

$Bel^-(x_k)$ è detta *stima a priori*, e rappresenta la probabilità condizionata di trovarsi nello stato x_k date tutte le misure z fino al passo k escluso.

$Bel^+(x_k)$ è detta *stima a posteriori*, e rappresenta la probabilità condizionata di trovarsi nello stato x_k date tutte le misure z fino al passo k incluso.

4.1.2 – Predizione e correzione

Il Filtro di Kalman elabora la stima dello stato calcolando prima la stima a priori e poi la stima a posteriori.

Il calcolo della stima a priori può essere visto come la *predizione* del sistema dopo un passo temporale. Senza servirsi di nuove misure, la stima calcola una previsione di come dovrebbe essere lo stato del sistema dopo un passo temporale, usando il modello del sistema e la stima dello stato al passo precedente.

Poiché un sistema è generalmente soggetto a rumori, si possono verificare errori nella predizione. In questi casi la predizione può essere diversa dallo stato reale. Il calcolo della stima a posteriori può essere visto come una *correzione* dello stato stimato dalla predizione. Dopo che il filtro ha calcolato la stima a priori, nuove misure forniscono informazioni sullo stato reale del sistema, che possono essere usate per correggere la predizione dello stato. Il Filtro di Kalman utilizza un modello della misura che descrive come dovrebbe essere una misura z_k dato lo stato x_k.

Per poter produrre la stima, il filtro necessita quindi dei modelli del sistema e delle misure. A seconda della linearità o meno del sistema, si distingue tra Filtro di Kalman Lineare e Filtro di Kalman Esteso. [WEL]

Nei paragrafi successivi l'apice "+" indicherà sempre una quantità stimata attraverso la predizione, l'apice "–" una quantità cui è stata applicata la correzione dovuta alla misura corrente.

4.2 – Il Filtro di Kalman Lineare (LKF)

Il Filtro di Kalman Lineare (LKF) assume che lo stato del sistema e le misure possano essere descritte tramite un sistema dinamico lineare, ovvero un sistema formato da un insieme di equazioni lineari che modellano l'evoluzione dello stato nel tempo, e che descrivono in che modo le misure siano legate allo stato.

Un sistema dinamico lineare è caratterizzato essenzialmente da due modelli [GRW]:

- *Modello dello stato*: descrive in che modo lo stato reale del sistema si evolve nel tempo. LKF necessita di questo modello per poter effettuare previsioni sullo stato:

$$x_k = Ax_{k-1} + w_{k-1}$$
$$x_k \in \Re^n$$

Lo stato attuale del sistema dipende dallo stato al passo precedente dell'algoritmo più un rumore additivo. La matrice A è una matrice $n \times n$ che relaziona lo stato precedente con lo stato attuale senza tenere conto del rumore.

- *Modello della misura*: descrive in che modo le misure sono legate allo stato. LKF necessita di questo modello per correggere la predizione dello stato ogniqualvolta sia disponibile una nuova misura.

$$z_k = Hx_k + v_k$$
$$z_k \in \Re^m$$

La misura dello stato dipende dallo stato attuale, più un rumore additivo. La matrice H è una matrice $m \times n$ che relaziona lo stato attuale con la misura attuale senza tenere conto del rumore.

Generalmente sistema e misure sono soggetti a rumore. Nella modellazione del LKF si assume che il rumore sullo stato w_k e il rumore sulla misura v_k siano indipendenti, bianchi e con distribuzioni di probabilità Gaussiane a media nulla, ovvero il rumore sullo stato e il rumore sulla misura non si influenzano tra loro, sono casuali, e sono scorrelati nel tempo. Su queste assunzioni si definiscono:

$$w_k \sim N(0, Q_k)$$

$$v_k \sim N(0, R_k)$$

dove Q_k è la matrice di varianza dell'errore sullo stato al passo k, e R_k è la matrice di varianza dell'errore sulla misura. Poiché gli errori sono indipendenti, le matrici Q_k e R_k sono diagonali.

4.2.1 – Algoritmo

L'algoritmo LKF si compone di tre passi (*step*) principali:

- *Inizializzazione*: in questo *step* iniziale vengono forniti i valori delle matrici di evoluzione dello stato A_k e H_k e delle matrici di varianza degli errori su stato e misura Q_k e R_k. Si definisce inoltre P_k la matrice di covarianza dell'errore sulla stima. Il valore iniziale di P_k^+ è arbitrario, generalmente viene utilizzata una matrice unitaria. Per quanto riguarda le matrici Q_k e R_k, vengono mantenute fisse durante l'esecuzione dell'algoritmo, in virtù delle precedenti asserzioni sui rumori. Si utilizza inoltre un valore medio per lo stato iniziale x_0^+ per inizializzare nell'algoritmo il valore dello stato stimato all'istante $k-1$, con $k=0$.

- *Predizione*: in questo *step* il sistema può trovarsi in uno stato qualsiasi, di conseguenza LKF calcola una nuova stima a priori ad ogni passo, aggiornando la matrice di covarianza P_k^-

$$x_k^- = Ax_{k-1}^+$$
$$P_k^- = AP_{k-1}^+A^T + Q_{k-1}$$

- *Correzione*: questo *step* viene eseguito solo quando una nuova misura è disponibile:

$$x_k^+ = x_k^- + K_k(z_k - Hx_k^-)$$
$$P_k^+ = (I - K_kH)P_k^-$$
$$K_k = P_k^- H^T(HP_k^-H^T + R_k)^{-1}$$

  K_k è detto *guadagno di Kalman*, ed è una matrice $n \times m$ di correzione della stima che pesa il contributo della misura rispetto al contributo della stima a priori.
  La nuova stima a posteriori ottenuta verrà impiegata nel calcolo della stima a priori nel successivo *step* di predizione.

4.3 – Il Filtro di Kalman Esteso (EKF)

Molti sistemi dinamici e modelli sensoriali non sono lineari. Questo significa che le funzioni che descrivono stato e misura non sono lineari, ma solo approssimativamente lineari per piccole variazioni dei valori delle variabili di stato. [WEL]

Si assume ora di avere un sistema dinamico non lineare, costituito da due modelli:

- *Modello dello stato non lineare*: descrive in che modo lo stato reale del sistema si evolve nel tempo, ma a differenza di LKF in EKF lo stato è governato da un'equazione non lineare:

$$x_k = f(x_{k-1}) + w_{k-1}$$

  dove $f(\cdot)$ è una funzione non lineare che mette in relazione lo stato al tempo precedente con lo stato attuale. Si assume che il rumore w sia come nel caso del LKF indipendente,

bianco, a media nulla e a distribuzione di probabilità Gaussiana.

- *Modello della misura non lineare*: descrive in che modo le misure sono legate allo stato:

$$z_k = h(x_k) + v_k$$

dove $h(\cdot)$ è una funzione non lineare che mette in relazione lo stato attuale con la misura. Analogamente al modello dello stato si assume che il rumore v sia indipendente, bianco, a media nulla e a distribuzione di probabilità Gaussiana.

4.3.1 – Algoritmo

L'algoritmo EKF si compone di tre passi (*step*) principali:

- *Inizializzazione*: come nel caso LKF in questo *step* iniziale vengono forniti i valori iniziali di P_k^+ e x_k^+ per il passo $k=0$.

- *Predizione*: in questo *step* EKF propaga stato e incertezza del sistema attraverso le equazioni di predizione:

$$x_k^- = f(x_{k-1}^+)$$
$$P_k^- = A_k P_{k-1}^+ A_k^T + Q_{k-1}$$

dove A_k è la matrice Jacobiana contenente le derivate parziali rispetto allo stato x della funzione di sistema $f(\cdot)$:

$$A_k = \left.\frac{\partial f(x)}{\partial x}\right|_{x=x_{k-1}^{+}}$$

- *Correzione*: questo *step* viene eseguito solo quando una nuova misura è disponibile:

$$x_k^{+} = x_k^{-} + K_k(z_k - h(x_k^{-}))$$
$$P_k^{+} = (I - K_k H)P_k^{-}$$
$$K_k = P_k^{-} H^T (HP_k^{-} H^T + R_k)^{-1}$$

  in cui H_k è la matrice Jacobiana contenente le derivate parziali rispetto allo stato x della funzione di misura $h(\cdot)$:

$$H_k = \left.\frac{\partial h(x)}{\partial x}\right|_{x=x_k^{-}}$$

Le matrici Jacobiane A_k e H_k utilizzate in EKF sono valutate sulla stima più recente dello stato, e devono quindi essere calcolate *on-line* aumentando i costi computazionale dell'algoritmo, migliorando però l'accuratezza della stima rispetto a metodi alternativi che calcolano *off-line* le matrici Jacobiane (come nel caso del *Perturbation Kalman Filter*, che calcola lo stato di un sistema non lineare linearizzando le sue non-linearità [MAY]).

4.4 –LKF e EKF a confronto

Per definizione LKF presenta le seguenti caratteristiche [KAL]:

- il filtro è ricorsivo;
- il filtro è lineare nello stato e nelle misure;

- il filtro minimizza la varianza dell'errore di stima, poiché il sistema è lineare e il rumore è Gaussiano bianco;
- il filtro è ottimo;
- tutte le variabili aleatorie utilizzate nel filtro sono Gaussiane, in quanto trasformazioni lineari di variabili aleatorie Gaussiane;
- nessun filtro non lineare può fare meglio;
- la ricorsività del filtro consente di processare ad ogni passo soltanto la misura relativa a quel passo, ottenendo lo stesso risultato che si otterrebbe processando tutte le misure contemporaneamente, quindi ad ogni passo viene estratta tutta l'informazione contenuta nelle misure.

Nel caso di EKF la non linearità dello stato e della misura cambiano le caratteristiche del filtro rispetto a LKF:

- il filtro è ricorsivo;
- il filtro non è lineare nello stato e nelle misure;
- le variabili aleatorie utilizzate nel filtro non sono Gaussiane;
- non si può assicurare l'ottimalità;
- non si può assicurare la convergenza;
- la matrice K_k dei guadagni di Kalman è ottenuta componendo le matrici A_k e H_k ottenute linearizzando intorno alla stima di predizione. Poiché tale stima è una variabile aleatoria, anche la matrice K_k è una variabile aleatoria.

4.5 – Localizzazione tramite landmark e filtro di Kalman

Normalmente quando vengono utilizzati i landmark per la localizzazione si dispone di una mappa contenente l'ubicazione di ogni landmark in coordinate globali. Le misure invece forniscono l'ubicazione dei landmark secondo il punto di vista del robot. Si rende quindi necessaria un'operazione di corrispondenza tra mappa e misure. Generalmente è il robot stesso a dover stabilire la relazione tra i landmark misurati e quelli mappati, attraverso un *modello di corrispondenza*, con cui vengono identificati sulla mappa i landmark corrispondenti ai dati grezzi acquisiti dai sensori. Assumendo che tale modello identifichi univocamente i landmark cui si riferiscono le misure, si possono opportunamente istanziare le equazioni di correzione del Filtro di Kalman:

$$x_k^+ = x_k^- + K_k\,(z_k - h(x_k^-, l_k))$$

$$H_{x,k} = \left[\begin{array}{ccc} \frac{\partial h_x}{\partial x_{[x]}} & \frac{\partial h_x}{\partial x_{[y]}} & \frac{\partial h_x}{\partial x_{[\phi]}} \\ \frac{\partial h_y}{\partial x_{[x]}} & \frac{\partial h_y}{\partial x_{[y]}} & \frac{\partial h_y}{\partial x_{[\phi]}} \\ \frac{\partial h_\phi}{\partial x_{[x]}} & \frac{\partial h_\phi}{\partial x_{[y]}} & \frac{\partial h_\phi}{\partial x_{[\phi]}} \end{array} \right]_{x=\hat{x}_k^-, l=\hat{l}_k}$$

$$= \left[\begin{array}{ccc} -\cos(x_{[\phi]}) & -\sin(x_{[\phi]}) & \begin{array}{l} -(l_{[x]} - x_{[x]})\sin(x_{[\phi]}) \\ +(l_{[y]} - x_{[y]})\cos(x_{[\phi]}) \end{array} \\ \sin(x_{[\phi]}) & -\cos(x_{[\phi]}) & \begin{array}{l} (x_{[x]} - l_{[x]})\cos(x_{[\phi]}) \\ -(l_{[y]} - x_{[y]})\sin(x_{[\phi]}) \end{array} \\ 0 & 0 & -1 \end{array} \right]_{x=\hat{x}_k^-, l=\hat{l}_k}$$

dove l_k è la posizione del landmark, utilizzata come ingresso nel modello della misura. Con questo modello è possibile stimare la posizione del robot in un ambiente con landmark riconoscibili. Viene quindi fornita al robot una lista dei landmark presenti nell'ambiente ed eventualmente l'incertezza associata a ciascuno

di essi. In presenza di landmark l'incertezza nella stima tende a decrescere, mentre aumenta quando il robot non è in grado di riconoscere alcun landmark.

Quando nel caso generale il modello della misura è non lineare, la non linearità causa errori nella linearizzazione della stima. Modificare le coordinate x e y dello stato iniziale portano a risultati simili nella modifica della stima, poiché la non linearità è essenzialmente determinata dalla coordinata θ. Per questo motivo un'orientazione iniziale del robot distante da quella reale causa a EKF significativi problemi di stima. [NEG]

Capitolo 5
Descrizione del sistema realizzato

5.1 – Introduzione al sistema

La realizzazione di un sistema di autolocalizzazione mediante riconoscimento di feature visive ha posto come primo passo la scelta di un impianto di visione adeguato. Sebbene sia possibile reperire sul mercato attrezzature di grande qualità e precisione, quali *framegrabber* e telecamere digitali, l'impianto di visione del sistema è stato affidato a una comune webcam. Una webcam è un dispositivo di acquisizione video molto semplice, generalmente utilizzato per funzioni di videoconferenza via *internet*, o comunque in casi nei quali non sia richiesta una grande qualità di immagine. Altra caratteristica della webcam è la quasi totale assenza di calibrazione manuale da parte dell'utente. Generalmente in una webcam il bilanciamento dei parametri di acquisizione (luminosità, saturazione, colori, ecc.) è automatico, senza possibilità di intervento dell'utente. Ciononostante la scelta effettuata non è da intendersi del tutto limitante. Le webcam moderne, grazie al notevole progresso tecnologico che le ha interessate, sono comunque dotate di una qualità di immagine decisamente buona, e possono essere usate con successo in molti sistemi di elaborazione di immagini. A differenza del passato, recentemente le webcam hanno anche raggiunto il pieno supporto in ambiente Linux, sistema operativo ideale per la realizzazione di applicazioni *real time*. Il costo irrisorio rispetto ad hardware più complesso rende inoltre le webcam uno strumento decisamente

interessante per esperimenti di acquisizione e trattamento di dati video.

Per quanto riguarda invece il robot in se, il progetto si è appoggiato al robot TRC – LabMate, un'architettura hardware commerciale, già in uso presso il laboratorio.

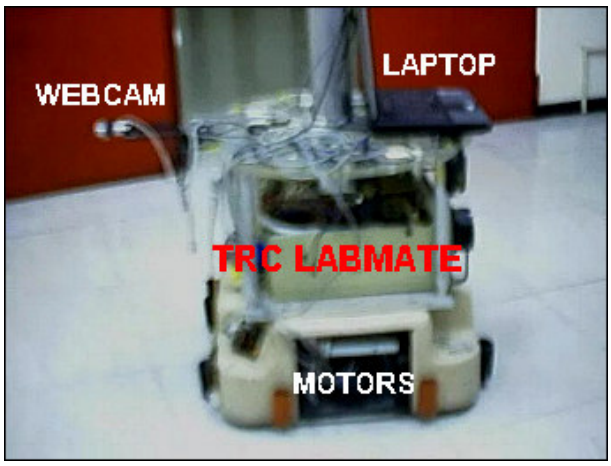


Fig. 22 – Particolare del robot TRC LabMate con evidenziata la strumentazione di bordo.

Con l'hardware a disposizione si è scelto di realizzare un sistema di navigazione seguendo la classica architettura *position-based control* (cap.1), in cui la localizzazione avviene in maniera assoluta tramite il riconoscimento di landmark (cap.2.3). Come elemento di landmark è stato scelto di identificare gli incroci delle *fughe* delle piastrelle del pavimento di lavoro del robot, posizionando la webcam in una posizione opportuna per inquadrare il pattern di lavoro. La scelta non è casuale, e si basa sull'osservazione che in ambienti molto standardizzati (quali uffici, ospedali e scuole, per fare alcuni esempi) la ripetitività degli schemi dei pavimenti è in generale alta; in questi casi i pavimenti sono tipicamente composti da piastrelle semplici e identiche, le cui fughe diventano quindi un elemento distintivo facilmente individuabile ed analizzabile.

Potendo inquadrare gli incroci dall'alto, si evitano inoltre i rischi di occlusione temporanea delle feature, un problema che affligge molte delle più diffuse tecniche di riconoscimento. Gli incroci di intersezione tra le fughe delle piastrelle dei pavimenti sono quindi a tutti gli effetti elemento distintivo dell'ambiente, e rappresentano perfettamente un esempio di *junction*, ovvero una tipica feature appartenente alla categoria dei corner. Come visto nel cap. 3, la rilevazione di corner da immagini è un'operazione relativamente non difficile, servendosi di tecniche opportune. La possibilità di lavorare su immagini in toni di grigio (cap. 3.3) alleggerisce inoltre il carico computazionale generale del sistema. Basandosi su queste considerazioni si è quindi scelto di realizzare il sistema, che idealmente appare in grado di aiutare con successo il robot nel difficile compito della localizzazione.

5.1.1 – Cinematica del sistema

Il sistema di navigazione è stato realizzato appoggiandosi all'architettura software ERASMUS (Expert-based Robotic Architecture for Sensing and Moving in Unknown Surroundings), sviluppata in loco nel laboratorio. ERASMUS è il software che si occupa dell'interazione con i diversi robot in uso nel laboratorio, tra cui il robot TRC LabMate utilizzato durante i test di sistema, di cui viene di seguito mostrata una schematizzazione della cinematica:

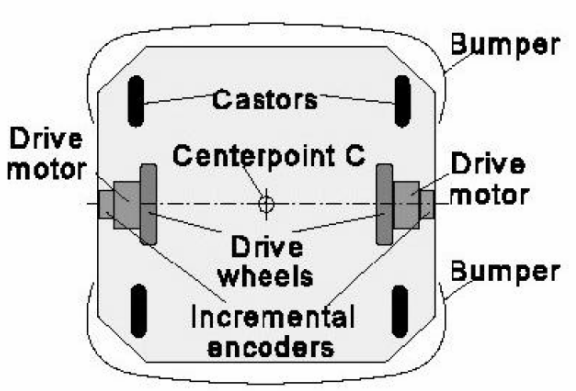


Fig. 23 - Schema della cinematica del robot LabMate.

La cinematica del robot segue il modello dell'uniciclo a guida differenziale, in cui gli *encoder* sono montati direttamente sulle ruote del robot. La cinematica inversa del robot viene approssimata attraverso le seguenti equazioni:

$$\begin{cases} x_k = x_{k-1} + ds_{k-1} * \cos\vartheta_{k-1} \\ y_k = y_{k-1} + ds_{k-1} * \sin\vartheta_{k-1} \\ \vartheta_k = \vartheta_{k-1} + d\omega_{k-1} \end{cases} \qquad \begin{aligned} ds &= (dr + dl)/2 \\ d\omega &= (dr - dl)/D \end{aligned}$$

dove x_k, y_k e θ_k rappresentano come di consueto la posizione e l'orientazione del robot, mentre ds rappresenta lo spostamento lineare incrementale del centro del robot, e $d\omega$ lo spostamento angolare (ovvero l'incremento di angolazione). D è la lunghezza dell'asse delle ruote, dl e dr sono invece rispettivamente gli spostamenti delle ruote di sinistra e destra. Le equazioni servono a calcolare la stima a priori della posizione e dell'orientazione del robot attraverso la ricostruzione odometrica (cap. 2.2.1.1).

5.1.2 – Il sistema di visione

Il sistema di visione è affidato a una webcam Philips ToUCam Pro II, capace di raggiungere una risoluzione massima di 640×480 pixel e un *frame rate* di 30 FPS. La comunicazione tra webcam e sistema avviene attraverso un'interfaccia USB 1.1, capace di trasferire dati alla velocità massima di 12 Mbps (Megabit per secondo). La webcam è montata su una staffa in posizione tale da inquadrare perpendicolarmente il pavimento, ad un'altezza di circa 65 cm da terra. La staffa rende il centro dell'immagine acquisita dalla webcam spostato rispetto al centro del robot di circa -62,5 cm lungo l'asse x e di 1,5 cm lungo l'asse y.

5.1.3 – La fusione sensoriale

La misura fornita dalla webcam e i dati forniti dalla stima a priori mediante la ricostruzione odometrica sono fusi assieme attraverso l'utilizzo di un Filtro di Kalman (cap.4):

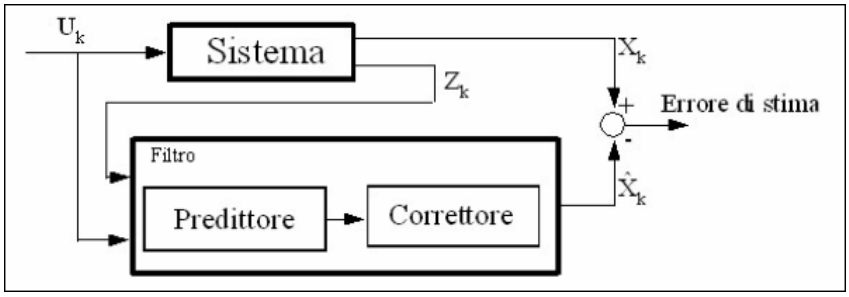


Fig. 24 – Schema generale di filtraggio per il Filtro di Kalman.

Poiché le equazioni della stima a priori e della misura non sono lineari, tenendo conto della presenza di errori possiamo definire le equazioni che governano il processo nel seguente modo:

$$\mathbf{x}_k = \mathrm{f}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{w}_{k-1}) \quad with \quad \mathbf{x} \in \Re^3$$

$$\mathbf{z}_k = \mathrm{h}(\mathbf{x}_k, \mathbf{v}_k) \quad with \quad \mathbf{z} \in \Re^2$$

dove u_k è la funzione di guida bidimensionale relativa alle due ruote, ω_k e υ_k sono i consueti rumori su stato e misura.
In particolare:

$$\begin{cases} ds = (dr + wr + dl + wl)/2 \\ d\omega = (dr + wr - dl - wl)/D \end{cases}$$

considerando i rumori su processo e misura a media nulla e con distribuzioni di probabilità gaussiana Q ed R, rispettivamente. Tali assunzioni sui rumori fanno sì che si possano trascurare errori sistematici legati all'approssimativa conoscenza geometrica del robot, rendendo necessario sovrastimare la matrice Q. Per i nostri scopi comunque la semplificazione è da considerarsi accettabile. Certamente un approccio basato su stato aumentato sarebbe più accurato, ma non strettamente necessario per mostrare le capacità di localizzazione del sistema

A causa delle non linearità finora descritte, ne consegue che è necessario utilizzare nel sistema il Filtro di Kalman nella sua versione Estesa (cap.4.3). Questa scelta, inevitabile, porta alla perdita delle garanzie di convergenza e ottimalità delle stime.

Tuttavia, come è stato discusso nel capitolo 4, le caratteristiche del filtro rendono comunque EKF una scelta privilegiata nella stima di sistemi non lineari.

5.2 – Implementazione dell'estrattore di Harris

L'estrattore è stato implementato seguendo l'idea dell'algoritmo originale di Harris (Cap. 3.3.1). L'algoritmo presenta una serie di calcoli decisamente complessi e pesanti per l'esecuzione su immagini ricevute a frequenza elevata, si è reso perciò necessario ridurre al minimo il numero di operazioni da eseguire, allo scopo di utilizzare efficacemente l'algoritmo in *real time*. Si è quindi scelto di realizzare la costruzione della *Local Structure Matrix* attraverso l'applicazione all'immagine del *Robert Cross Operator*, che come visto nel cap. 3.2.1.3 consente di calcolare le derivate parziali velocemente. L'immagine in input è inoltre pre-trattata con un filtro di Sobel aggiuntivo, allo scopo di evidenziare gli edge. Nel caso del pavimento gli edge corrispondono alle fughe delle piastrelle, il filtraggio dovrebbe quindi consentire di migliorare ulteriormente l'estrazione dei corner. Il valore di D (relativo al *neighbourhood* di analisi di ogni pixel) è stato infine scelto pari a 3, garantendo una buona accuratezza di analisi senza gravare troppo sul sistema.

I valori relativi alle soglie sono stati scelti pari a $H_{thr} = 1000000$ e $\alpha = 0.20$, allo scopo di assicurare una buona robustezza e rigidità generale nell'individuazione del corner. E' opportuno scegliere tali valori in base a risultati sperimentali, a seconda delle caratteristiche dell'ambiente di utilizzo e della qualità della webcam impiegata. Poiché sporadicamente si presentano rumori sui bordi delle immagini acquisite, si è anche scelto di trascurare dall'analisi 30 pixel di bordo in orizzontale e in verticale, indipendentemente dalla risoluzione. L'estrattore è implementato in modo da restituire in output solo il corner dall'intensità più alta tra tutti i corner candidati.

5.3 – Implementazione del filtro di Kalman

Il Filtro di Kalman Esteso attraverso cui vengono prodotte le stime dello stato del sistema è stato realizzato seguendo lo schema originale presentato nel cap. 4.3, tenendo conto delle approssimazioni discusse nel cap. 5.1.3. Si presentano di seguito le scelte operative nella definizione e inizializzazione degli elementi costitutivi del filtro.

5.3.1 – Matrici e vettori del sistema

Mantenendo analogia con la simbologia introdotta nel capitolo 4, si definiscono le principali grandezze presenti nel sistema:

- X(k), vettore 3×1, composto dalle componenti x, y e θ dello stato del sistema;
- K(k), matrice 3×2, relativa ai guadagni di Kalman;
- H(k), matrice 2×3, Jacobiano della funzione h rispetto allo stato X;
- X_m(k), vettore 3×1, stima a priori dello stato fornita dall'odometria;
- P(k), matrice 3×3, covarianza a priori dell'errore sullo stato;
- P_m(k), matrice 3×3, covarianza a posteriori dell'errore sullo stato;
- R(k), matrice 2×2, covarianza del rumore sulla misura;
- Q(k), matrice 3×3, covarianza del rumore di processo;
- Z(k), vettore 2×1, composto dalle componenti x e y della misura;
- Z_{mm}(k), vettore 2×1, rappresenta il modello della misura, linearizzazione della funzione non lineare h;

Il simbolo (k) indica naturalmente che i valori si riferiscono al passo k.

Le equazioni di aggiornamento del Filtro di Kalman Esteso risultano pertanto:

- *Predizione*: $X_m(k) = f(X(k-1))$
  $P_m(k) = A\ P(k-1)\ A^T + Q(k)$

- *Correzione*: $X(k) = X_m(k) + K(k)\ [Z(k) - Z_{mm}(k)]$
  $P(k) = (I - K(k)\ H(k))\ P_m(k)$
  $K(k) = P_m(k)\ H(k)^T\ (H(k)\ P_m(k)\ H(k)^T + R(k))^{-1}$

Lo stato $X_m(k)$ è il risultato della linearizzazione eseguita dal modulo di ERASMUS che si occupa della ricostruzione odometrica; viene posto A≡I, dove A è la consueta matrice di evoluzione dello stato e I è la matrice identica.

I dati che verranno di seguito presentati, ove non diversamente indicato, saranno da ritenersi sempre riferiti a grandezze calcolate in base alla posizione del robot espressa in millimetri e all'orientazione espressa in radianti.

5.3.2 – Parametri del filtro

Le matrici R, Q e P_0 sono i parametri del filtro che necessitano di un'inizializzazione a priori.

Per quanto riguarda il valore di R, solitamente viene scelto attraverso l'analisi di campioni di misura calcolati off-line. La determinazione di Q è invece più complessa, perché non sempre è possibile osservare direttamente il processo da stimare. Le massime performance del filtro si ottengono spesso settando proprio in modo opportuno R e Q. Si osserva inoltre che sotto le

ipotesi in cui Q ed R rimangono costanti (rumori gaussiani bianchi a media nulla) la covarianza dell'errore di stima P e i guadagni K si stabilizzano velocemente e rimangono anch'essi su valori pressoché costanti durante l'esecuzione del filtro.

Attraverso ripetute prove *off-line*, è stato individuato un buon valore di R pari a:

$$R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$$

Per quanto riguarda P_0, non si può porre $P_0 \equiv 0$ poiché non si è assolutamente certi che la posizione iniziale sia esatta, a causa dell'impossibilità di posizionare manualmente il robot con accuratezza. Tramite varie prove si è individuato un buon valore di P pari a:

$$P = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$$

Assumendo una trascurabile covarianza del rumore di processo, è stato infine posto:

$$Q = \begin{bmatrix} 0.09 & 0 & 0 \\ 0 & 0.09 & 0 \\ 0 & 0 & 0.09 \end{bmatrix}$$

5.3.3 – Il modello della misura

I vettori Z e Z_{mm}, che intervengono nel calcolo dell'innovazione nel Filtro di Kalman, sono rappresentati in unità di misura (il pixel)

relativa al piano immagine della webcam. E' stato calcolato un opportuno coefficiente di scala per convertire correttamente valori espressi in pixel in valori espressi in millimetri.

Il modello della misura Z_{mm} è stato realizzato attraverso l'implementazione di una funzione c++, chiamata *Feature Prevision*, in grado di ricostruire a priori il pattern di corner presenti nell'ambiente di lavoro di LabMate. La funzione restituisce, dati la posizione e l'orientamento correnti, la posizione nel piano immagine in cui dovrebbe essere visto il corner. Feature Prevision lavora correttamente sotto le seguenti ipotesi:

- Il robot deve essere avviato con il centro odometrico corrispondente a un corner del pattern, con gli assi x e y orientati parallelamente alle fughe del pavimento. Tale corner corrisponderà per il robot all'origine odometrica $O(0,0,0)$ del mondo.

- Il pattern è costituito da uno schema ripetuto di piastrelle rettangolari o quadrate, tutte della medesima dimensione.

Feature Prevision necessita a priori solamente dell'impostazione delle dimensioni delle piastrelle che costituiscono il pattern, e della definizione del fattore di scala. Naturalmente nel caso del robot LabMate è stato tenuto anche conto degli offset relativi alla posizione della webcam rispetto al centro odometrico del robot (cap. 5.1.2).

La matrice H, Jacobiano della funzione h rispetto allo stato X, assume la forma descritta nel cap. 4.5, con la differenza che in questo caso si tratta di una matrice 2×3:

$$H_{x,k}=\begin{bmatrix} -\cos(\vartheta_{odo}) & -\sin(\vartheta_{odo}) & -(x_{corner}-x_{odo})\cdot\sin(\vartheta_{odo})+(y_{corner}-y_{odo})\cdot\cos(\vartheta_{odo}) \\ \sin(\vartheta_{odo}) & -\cos(\vartheta_{odo}) & (x_{odo}-x_{corner})\cdot\cos(\vartheta_{odo})-(y_{corner}-y_{odo})\cdot\sin(\vartheta_{odo}) \end{bmatrix}$$

dove x_{odo}, y_{odo} e θ_{odo} rappresentano le coordinate dello stato corrente fornite dall'odometria, e x_{corner} e y_{corner} rappresentano le coordinate del corner restituite dalla webcam.

Capitolo 6
Analisi dei risultati sperimentali

6.1 – Test dell'estrattore di Harris

Per verificare la corretta implementazione dell'estrattore di Harris e le prestazioni dell'algoritmo è stata realizzata un'applicazione di testing, chiamata CrossFinder. CrossFinder è un'applicazione scritta in c++ e sviluppata in ambiente Ethnos (vedere Appendice A), costituita da 3 esperti che si occupano dell'interfacciamento con la webcam e dell'estrazione e visualizzazione del corner in un pannello grafico, chiamato CrossFinder Viewer (vedere Appendici B e C). CrossFinder Viewer presenta all'utente 2 finestre affiancate, in cui visualizza nella finestra di sinistra la posizione del corner sull'immagine corrente, mentre nella finestra di destra appare la sola posizione del corner su uno sfondo bianco, corrispondente all'area di cattura della webcam. Il riquadro rosso presente in entrambe le immagini visualizzate indica l'area effettiva di analisi dell'estrattore. Le aree di immagine al di fuori del riquadro rosso non vengono prese in considerazione a causa dei citati problemi di rumore ai bordi della webcam.

In ogni test sono stati processati 200 passi temporali, con un ritmo di acquisizione della webcam pari a 30 FPS. Sono state acquisite immagini della risoluzione di 320×240 pixel.

Ogni test si compone di diverse serie di acquisizioni, ciascuna delle quali si distingue per una diversa calibrazione dei parametri dell'algoritmo di Harris, e per le diverse condizioni di pulizia e illuminazione del pattern di analisi. Poiché sperimentalmente si verificano costantemente problemi di rumore nei primi frame

acquisiti dalla webcam (relativi all'autocalibrazione di luminosità e fuoco dell'obiettivo) si è scelto di mostrare la media e la varianza dei risultati ottenuti sia nelle serie complete, sia non considerando i primi 4 frame processati in ogni serie.

6.1.1 – Test della posizione del corner in condizioni di pattern molto sporco

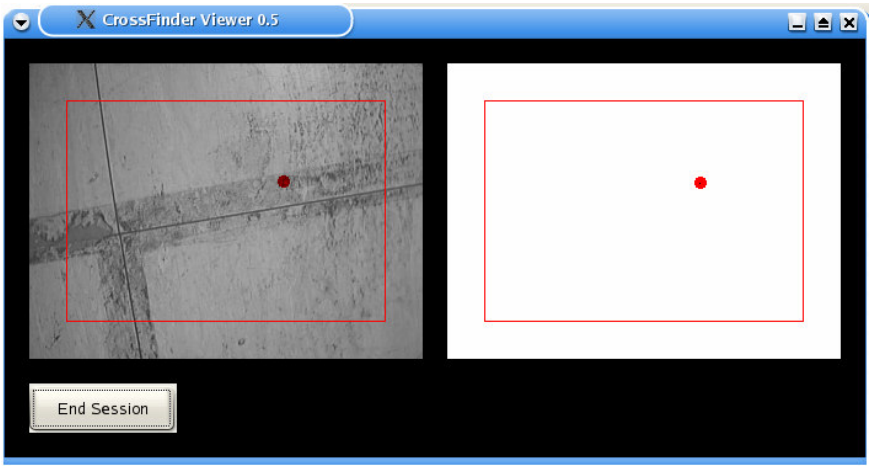


Fig. 25 - Output dell'applicazione di test, in condizioni di pavimento molto sporco con luce naturale. Si vedono chiaramente diffusi residui di colla lungo le fughe del pattern, e aree opache dovute ad usura e a sporcizia. Il punto rosso identifica la posizione del corner.

Le serie di dati sono raccolte in condizioni di pavimento molto sporco, luce naturale. L'algoritmo è inizializzato con i seguenti parametri:

Serie 1: $H_{thr} = 1000000$, $a = 0.24$

Serie 2: $H_{thr} = 1000$, $a = 0.01$

Serie 3: $H_{thr} = 100000$, $a = 0.20$

Serie 4: $H_{thr} = 800000$, $a = 0.24$

K=1 to 200	X		Y	
	media	varianza	media	varianza
Serie 2	148,11	912,812	122,27	110,4393
Serie 3	179,695	245,0271	128,925	92,87374
Serie 4	182	57,73869	130,4	4,462312
K=5 to 200	X		Y	
	media	varianza	media	varianza
Serie 2	147,9541	908,4133	123,1735	38,86206
Serie 3	179,5867	249,3822	129,8316	11,00228
Serie 4	183,0816	0,126635	130,6939	0,213501

Media e varianza delle rilevazioni di ascissa e ordinata corner nelle diverse serie di test. K rappresenta il passo temporale.

Questo test mostra come in condizioni di pattern molto sporco l'algoritmo presenti frequenti falsi riconoscimenti, come si può vedere dall'alta varianza dei risultati. Rafforzando i parametri di soglia la varianza cala vistosamente nei risultati relativi alla serie 4, tuttavia il corner riconosciuto non è quello voluto, ovvero l'in crocio delle fughe, ma un "falso" corner (come mostrato dall'immagine catturata con CrossFinder Viewer in Fig. 24). Non sono presenti i dati relativi alla serie 1 poiché non è stato riconosciuto alcun corner in nessuno dei 200 passi di esecuzione dell'algoritmo, mostrando che la soglia scelta era troppo alta in questo caso per consentire di rilevare un corner.

Per tutte le serie vengono di seguito presentati i grafici relativi all'andamento dell'ordinata del corner (in ascissa, misurata in pixel) rispetto al passo temporale (in ordinata, misurato in passi), allo scopo di mostrare l'oscillazione e l'instabilità del rilevamento della posizione del corner reale.

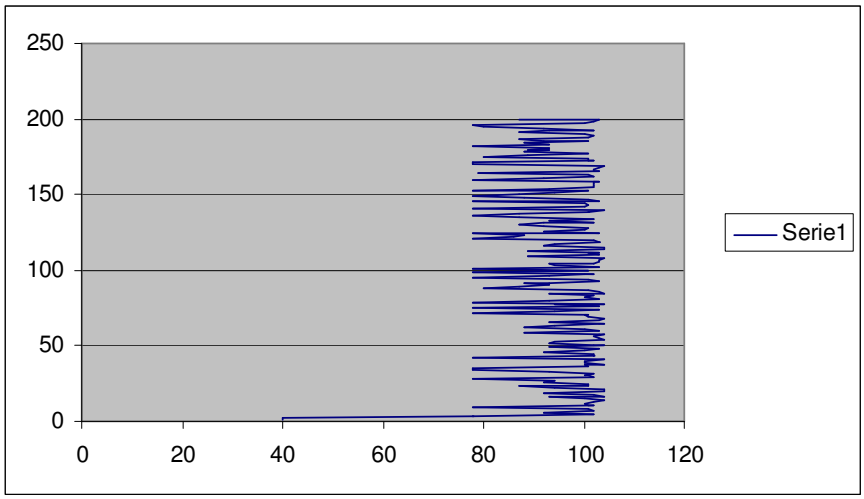


Fig. 26 - Serie 1: andamento dell'ordinata del corner rilevato (in pixel) rispetto al passo temporale.

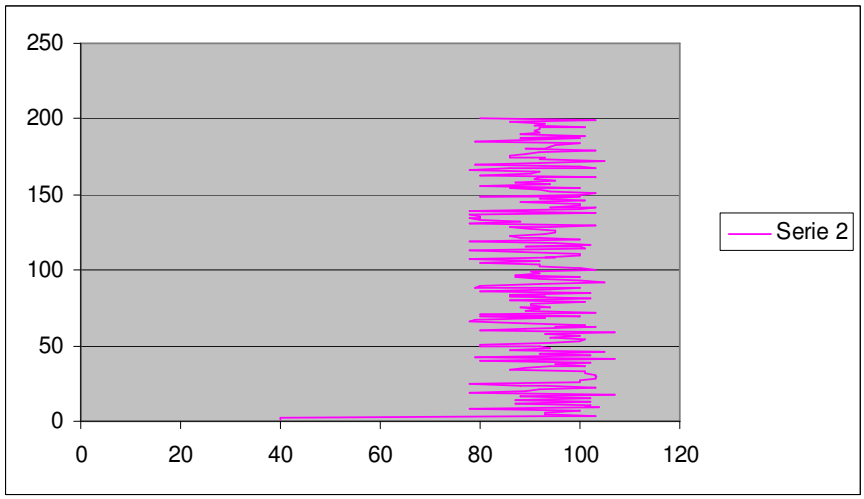


Fig. 27 - Serie 2: andamento dell'ordinata del corner rilevato (in pixel) rispetto al passo temporale.

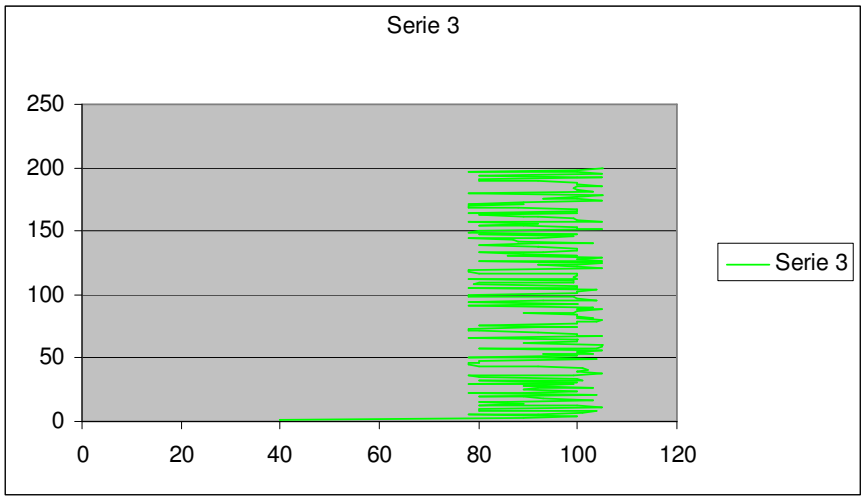


Fig. 28 - Serie 3: andamento dell'ordinata del corner rilevato (in pixel) rispetto al passo temporale.

Come si vede dai grafici, l'oscillazione nel rilevamento della posizione del corner è significativa in tutte le serie di dati raccolti.

6.1.2 – Test della posizione del corner in condizioni di pattern leggermente sporco

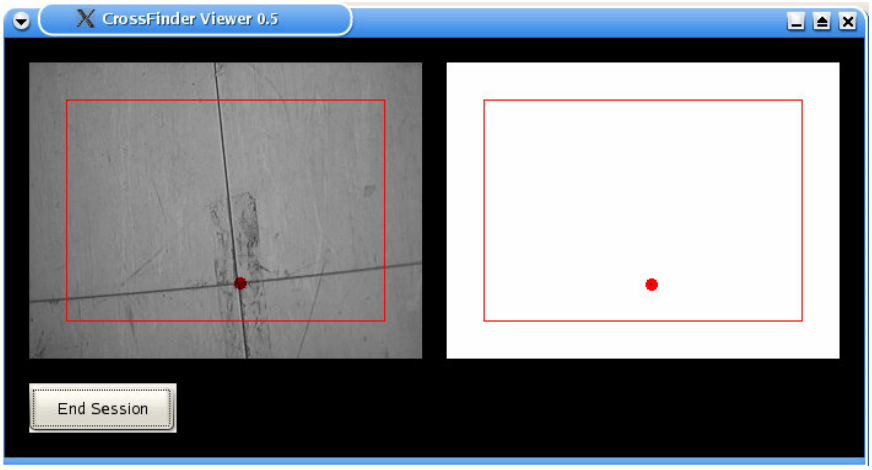


Fig. 29 - Output dell'applicazione di test, in condizioni di pavimento moderatamente sporco con luce naturale.

Le serie di dati sono raccolte in condizioni di pavimento moderatamente sporco, luce naturale. L'algoritmo è inizializzato con i seguenti parametri:

Serie 1 : $H_{thr} = 100000$, $a = 0.20$
Serie 2 : $H_{thr} = 1000$, $a = 0.01$
Serie 3 : $H_{thr} = 1000000$, $a = 0.24$

K=1 to 200	X		Y	
	media	varianza	media	varianza
Serie 1	173,065	116,6842	94,775	106,5069
Serie 2	170,56	81,79538	91,99	93,45719
Serie 3	171,095	120,8502	92,735	125,4621
K=5 to 200	X		Y	
	media	varianza	media	varianza
Serie 1	173,7513	84,68782	95,45596	75,1556
Serie 2	170,7202	77,00464	92,41451	67,54604
Serie 3	171,4352	113,695	93,30052	98,90922

Media e varianza delle rilevazioni di ascissa e ordinata corner nelle diverse serie di test. K rappresenta il passo temporale.

In questo test le condizioni di sporcizia del pattern influenzano ancora la varianza in modo significativo, come nel caso del Test n°1.

Per tutte le serie vengono di seguito presentati i grafici relativi all'andamento dell'ascissa del corner (in ascissa, misurato in pixel) rispetto al passo temporale (in ordinata, misurato in frame), allo scopo di mostrare l'oscillazione e l'instabilità del rilevamento della posizione del corner reale.

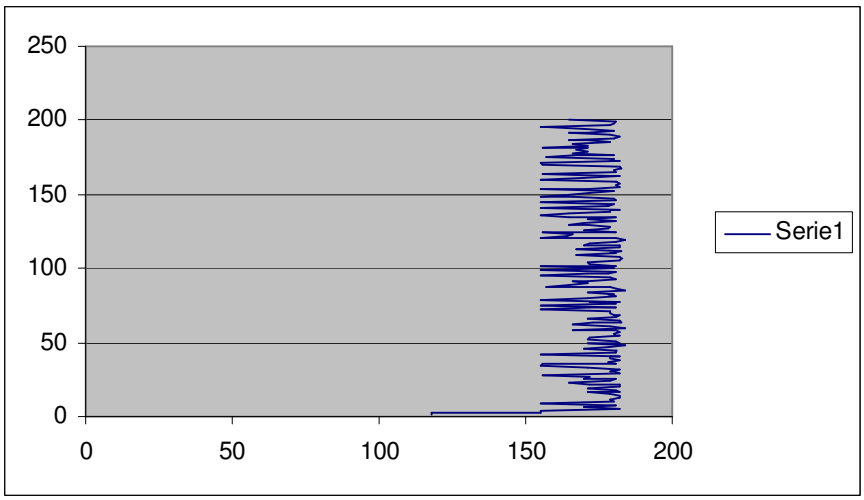


Fig. 30 - Serie 1: andamento dell'ascissa del corner rilevato (in pixel) rispetto al passo temporale.

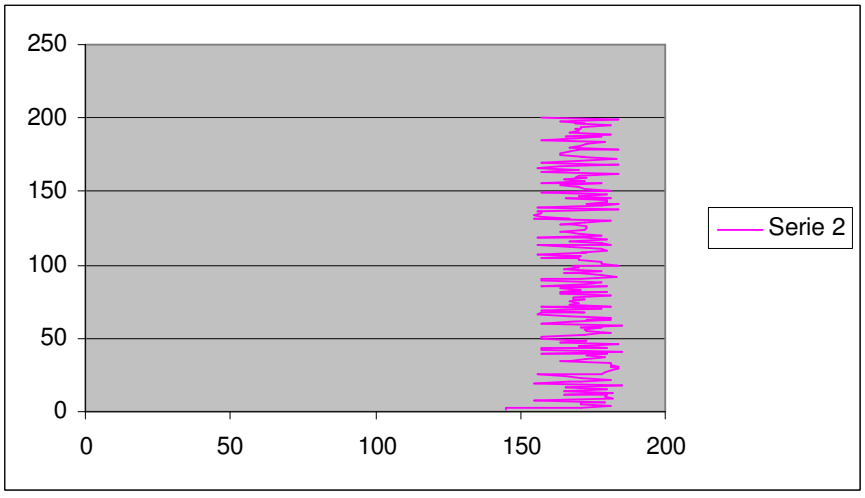


Fig. 31 - Serie 2: andamento dell'ascissa del corner rilevato (in pixel) rispetto al passo temporale.

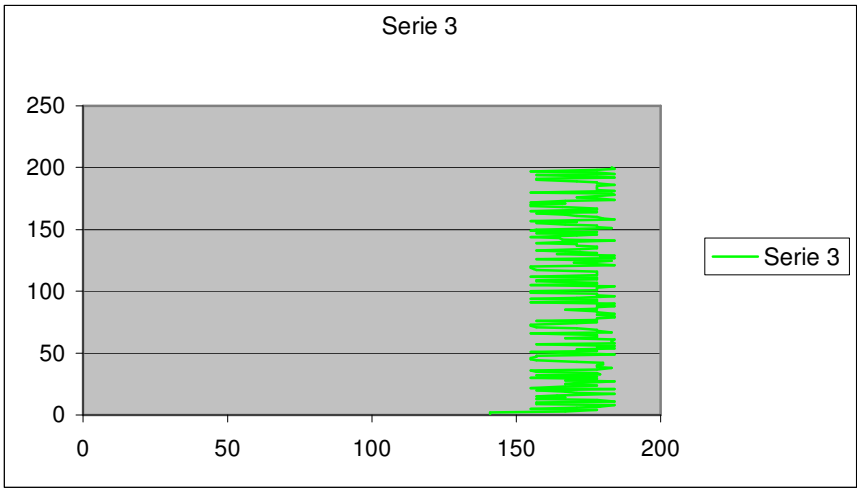


Fig. 32 - Serie 3: andamento dell'ascissa del corner rilevato (in pixel) rispetto al passo temporale.

Anche in questo test, come nel precedente, si verifica un'oscillazione notevole nel rilevamento della posizione del corner. Le pessime condizioni del pattern causano frequenti falsi rilevamenti, e quindi l'impossibilità di identificare il corner reale.

6.1.3 – Test della posizione del corner in condizioni di pattern pulito

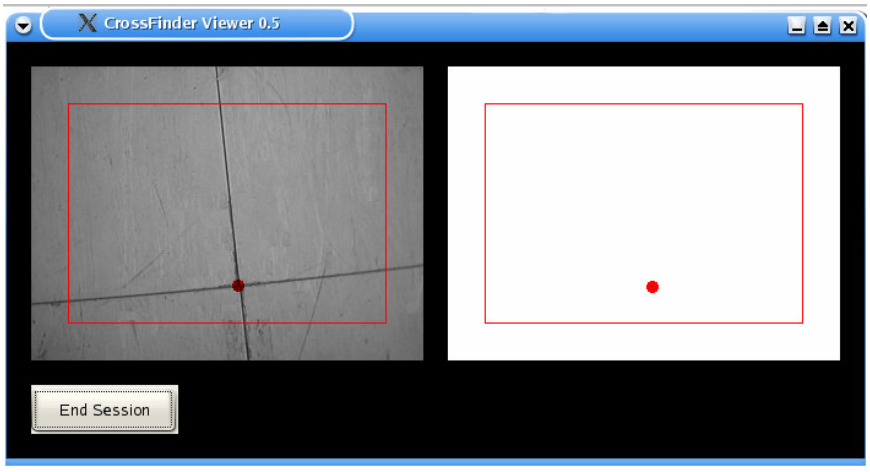


Fig. 33 - Output dell'applicazione di test, in condizioni di pavimento pulito con luce naturale. Il corner viene rilevato correttamente.

Le serie di dati sono raccolte in condizioni di pavimento pulito, luce naturale. L'algoritmo è inizializzato con i seguenti parametri:

Serie 1 : $H_{thr} = 100000,\ a = 0.20$
Serie 2 : $H_{thr} = 1000000,\ a = 0.24$
Serie 3 : $H_{thr} = 1000,\ a = 0.01$

K=1 to 200	X		Y	
	media	varianza	media	varianza
Serie 1	155,31	9,561709	116,22	60,53427
Serie 2	155,01	0,00995	116,19	62,08432
Serie 3	155,6	37,86935	113,48	43,78854
K=5 to 200	X		Y	
	media	varianza	media	varianza
Serie 1	155	0	117	0
Serie 2	155	0	116,9796	0,020094
Serie 3	156,199	0,560204	114,1327	0,24898

Media e varianza delle rilevazioni di ascissa e ordinata corner nelle diverse serie di test. K rappresenta il passo temporale.

I risultati mostrano che la rilevazione del corner è precisa e robusta in condizioni di pavimento pulito e luce naturale. La varianza è nulla o comunque molto bassa in tutte le serie (in particolare non considerando i primi 4 frame).

Vengono di seguito presentati i grafici relativi all'andamento di ascissa e ordinata del corner (espresse in pixel) in tutte e tre le serie. L'ordinata dei grafici rappresenta il passo temporale.

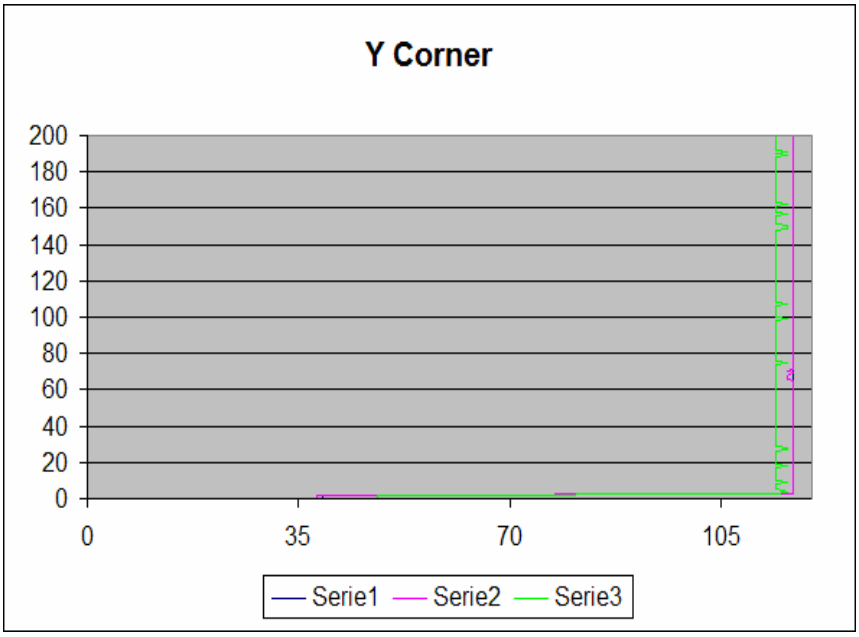


Fig. 34 - Andamento dell'ordinata del corner rilevato (in pixel) nelle tre serie di test. Le serie 1 e 2 si sovrappongono a partire dai primi passi di esecuzione dell'algoritmo, cioè le rilevazioni coincidono.

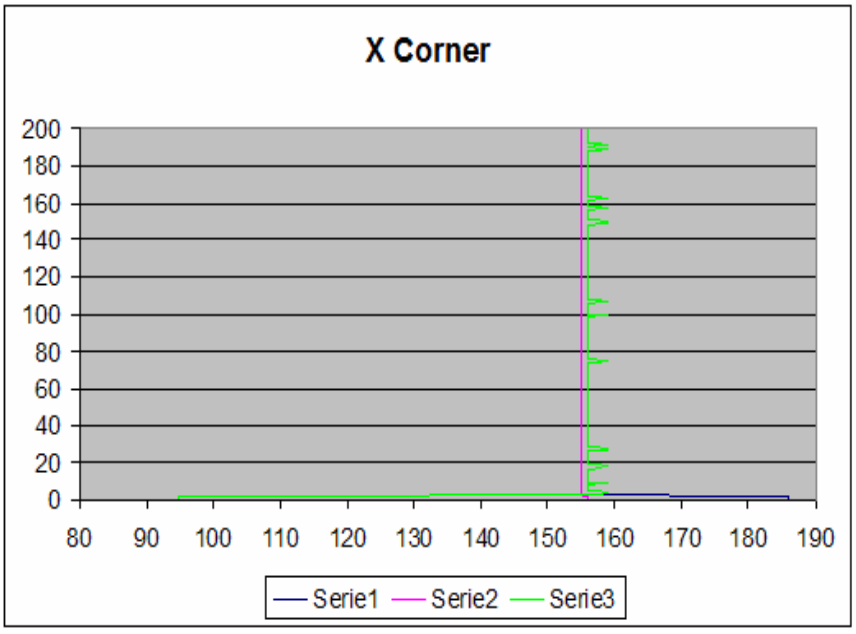


Fig. 35 - Andamento dell'ascissa del corner rilevato (in pixel) nelle tre serie di test. Le serie 1 e 2 si sovrappongono a partire dai primi passi di esecuzione dell'algoritmo.

In questo test le condizioni di buona pulizia del pattern consentono di rilevare il corner con grande precisione. I dati raccolti nella serie 3 sono interessanti, poiché mostrano come l'estrattore sia impreciso quando la soglia di Harris è bassa, e il parametro α è minimo. Per questo motivo l'algoritmo durante la serie 3 ha rilevato il corner in una posizione leggermente diversa dalle altre due serie, con sporadici falsi rilevamenti. La varianza in serie 3 è minima, ma comunque maggiore che nelle altre due serie di test. E' quindi verificata sperimentalmente la necessità do impostare un coefficiente $a > 0.10$ per ridurre la rilevazione di corner da parte dell'algoritmo, e quindi nel nostro caso aumentarne la precisione.

6.1.4 – Test della posizione del corner in condizioni di pattern pulito e luminosità proveniente da diverse direzioni

Nel seguente test vengono mostrate le prestazioni dell'algoritmo mantenendo fissi i parametri dell'estrattore di Harris e le condizioni del pavimento, variando però per ogni serie di dati le condizioni di illuminazione.

Parametri : H_{thr} = 100000, a = 0.20

K=1 to 200	X		Y	
	media	varianza	media	varianza
Serie 1	154,85	10,4397	115,68	54,20864
Serie 2	154,1	80,59296	118,23	55,95688
Serie 3	154,71	8,367739	118,29	50,15668
K=5 to 200	X		Y	
	media	varianza	media	varianza
Serie 1	155,17	0,141809	116,42	0,586533
Serie 2	155	0	118,98	0,019698
Serie 3	155	0	119	0

Media e varianza delle rilevazioni di ascissa e ordinata corner nelle diverse serie di test. K rappresenta il passo temporale.

Parametri : H_{thr} = 1000, a = 0.01

K=1 to 200	X		Y	
	media	varianza	media	varianza
Serie 1	153,9	120,392	118,25	55,96734
Serie 2	158,965	0,084196	115,19	63,68231
Serie 3	155,03	0,089548	118,19	65,2803
K=5 to 200	X		Y	
	media	varianza	media	varianza
Serie 1	155	0	119	0
Serie 2	158,985	0,045	115,99	0,02
Serie 3	155	0	119	0

Media e varianza delle rilevazioni di ascissa e ordinata corner nelle diverse serie di test. K rappresenta il passo temporale.

Parametri : H_{thr} = 1000000, a = 0.24

K=1 to 200	X		Y	
	media	varianza	media	varianza
Serie 1	155,42	33,47095	115,28	51,5795
Serie 2	153,925	120,4717	114,29	50,15668
Serie 3	154,94	1,765226	115,98	58,93427
K=5 to 200	X		Y	
	media	varianza	media	varianza
Serie 1	156	0	116	0
Serie 2	155,025	0,024497	115	0
Serie 3	155,07	0,065427	116,75	0,329146

Media e varianza delle rilevazioni di ascissa e ordinata corner nelle diverse serie di test. K rappresenta il passo temporale.

Anche in questo test la rilevazione del corner è precisa, ancora una volta la scelta dei parametri di Harris influenza leggermente la posizione del corner individuato, il cui valor medio varia da un test all'altro al massimo di soli 3 pixel circa.

6.1.5 – Test dei valori della funzione di Harris in differenti condizioni di luminosità

L'ultimo test eseguito sull'estrattore di corner ha avuto come oggetto di studio i valori restituiti dalla funzione di Harris. Si è voluto con questo test verificare se e in che modo i valori della funzione siano influenzati dalle condizioni di pulizia del pavimento, e soprattutto se questi valori varino significativamente cambiando le condizioni di luce. Come campione sono stati scelti 20 incroci di fughe del consueto pattern di riferimento, di cui 10 perfettamente puliti e 10 molto sporchi. Sono stati raccolti dati relativi al valore della funzione di Harris per ogni serie dapprima in condizioni di luce naturale, poi in condizioni di sola luce artificiale, con la fonte di luce posizionata a fianco della webcam di prova. Il parametro di Harris α è stato mantenuto fisso e pari a 0.20.

Nella tabella seguente vengono presentati i risultati sperimentali ottenuti relativi al valor medio, valore massimo, e valore minimo della funzione di Harris:

corner sporchi				
	naturale	artificiale		differenza
media	1058266874	1067003745		8736871
massimo	2140772761	2146644787		5872026
minimo	3355443	838860		-2516583
corner puliti				
	naturale	artificiale		differenza
media	1064139320	1064100366		-38954
massimo	2147483647	2147483647		0
minimo	1677722	1677721		-1

Risultati del test sui valori della funzione di Harris riscontrati nelle serie di campioni acquisiti. Vengono presentati il valore medio, il massimo e il minimo riscontrati sia in condizioni di luce naturale sia in condizioni di luce artificiale, e la differenza riscontrata.

I risultati del test mostrano che nel caso di corner pulito i valori della funzione di Harris non sono influenzati in maniera significativa dalla variazione di luminosità. Il valore massimo e minimo registrati rimangono praticamente identici, mentre il valore medio è solo leggermente più basso nel caso di luce artificiale. In ogni caso il valore minimo è comunque maggiore del valore 1000000 identificato dai test precedenti come soglia ideale per una rilevazione molto rigida del corner, fisso il parametro α al valore alto di 0.20.

Riguardo alla serie di corner sporchi, i risultati del test mostrano alcune significative variazioni nei risultati. Infatti in caso di luce artificiale il valor medio e il valor massimo della funzione di Harris sono molto più alti che nella condizione di luce naturale. Il valor minimo invece è molto più basso.

Confrontando tra loro i risultati ottenuti nelle due diverse serie di corner, non sembra comunque possibile né relazionare in qualche modo tra loro i risultati, né individuare dei valori tipici della funzione di Harris che consentano di distinguere a priori tra corner sporchi e corner puliti. Quanto emerge di importante da questo test è invece la conferma dei risultati del cap. 6.1.4, ovvero non c'è un'influenza significativa della luminosità del pattern sui risultati dell'estrazione nel caso in cui il pattern di lavoro è pulito.

6.1.6 – Conclusioni

L'estrattore di Harris si è dimostrato molto robusto e preciso nei casi in cui il pattern in esame è pulito e non alterato. Nei casi in cui il pattern è sporco l'algoritmo in genere individua comunque un corner, anche se non è quello relativo all'incrocio delle fughe. Ciò è naturale conseguenza del fatto che l'estrattore di Harris non individua solo *junctions*, ma rileva i corner in generale.

Naturalmente in caso di pattern pulito e assenza di corner l'estrattore non restituisce output, è stato verificato che in questi casi la funzione di Harris presenta valori sempre inferiori a zero.

I test hanno permesso di confermare i valori ottimali dei parametri dell'algoritmo scelti in fase di implementazione dell'estrattore, pari a $H_{thr} = 100000$ e $a = 0.20$.

L'ottimizzazione operata sul codice dell'algoritmo ha consentito di elaborare agevolmente immagini in 320×240 ad un *frame rate* di 30 FPS sulla macchina di prova (raggiungendo il limite costruttivo della webcam impiegata). Di seguito sono presentate le prestazioni riscontrate sulle diverse macchine di prova utilizzate, in termini di FPS:

Macchina	Max FPS
Athlon64 3400+ 2,2 Ghz, 512MB RAM	30 FPS
Duron 1 Ghz, 256MB RAM	20 FPS
Athlon K7 700 Mhz, 256MB RAM	15 FPS
Celeron 800 Mhz, 128 MB RAM	10 FPS

Fig. 36 - FPS massimi raggiunti da CrossFinder sulle varie macchine di prova utilizzate durante i test di laboratorio.La webcam impiegata poteva raggiungere fino a un massimo di 30 FPS.

In ogni caso per un utilizzo ottimale dell'estrattore all'interno di altre applicazioni è consigliabile ridurre il *frame rate* di acquisizione dalla webcam, soprattutto su macchine datate.

6.2 – Test del sistema in ambiente simulato

L'implementazione del filtro di Kalman è stata testata inizialmente tramite il software AESimServer [SGA], un'applicazione Ethnos sviluppata presso il Laboratorium in grado di riprodurre il comportamento dell'hardware a bordo di Labmate.

AESimServer è in grado di simulare il calcolo dell'odometria del robot, introducendo errori casuali per riprodurre le condizioni reali di utilizzo del robot. Per l'impiego di AESimServer si è reso necessario introdurre una sezione di codice per simulare l'acquisizione dell'immagine da parte della webcam. Un nuovo esperto riceve i dati odometrici da AESimServer, e calcola dove dovrebbe essere visto il corner nell'immagine simulata, svolgendo di fatto un ruolo molto simile a quello del modello della misura del Filtro di Kalman implementato. Il nuovo esperto è in grado inoltre di introdurre errori casuali che scostano la posizione del corner di alcuni pixel, per simulare i rumori e l'eventuale impurità del pattern di analisi, situazione che si presenta frequentemente in ambiente reale.

Nei test che vengono di seguito presentati si è voluto analizzare l'errore tra posizione reale e posizione stimata, allo scopo di verificare la precisione con cui il robot identifica la propria posizione. Viene calcolato in tutti i test l'errore di posizione medio e la varianza, e con l'ausilio di grafici viene data una verifica visiva delle traiettorie del robot secondo l'andamento della stima e della posizione reale.

6.2.1 – Test percorso rettilineo simulato

Il percorso simulato seguito dal robot in questo test è rettilineo e lungo 8 metri. Il robot si muove ripetutamente sul percorso tra

due vertici A e B, senza alcuna interazione manuale da parte dell'utente, basando il calcolo delle traiettorie sulla posizione indicata dalla localizzazione.

Nel complesso la stima si mantiene prossima al valore reale della posizione, come mostrato dal grafico seguente, in cui la linea scura rappresenta la traiettoria simulata, la linea chiara la traiettoria stimata dal filtro di Kalman, l'asse delle ascisse rappresenta l'ascissa della posizione del robot, l'asse delle ordinate rappresenta l'ordinata della posizione del robot:

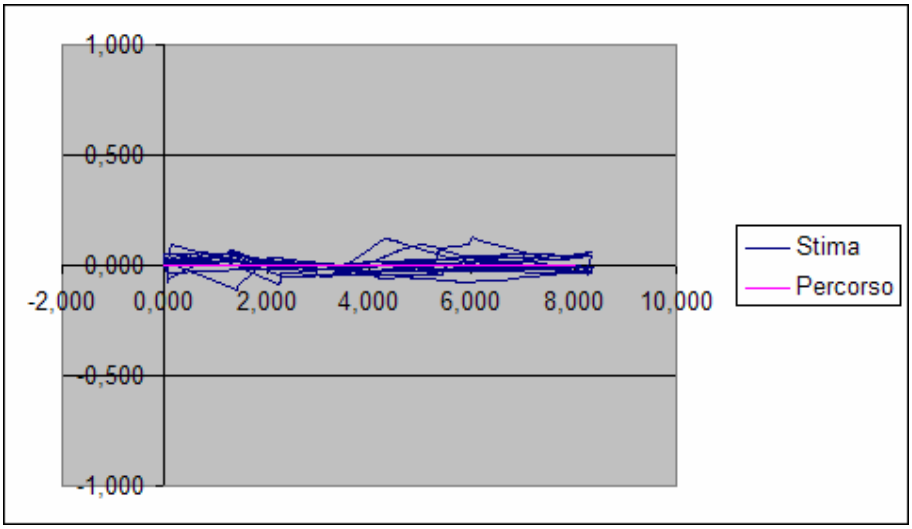


Fig. 37 - Traiettoria simulata (linea scura) e traiettoria stimata (linea chiara) a confronto.

Sono stati processati complessivamente 1600 passi temporali di filtraggio, simulando un ritmo di acquisizione della webcam pari a 25 FPS.

Vengono di seguito presentati i risultati relativi alla media dell'errore di stima sulle tre componenti dello stato del robot, e la rispettiva varianza:

errore medio			varianza		
X	Y	θ	X	Y	θ
0,00654	0,149949	0,2297	0,016784	0,09358	0,22969

Media (dati espressi in metri e radianti) e varianza degli errori di stima.

6.2.2 – Test percorso quadrato simulato

Il percorso seguito dal robot in questo test è un quadrato del lato di 1,8 metri. Il robot si muove ripetutamente sul percorso tra i 4 vertici A,B,C e D, senza alcuna interazione manuale da parte dell'utente, basando il calcolo delle traiettorie sulla posizione indicata dalla localizzazione.

Sono stati processati complessivamente 1600 passi temporali, simulando un ritmo di acquisizione della webcam pari a 25 FPS.

Nel complesso anche in questo caso la stima si mantiene prossima al valore reale della posizione, come mostrato dal grafico seguente, in cui la linea scura rappresenta la traiettoria simulata, la linea chiara la traiettoria stimata dal filtro di Kalman, l'asse delle ascisse rappresenta l'ascissa della posizione del robot, l'asse delle ordinate rappresenta l'ordinata della posizione del robot:

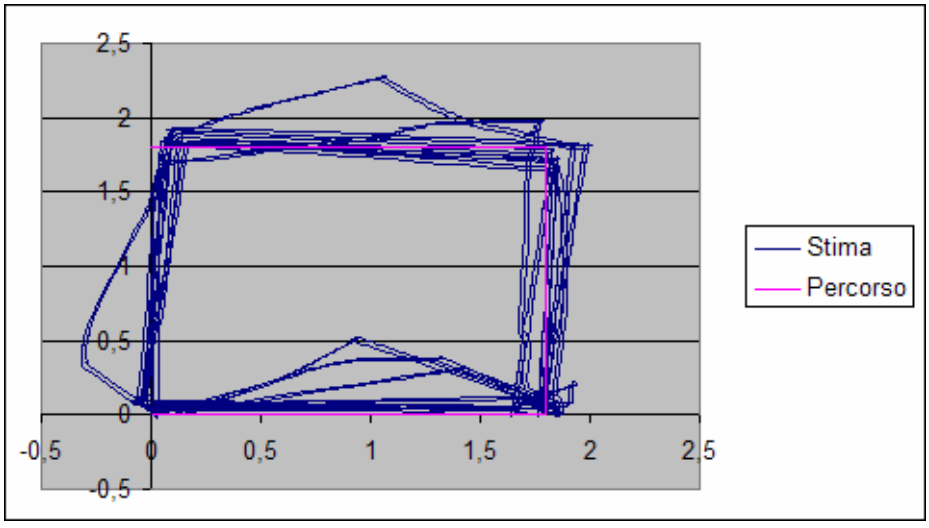


Fig. 38 - Traiettoria simulata (linea chiara) e traiettoria stimata (linea scura) a confronto.

Vengono di seguito presentati i risultati relativi alla media dell'errore di stima sulle tre componenti dello stato del robot, e la rispettiva varianza:

errore medio			varianza		
X	Y	θ	X	Y	θ
0,02199	0,23809	0,04721	0,279019	0,042108	0,8034

Media (dati espressi in metri e radianti) e varianza degli errori di stima.

6.2.3 – Conclusioni

In presenza di errori e rumori casuali il sistema è comunque in grado di stimare con discreta precisione la posizione del robot. Nei test presentati si verifica spesso in alcune zone l'allontanamento della stima dal valore reale dello stato simulato, tuttavia si è visto come il Filtro di Kalman Esteso riesca comunque a limitare la divergenza, mantenendo la varianza dell'errore di stima entro valori accettabili. La risposta del sistema in ambiente simulato è quindi soddisfacente.

6.3 – Test del sistema in ambiente reale

Vengono di seguito presentati i test di esecuzione in ambiente reale eseguiti con il sistema finora implementato. Il pattern è un tipico pavimento costituito da piastrelle di forma quadrata regolari, sulle cui dimensioni è inizializzato il modello della misura del filtro di Kalman.

Seguendo le indicazioni fornite dai test precedenti i parametri dell'estrattore di Harris sono inizializzati a valori alti, *Hthr = 1000000* e $\alpha = 0.20$, allo scopo di irrobustire il riconoscimento del corner. Il ritmo di acquisizione della webcam a bordo del robot è pari a 5 FPS, vengono acquisite immagini della risoluzione di

320×240 pixel. E' stato scelto un *frame rate* ridotto per non gravare negativamente sulle prestazioni del sistema, che durante l'utilizzo in ambiente reale si trova a processare una notevole quantità di dati.

I test iniziano sempre con il robot fermo in posizione iniziale, generalmente nell'origine del sistema di riferimento odometrico *O(0;0;0)* (in cui x e y sono espressi in millimetri e θ in radianti), punto che, per ipotesi, deve corrispondere a un corner del pattern. Per ovvi problemi di precisione, non si può garantire a priori che il robot si trovi effettivamente in O senza errori, il posizionamento manuale del robot avviene sistematicamente con piccoli errori sulla posizione e sull'orientamento.

6.3.1 – Test in ambiente reale, robot fermo

Questa serie di test verifica la convergenza del filtro di Kalman mentre il robot Labmate rimane fermo nella posizione iniziale, secondo le ipotesi del capitolo precedente.

6.3.1.1 – Test con robot fermo in posizione iniziale corrispondente all'origine odometrica

Il robot, fermo in posizione iniziale, inquadra un pattern leggermente sporco (del tutto simile a quello impiegato nel cap. 5.2.1.2). Si verifica la presenza di disturbi, noti dai test precedenti, nell'individuazione del corner nei frame iniziali, dovuti all'autocalibrazione della webcam. Tale tipo di rumori è inevitabile, e influenzerà anche i test successivi.

Vengono eseguiti 155 passi di filtraggio di Kalman, durante i quali il robot si localizza e calcola l'errore tra stima e presunta posizione reale.

Vengono mostrati ora gli andamenti del riconoscimento del corner e della stima della posizione del corner:

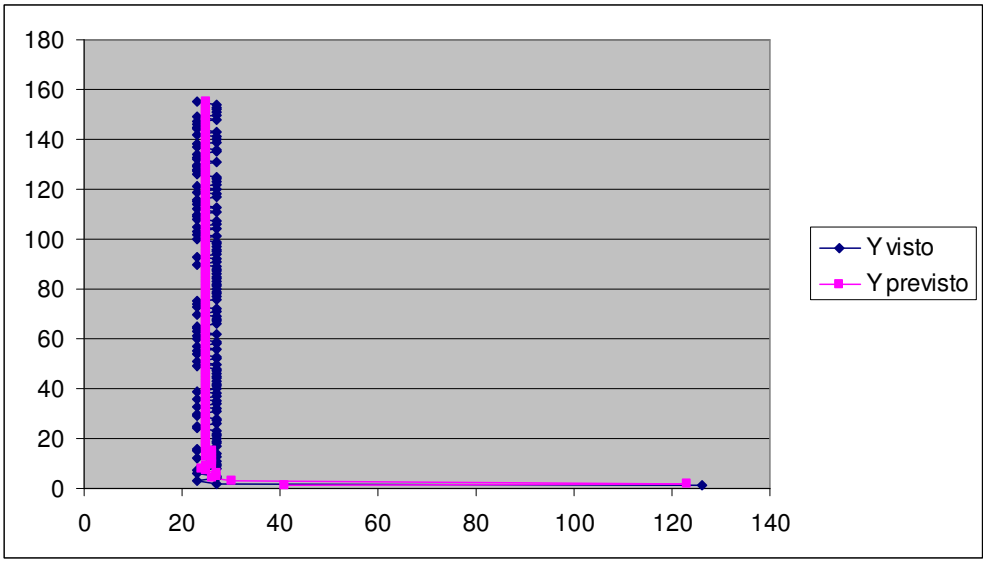


Fig. 39 - Andamento dell'ordinata del corner rilevato (in pixel) e del corner previsto sulla base dell'odometria corrente rispetto al passo temporale (rappresentato sull'asse delle ordinate nel grafico).

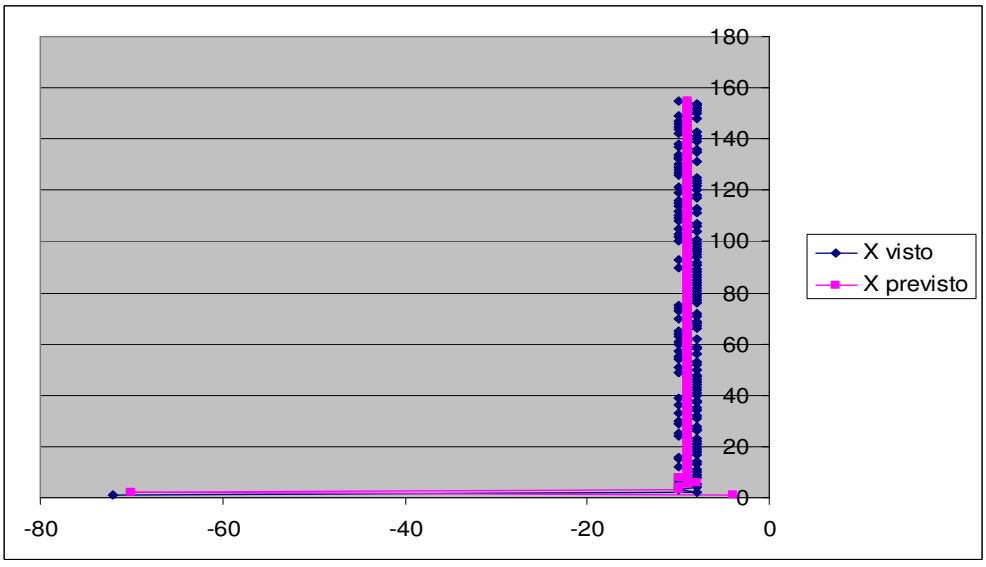


Fig. 40 - Andamento dell'ascissa del corner rilevato (in pixel) e del corner previsto sulla base dell'odometria corrente rispetto al passo temporale (rappresentato sull'asse delle ordinate nel grafico).

Di seguito si presentano gli andamenti dell'errore di stima nel tempo:

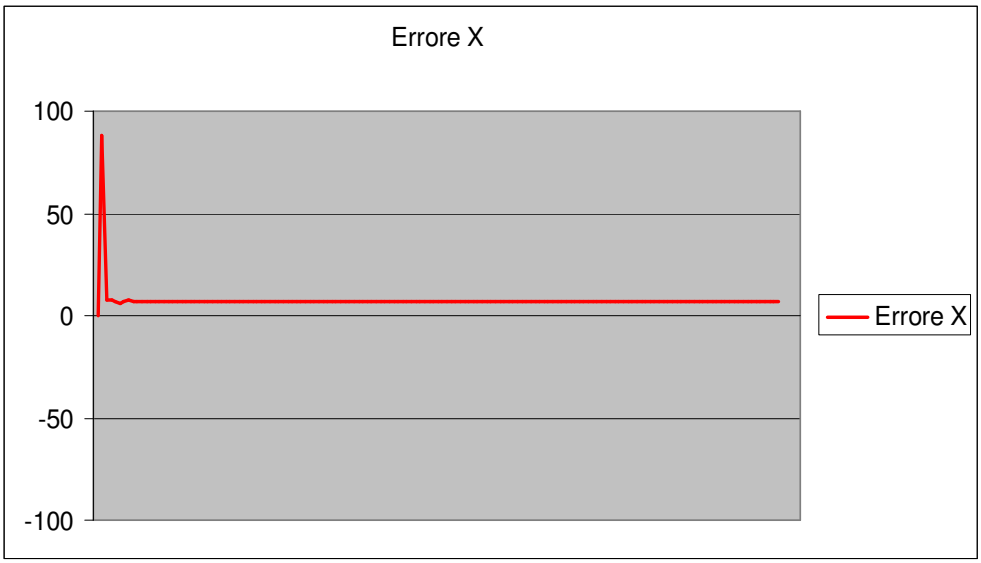


Fig. 41 - Andamento dell'errore nella stima della coordinata X dello stato del sistema (in mm) rispetto al passo temporale (rappresentato sull'asse delle ascisse nel grafico).

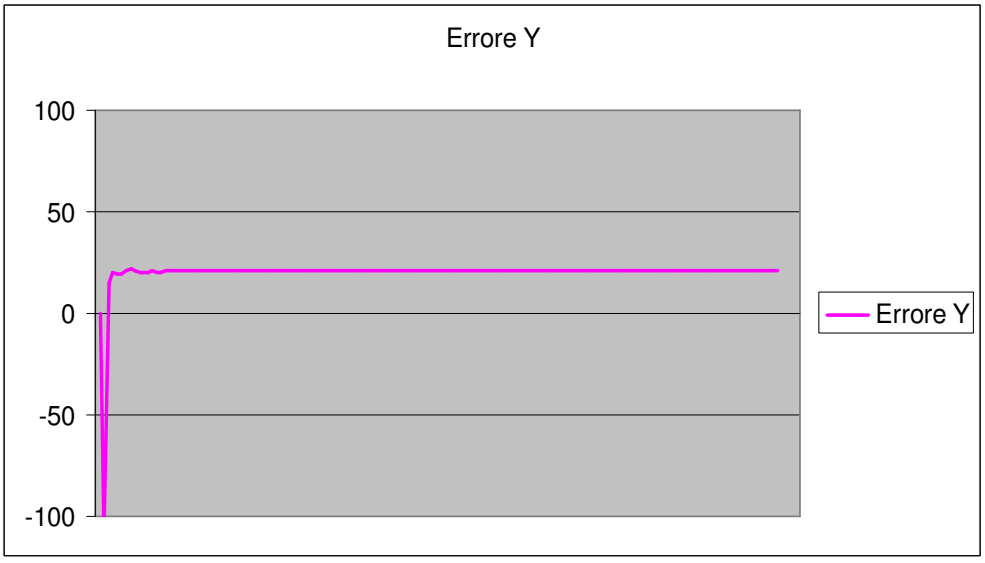


Fig. 42 - Andamento dell'errore nella stima della coordinata Y dello stato del sistema (in mm) rispetto al passo temporale (rappresentato sull'asse delle ascisse nel grafico).

L'errore di stima dopo alcune oscillazioni iniziali converge a un valore fisso, che rappresenta allo stesso tempo la correzione della posizione. La posizione iniziale stimata all'ultimo passo di esecuzione considerato è $\boldsymbol{X}_0=(7;21;0.001)$. Dato il piccolo errore di stima è plausibile ritenere che sia dovuto all'errore di posizionamento manuale. Si ricorda che si sta valutando l'errore di stima rispetto alla presunta posizione iniziale reale (il robot è fermo), che come detto in precedenza potrebbe non essere quella che ci si aspetta, in questo caso O.

6.3.1.2 – Test con robot fermo in posizione iniziale corrispondente all'origine odometrica

In questo test il robot inquadra un pattern pulito. Vengono eseguiti 66 passi di filtraggio di Kalman.

Vengono mostrati di seguito gli andamenti del riconoscimento del corner e della stima della posizione del corner:

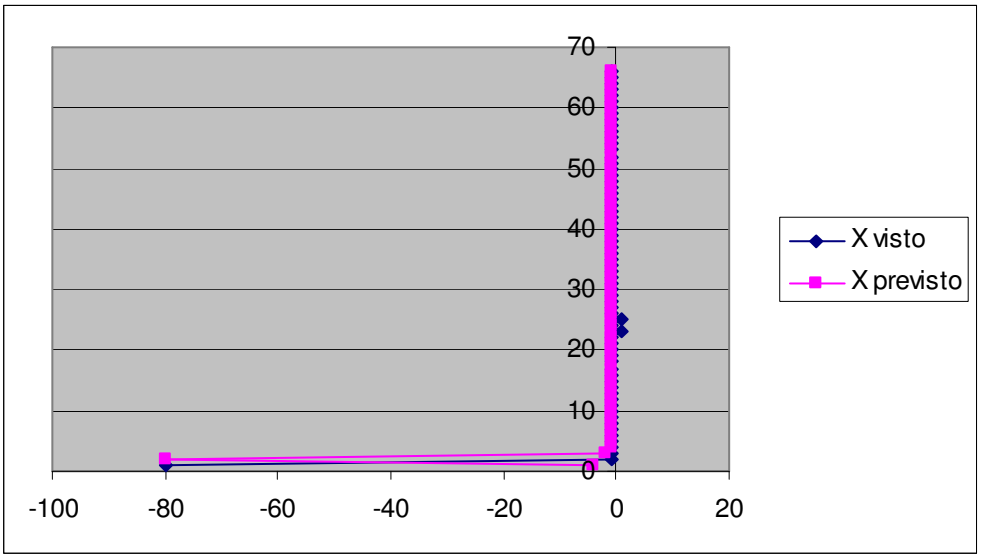


Fig. 43 - Andamento dell'ascissa del corner rilevato (in pixel) e del corner previsto sulla base dell'odometria corrente rispetto al passo temporale (rappresentato sull'asse delle ordinate nel grafico).

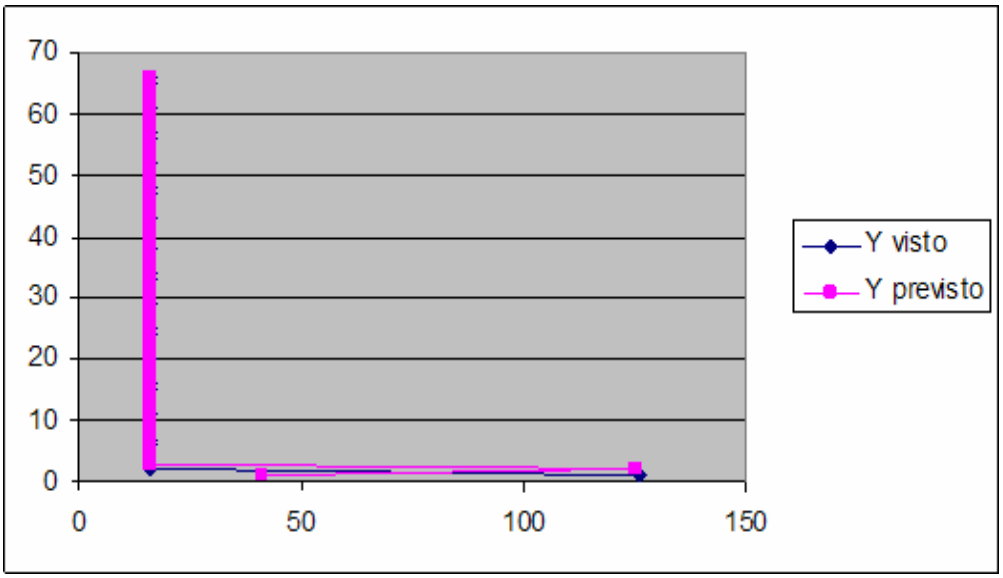


Fig. 44 - Andamento dell'ordinata del corner rilevato (in pixel) e del corner previsto sulla base dell'odometria corrente rispetto al passo temporale (rappresentato sull'asse delle ordinate nel grafico).

Di seguito si presentano gli andamenti dell'errore di stima nel tempo:

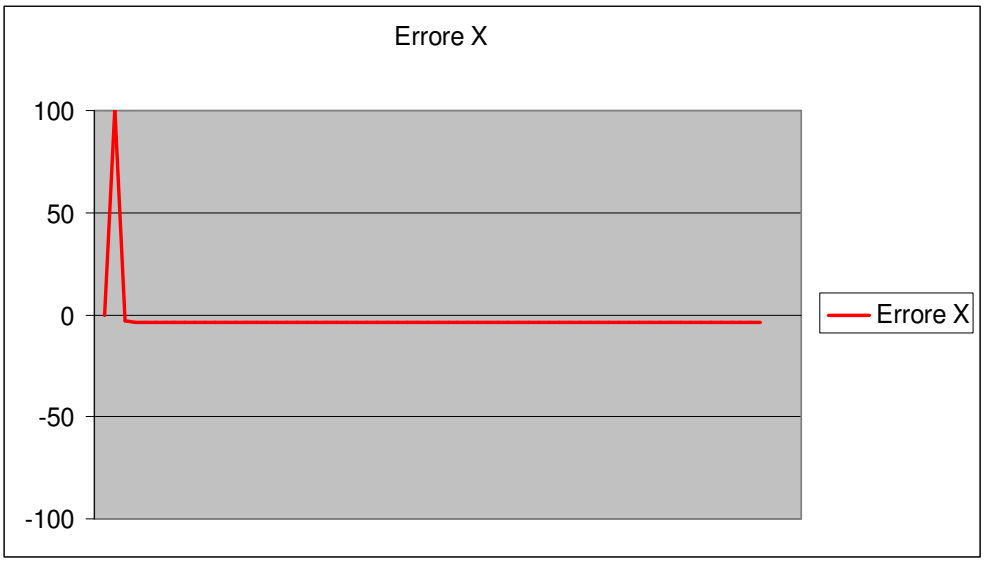


Fig. 45 - Andamento dell'errore nella stima della coordinata X dello stato del sistema (in mm) rispetto al passo temporale (rappresentato sull'asse delle ascisse nel grafico).

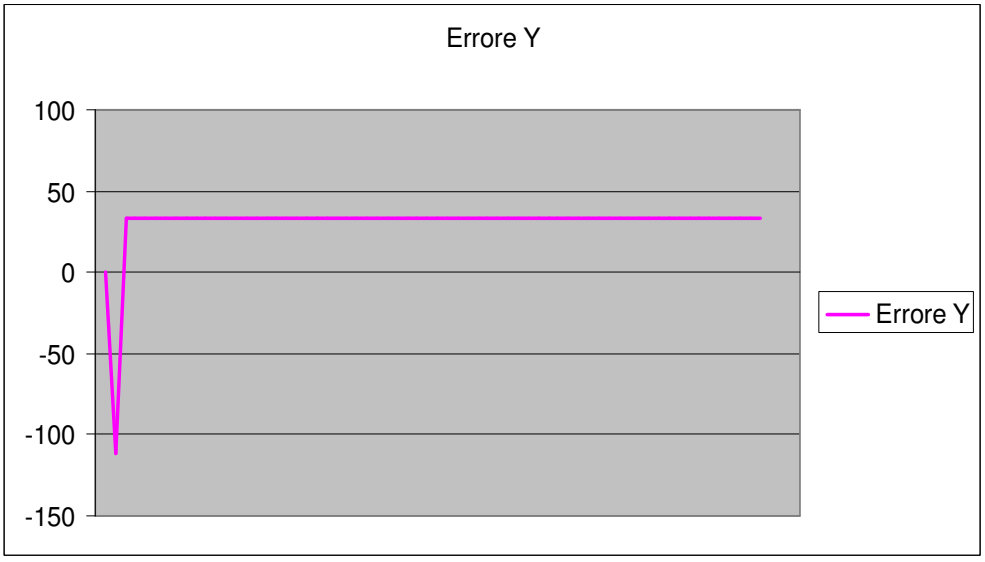


Fig. 46 - Andamento dell'errore nella stima della coordinata Y dello stato del sistema (in mm) rispetto al passo temporale (rappresentato sull'asse delle ascisse nel grafico).

In condizioni di pattern pulito il corner è riconosciuto con grande precisione e la stima converge velocemente.

La posizione iniziale stimata all'ultimo passo di esecuzione considerato è $\boldsymbol{X}_0=(4;33;0)$. Anche in questo caso l'errore di posizionamento rilevato è più che accettabile.

6.3.1.3 – Test con robot fermo in posizione iniziale leggermente diversa dall'origine odometrica.

In questo test il robot inquadra un pattern pulito.

La posizione iniziale è volontariamente spostata di circa 5 cm lungo entrambi gli assi, ovvero l'origine odometrica si trova in realtà nella posizione $O'(50,50,0)$. Si vuole in tal modo verificare le capacità di correzione del filtro con una posizione iniziale leggermente spostata in entrambe le direzioni.

Vengono eseguiti 52 passi di filtraggio di Kalman.

Di seguito sono mostrati gli andamenti del riconoscimento del corner e dell'errore di stima della posizione del corner:

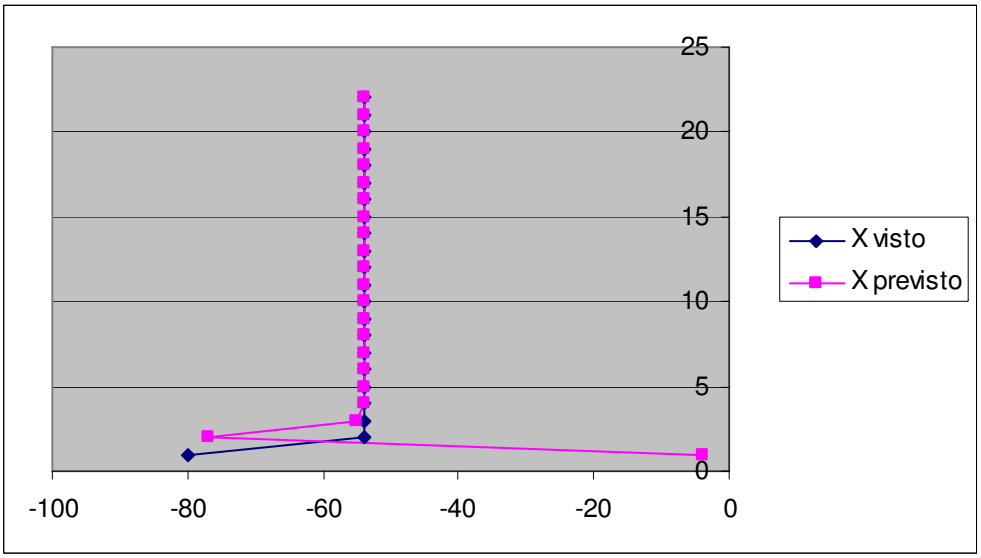


Fig. 47 - Andamento dell'ascissa del corner rilevato (in pixel) e del corner previsto sulla base dell'odometria corrente rispetto al passo temporale (rappresentato sull'asse delle ordinate nel grafico).

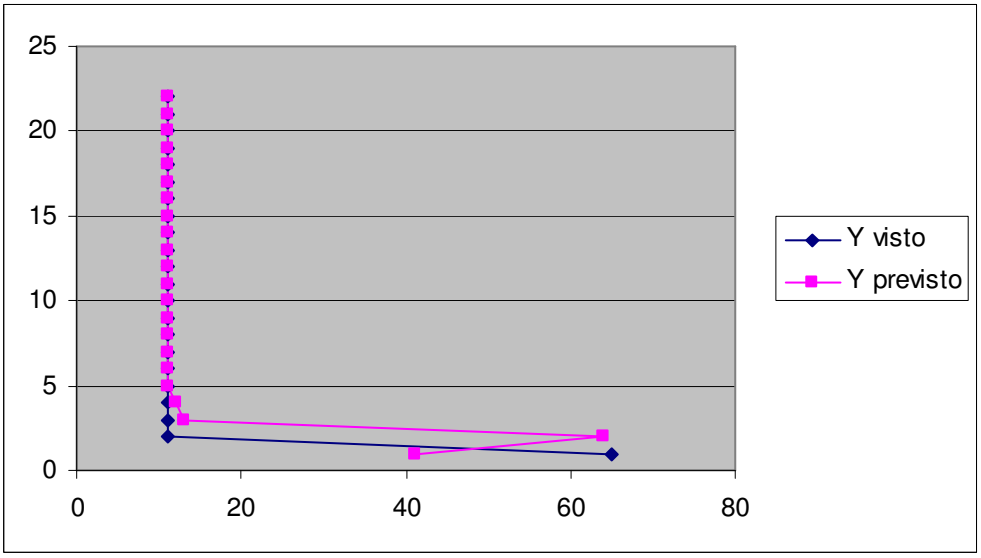


Fig. 48 - Andamento dell'ordinata del corner rilevato (in pixel) e del corner previsto sulla base dell'odometria corrente rispetto al passo temporale (rappresentato sull'asse delle ordinate nel grafico)

.

Di seguito si presentano gli andamenti dell'errore di stima nel tempo:

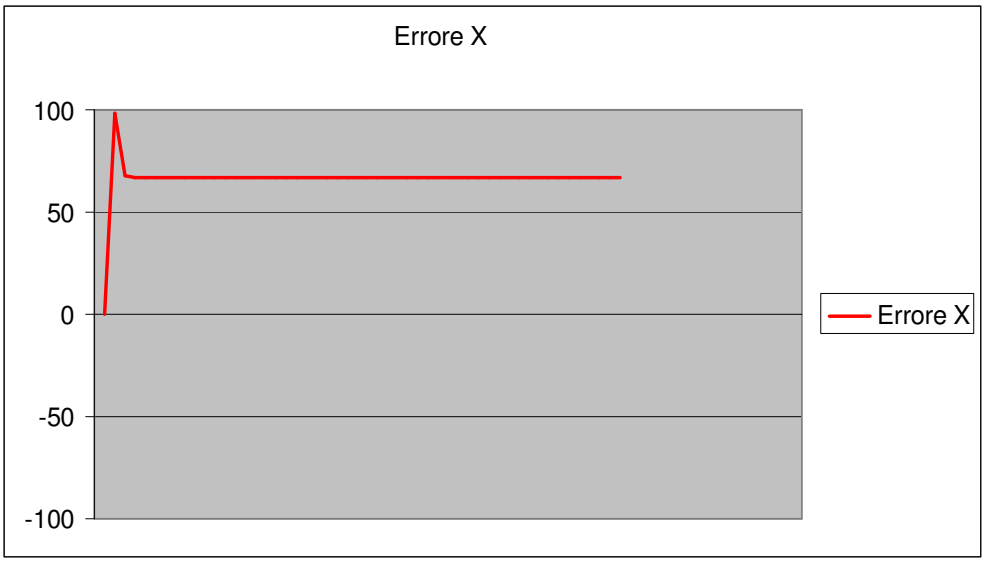


Fig. 49 - Andamento dell'errore nella stima della coordinata X dello stato del sistema (in mm) rispetto al passo temporale (rappresentato sull'asse delle ascisse nel grafico).

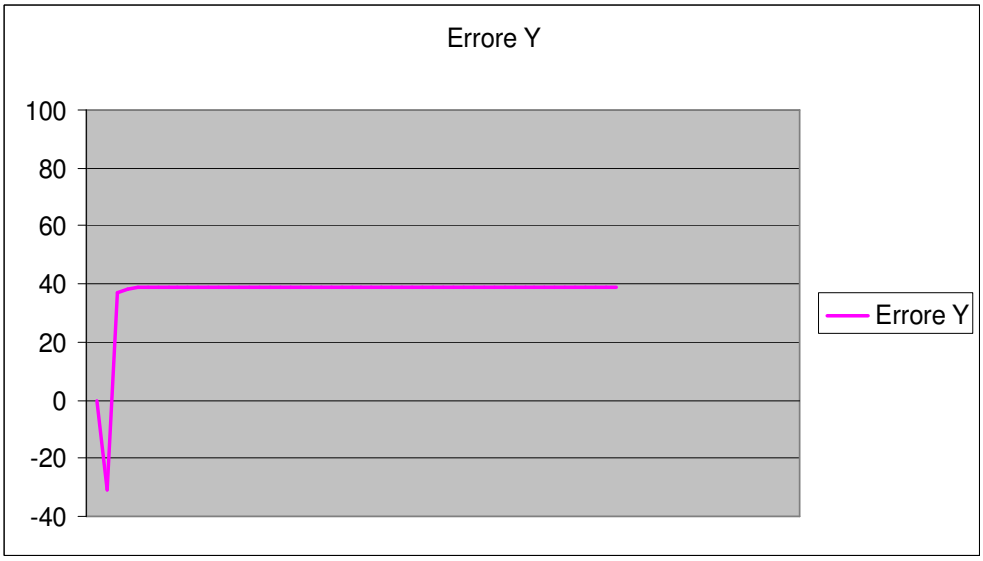


Fig. 50 - Andamento dell'errore nella stima della coordinata Y dello stato del sistema (in mm) rispetto al passo temporale (rappresentato sull'asse delle ascisse nel grafico).

Il filtro ha corretto la stima identificando la posizione reale in $\boldsymbol{X}_0=(67;39;0.001)$. L'errore è quindi di 1,7 cm lungo *x* e di 1,1 cm lungo *y*, un risultato soddisfacente considerando l'ulteriore imprecisione dovuta al posizionamento manuale del robot.

6.3.1.4 – Test con robot fermo in posizione iniziale leggermente diversa dall'origine odometrica

In questo test il robot inquadra un pattern pulito.

La posizione iniziale è volontariamente spostata di circa 5 cm lungo il solo asse *Y*, ovvero l'origine odometrica si trova in realtà nella posizione *O'(0,50,0)*. Si vuole in tal modo verificare le capacità di correzione del filtro con una posizione iniziale spostata lungo una sola direzione.

Vengono eseguiti 152 passi di filtraggio di Kalman.

Vengono mostrati di seguito gli andamenti del riconoscimento del corner e della stima della posizione del corner:

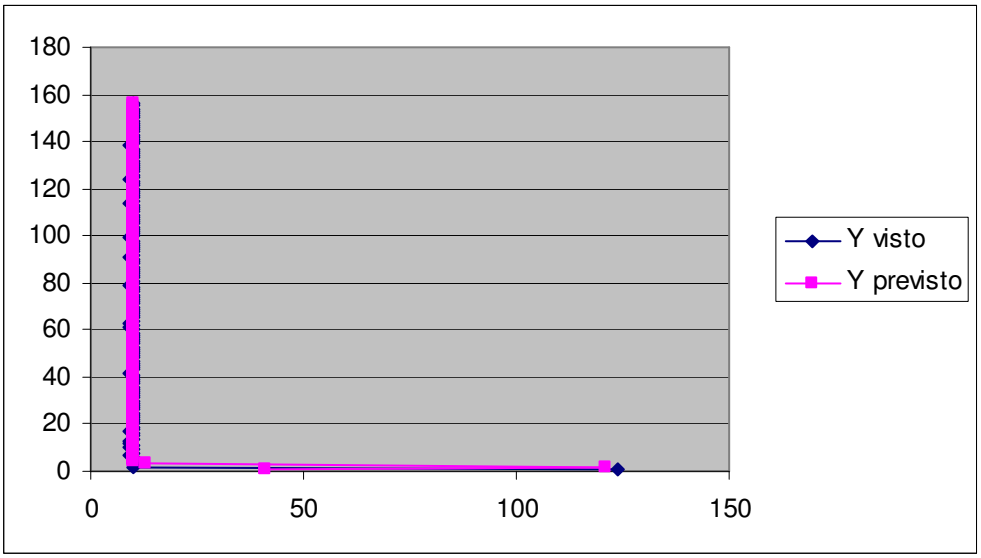


Fig. 50 - Andamento dell'ordinata del corner rilevato (in pixel) e del corner previsto sulla base dell'odometria corrente rispetto al passo temporale (rappresentato sull'asse delle ordinate nel grafico).

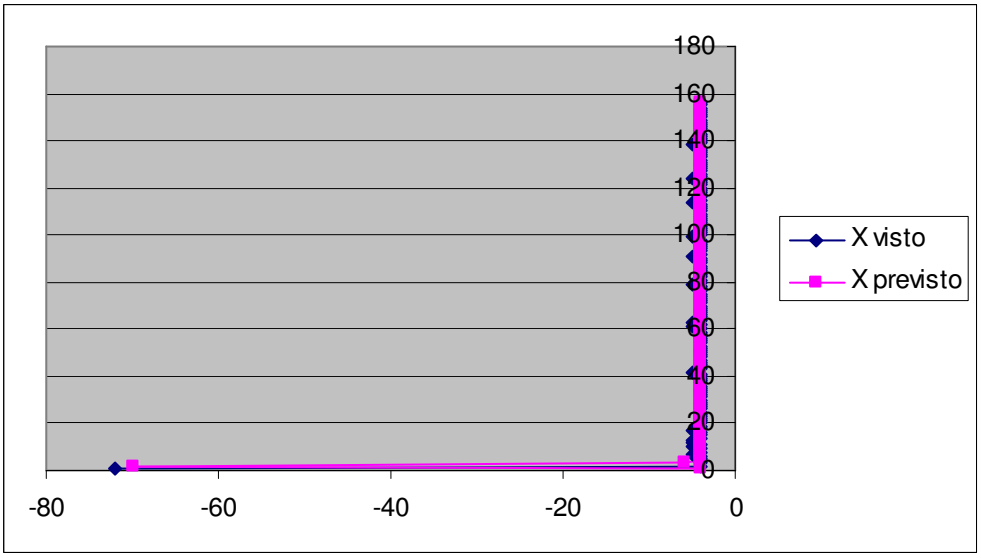


Fig. 51 - Andamento dell'ascissa del corner rilevato (in pixel) e del corner previsto sulla base dell'odometria corrente rispetto al passo temporale (rappresentato sull'asse delle ordinate nel grafico).

Di seguito si presentano gli andamenti dell'errore di stima nel tempo:

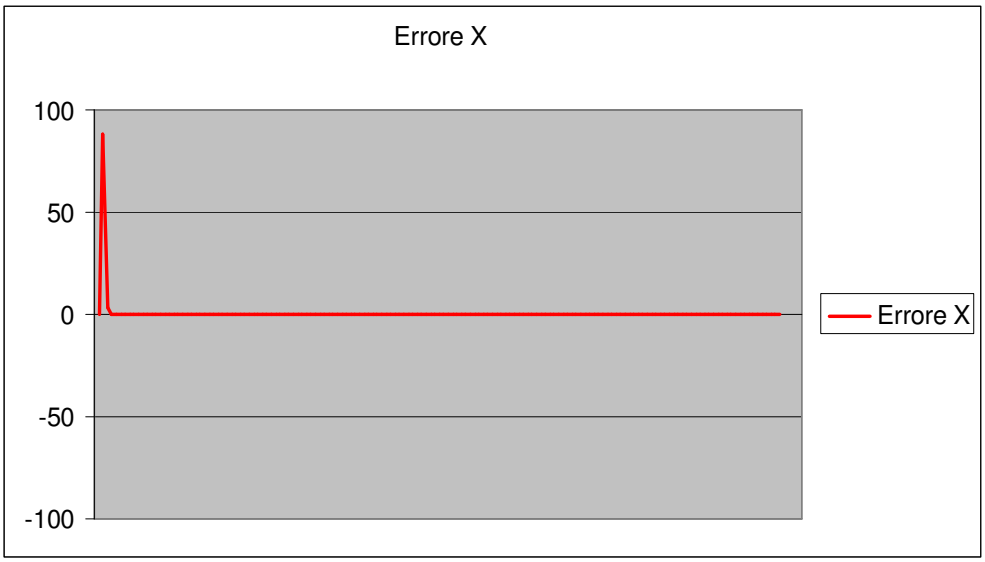


Fig. 52 - Andamento dell'errore nella stima della coordinata X dello stato del sistema (in mm) rispetto al passo temporale (rappresentato sull'asse delle ascisse nel grafico).

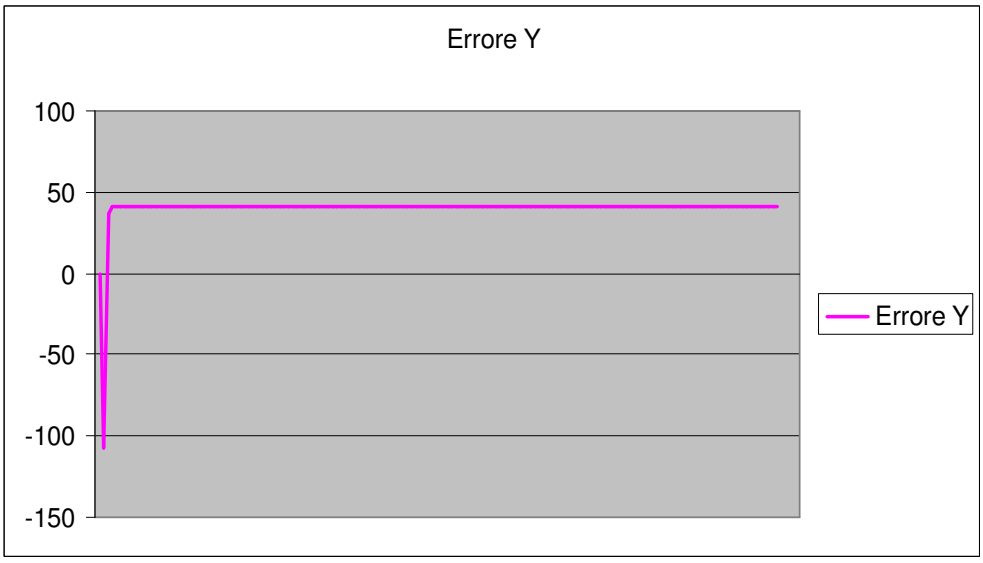


Fig. 53 - Andamento dell'errore nella stima della coordinata Y dello stato del sistema (in mm) rispetto al passo temporale (rappresentato sull'asse delle ascisse nel grafico).

Il filtro ha corretto la stima identificando la posizione reale in $\boldsymbol{X}_0=(0;41;0)$. L'errore è quindi di soli 0,9 cm lungo y, un risultato più che soddisfacente.

6.3.1.5 – Test con robot fermo in posizione iniziale e simulazione di rumore e disturbi.

Il robot inquadra un pattern pulito. Questo test è stato eseguito come prova del sistema in condizioni di disturbi e rumori significativi. Dopo una fase iniziale di assestamento della stima, il pattern viene modificato e ostruito dinamicamente in maniera manuale. La fase di disturbo avviene dai passi 66 al passo 230 di filtraggio. Nei passi successivi, dal 231 al 265, il pattern torna ad essere visibile nelle sue condizioni reali.

Vengono eseguiti 265 passi di filtraggio di Kalman.

Vengono mostrati nella pagina seguente gli andamenti del riconoscimento del corner e della stima della posizione del corner:

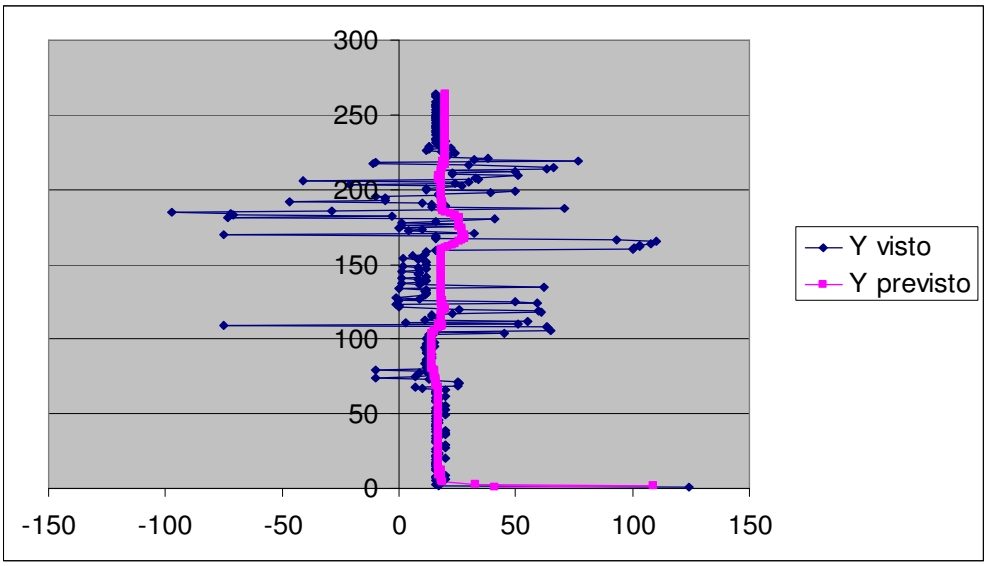


Fig. 54 - Andamento dell'ordinata del corner rilevato (in pixel) e del corner previsto sulla base dell'odometria corrente rispetto al passo temporale (rappresentato sull'asse delle ordinate nel grafico).

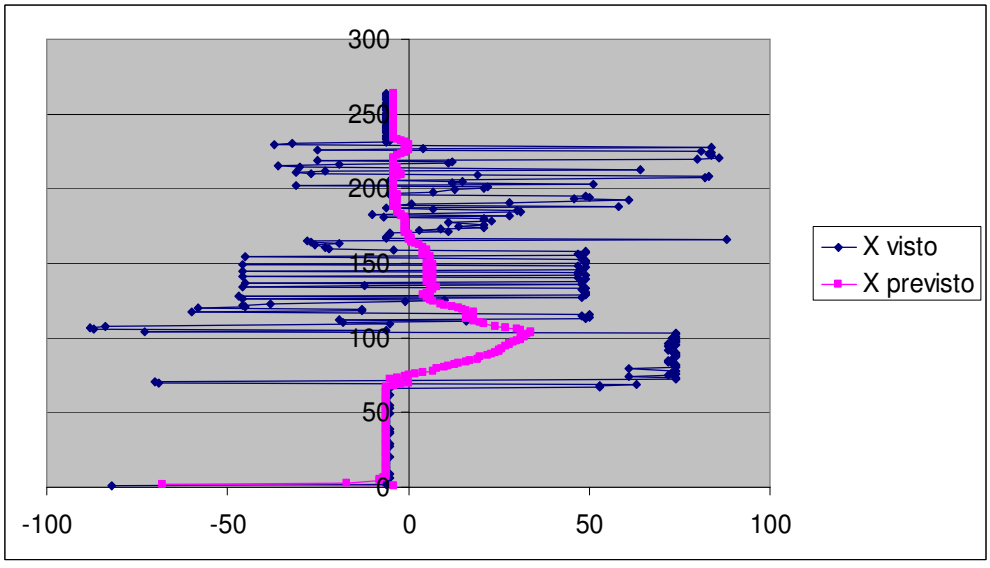


Fig. 55 - Andamento dell'ascissa del corner rilevato (in pixel) e del corner previsto sulla base dell'odometria corrente rispetto al passo temporale (rappresentato sull'asse delle ordinate nel grafico).

Di seguito si presentano gli andamenti dell'errore di stima nel tempo:

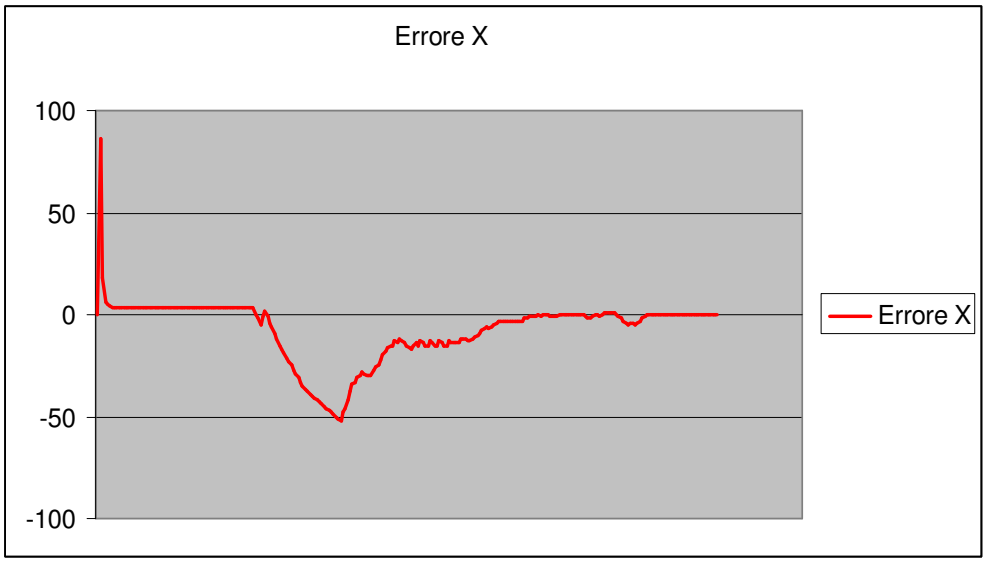


Fig. 56 - Andamento dell'errore nella stima della coordinata X dello stato del sistema (in mm) rispetto al passo temporale (rappresentato sull'asse delle ascisse nel grafico).

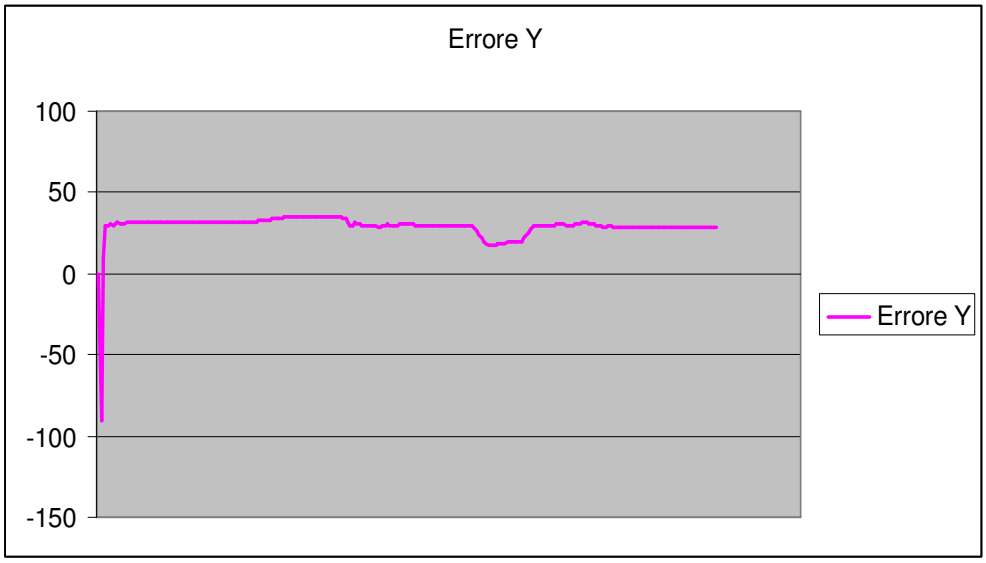


Fig. 57 - Andamento dell'errore nella stima della coordinata Y dello stato del sistema (in mm) rispetto al passo temporale (rappresentato sull'asse delle ascisse nel grafico).

Al termine dei primi 65 passi la stima identifica la posizione del robot in $\boldsymbol{X}_0=(4;33;0.004)$. Un risultato in linea con i risultati dei test precedenti, e decisamente soddisfacente.

Durante la successiva fase di disturbo la stima è caratterizzata da una varianza elevata in tutte e tre le componenti dello stato:

varianza		*k=66 to 230*
X	Y	θ
217,612195	19,876792	2,903776

In particolare è la coordinata *x* a subire le variazioni più elevate.

Nonostante i disturbi introdotti l'analisi dei grafici mostra l'errore convergere nuovamente a un valore stabile, la stima si stabilizza a partire dal passo 240, e al passo finale identifica la posizione in $\boldsymbol{X}_0=(3;32;0)$. La differenza tra le stime *pre* e *post* fase di disturbo è quindi minima. Se ne conclude che il Filtro di Kalman ha contribuito positivamente alla localizzazione, evitando la divergenza e mantenendo la stima entro valori accettabili.

6.3.1.6 – Conclusioni

Il sistema analizzato in ambiente reale con il robot fermo in posizione iniziale si è rivelato dotato di buona precisione e robustezza ai rumori. La mancanza di movimento annulla di fatto possibili errori odometrici, e quando il pattern si presenta pulito il robot è in grado di localizzarsi con sicurezza.

Il test di robustezza ai disturbi è stato soddisfacente, e ha mostrato la convergenza del sistema nel suo complesso anche in condizioni di lavoro pessime. Tuttavia bisogna tener conto che i buoni risultati sono ottenuti comunque in condizioni di assenza di

disturbi ed errori significativi sulla componente di orientazione, che (come visto nel cap. 4.5) causa la non linearità nel sistema, e può rendersi responsabile di errori nella linearizzazione della stima.

6.3.2 – Test in ambiente reale, robot in movimento

La serie di prove che vengono ora presentate rappresentano il risultato del comportamento del robot in movimento in ambiente reale. Dopo aver verificato il soddisfacente funzionamento del sistema a robot fermo, l'analisi del sistema è proseguita pianificando dei percorsi da far eseguire al robot, in modo da valutare l'operato del sistema di localizzazione nelle condizioni di lavoro complete.

Sono stati scelti due tipi di percorsi significativi, analoghi a quelli provati in ambiente simulato. Il primo prevede che il robot esegua ciclicamente una traiettoria quadrata. Il secondo prevede invece che il robot si muova in linea retta ciclicamente tra due posizioni diverse. Nelle figure seguenti vengono mostrate le schematizzazioni dei percorsi seguiti.

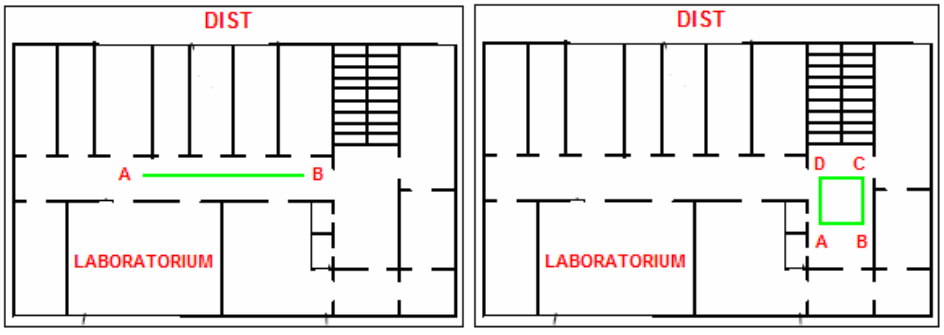


Fig.58 - Riproduzione schematizzata dei percorsi seguiti dal robot nei test. Il percorso dell'immagine di sinistra, di circa 4 metri, prevede che il robot si muova in linea retta dal punto A al punto B e viceversa. Il percorso di destra invece è relativo a un quadrato ABCD regolare di circa 180 cm di lato.

Allo scopo di mostrare l'incapacità della sola ricostruzione odometrica di localizzare efficacemente il robot, i percorsi sono stati seguiti da LabMate anche senza l'utilizzo del filtro di Kalman e del sistema di visione, e ne verranno presentati i risultati.

Le condizioni del pattern in entrambi i percorsi scelti per il movimento risultavano discrete: l'illuminazione era per lo più uniforme, le piastrelle del pattern presentavano saltuariamente qualche irregolarità e alcune zone di sporcizia.

Fig.59 - Questa foto mostra un particolare del pavimento del corridoio utilizzato durante i test con il robot in movimento. La superficie appare abbastanza pulita e uniforme, con le fughe ben visibili..

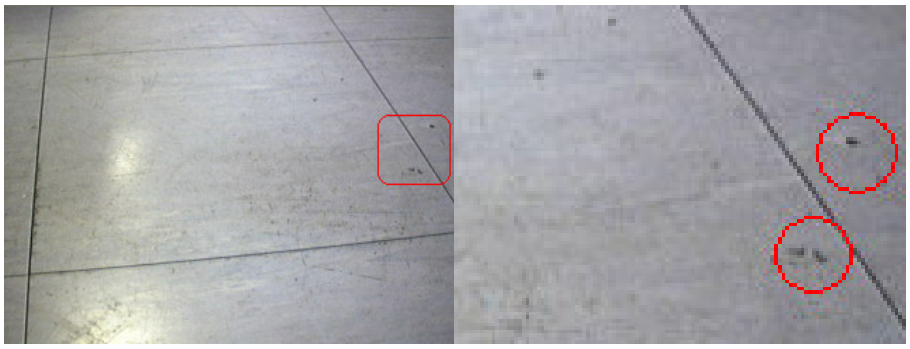


Fig. 60 - Questa foto mostra, in una zona di pavimento di test a prima vista uniforme, la presenza di alcune piccole impurità in grado potenzialmente di ingannare l'estrattore, portandolo all'individuazione di corner diversi dall'incrocio tra le fughe delle piastrelle del pavimento.

L'inevitabile presenza di saltuarie zone di sporco, capaci di portare l'estrattore di corner all'individuazione di corner differenti dall'incrocio delle fughe delle piastrelle del pattern, ha reso necessaria l'introduzione di un controllo nel Filtro di Kalman Esteso allo scopo di impedire la correzione qualora si ritenga che il corner fornito dal modello della misura non sia un incrocio. Ciò è stato implementato analizzando la distanza tra la posizione (nel piano immagine) del corner previsto e del corner rilevato: se la distanza è maggiore di una quantità prefissata di pixel, la correzione viene scartata, per evitare il rischio di correggere la stima con un'informazione quasi sicuramente errata. In fase di *testing* è stato quindi scelto di scartare corner la cui distanza dalla posizione prevista sia maggiore di 30 pixel (quantità significativa tenendo presente che viene analizzata un'area pari a 260×180 pixel per ogni immagine acquisita).

6.3.2.1 – Test percorso quadrato

Il percorso seguito dal robot in questo test è un quadrato regolare di lato pari a 180 centimetri, corrispondenti a circa la lunghezza di 3 piastrelle del pavimento. Il robot prosegue nel percorso senza alcuna interazione manuale da parte dell'utente, calcolando le traiettorie basandosi sulla posizione indicata dalla localizzazione.

Il percorso prevede che il robot passi in 4 punti significativi A,B,C,D, corrispondenti ai vertici del quadrato.

La prima prova è stata eseguita servendosi della sola ricostruzione odometrica per localizzare il robot. Sono stati disposti dei marker artificiali sul percorso, numerati allo scopo di evidenziare i diversi punti di passaggio nei vertici A,B,C,D ad ogni giro del robot. Dopo 10 giri si è reso necessario interrompere manualmente il movimento per impedire al robot di entrare in collisione con il muro perimetrale del corridoio, essendosi il robot allontanato di alcuni metri dal vertice di passaggio prefissato. La quantità di errori odometrici accumulati ha quindi impedito al robot di localizzarsi correttamente. Vengono mostrati nelle figure seguenti i risultati ottenuti:

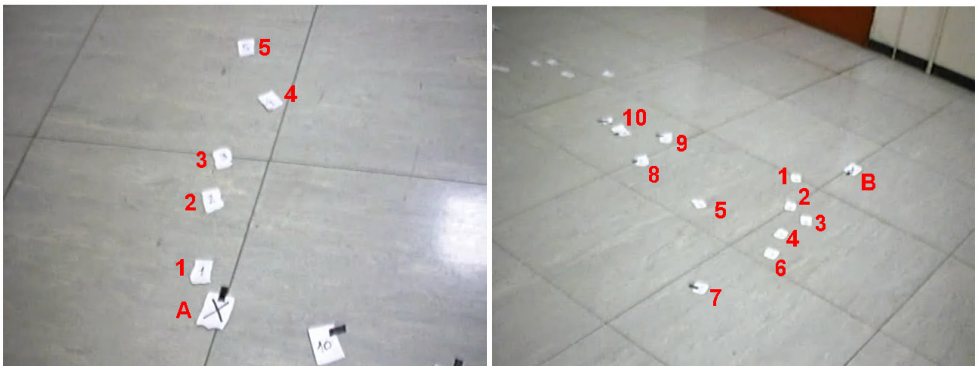


Fig. 61 – Particolari dei marker artificiali distribuiti sul pavimento. Nella foto di sinistra si può vedere l'allontanarsi del robot al passaggio nel punto A, già nel giro numero 5 la posizione è distante oltre 60 centimetri. Nella foto di destra si possono invece vedere dall'alto i diversi passaggi del robot nel punto B.

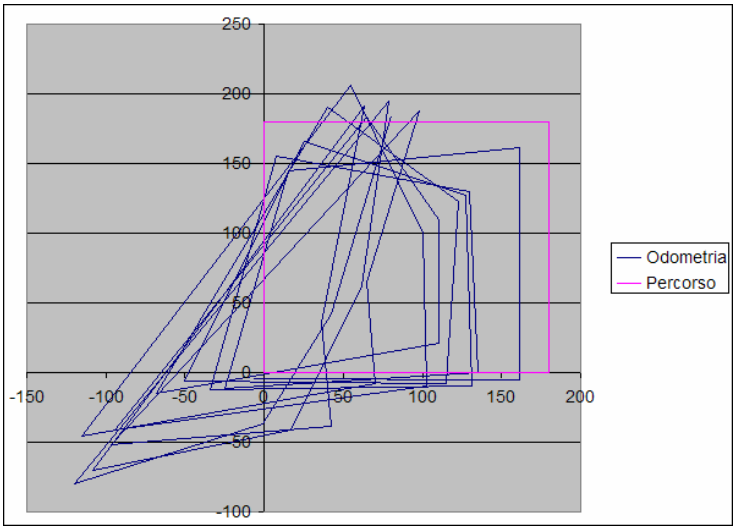


Fig. 62 – Questo grafico mostra l'andamento della posizione del robot nel quadrato di prova, utilizzando per la localizzazione la sola ricostruzione odometrica.

Si è di seguito proceduto a far compiere il percorso di prova al robot attivando tutto il sistema di navigazione realizzato:

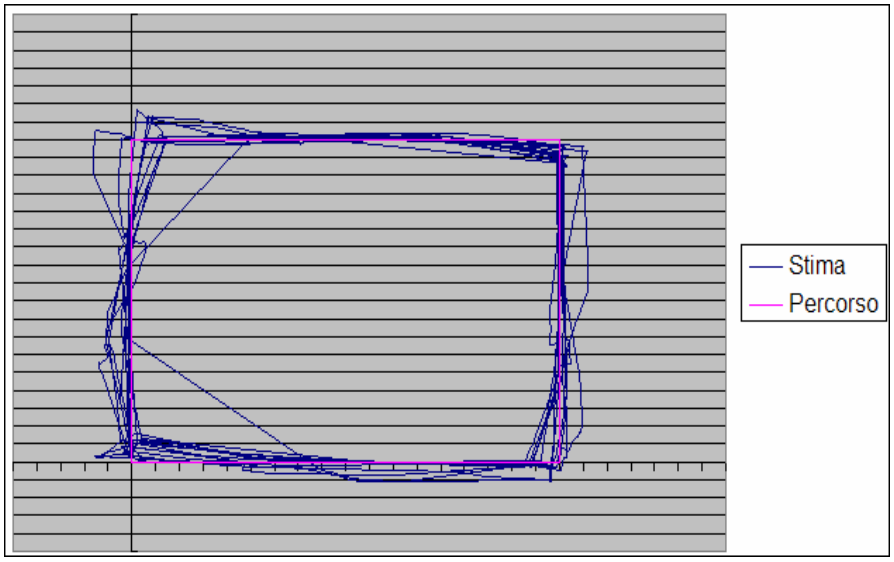


Fig. 63 – Andamento della stima della posizione del robot. La linea chiara rappresenta il percorso prefissato da seguire.

Il grafico precedente mostra l'andamento della stima della posizione del robot durante il test.

L'andamento della stima è nel complesso quella che ci si aspettava come risposta del sistema. Il robot ha percorso in totale 20 volte il quadrato, seguendo con buona approssimazione la traiettoria prefissata. Non è stato possibile utilizzare i marker artificiali come nel caso odometrico, in quanto avrebbero alterato il pattern impedendo la corretta individuazione dei corner. Sono comunque state eseguite alcune misurazioni manuali in maniera abbastanza precisa, nelle quali l'errore di stima massimo registrato nei vertici del quadrato è stato pari a circa 30 centimetri, un risultato più che soddisfacente.

6.3.2.2 – Test percorso rettilineo

Il percorso seguito dal robot in questo test è rettilineo e lungo 4 metri, corrispondenti alla lunghezza di circa 6,5 piastrelle del pavimento. Il robot si muove sul percorso tra i due vertici A e B, senza alcuna interazione manuale da parte dell'utente, calcolando le traiettorie basandosi sulla posizione indicata dalla localizzazione.

La prima prova è stata eseguita servendosi della sola ricostruzione odometrica per localizzare il robot. Sono stati disposti dei marker artificiali sul percorso, numerati allo scopo di evidenziare i diversi punti di passaggio relativi ai vertici A e B ad ogni giro del robot. Dopo 6 giri si è reso necessario interrompere manualmente il movimento per impedire al robot di entrare in collisione con il muro perimetrale del corridoio. La quantità di errori odometrici accumulati ha quindi impedito al robot di localizzarsi correttamente. Vengono mostrati nelle figure seguenti i risultati ottenuti:

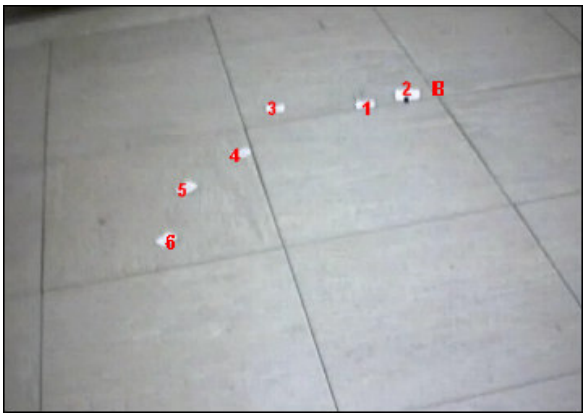


Fig. 64 – Particolare della posizione dei marker relativi al movimento del robot con sola ricostruzione odometrica.

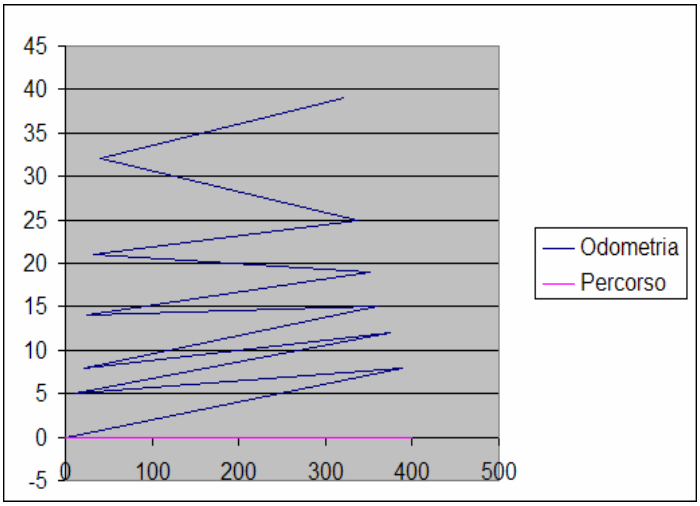


Fig. 65 – Questo grafico mostra l'andamento della posizione del robot nel percorso di prova, utilizzando per la localizzazione la sola ricostruzione odometrica.

Si è di seguito proceduto a far compiere il percorso di prova al robot attivando tutto il sistema di navigazione realizzato. Il grafico

seguente mostra l'andamento della stima della posizione del robot durante il test:

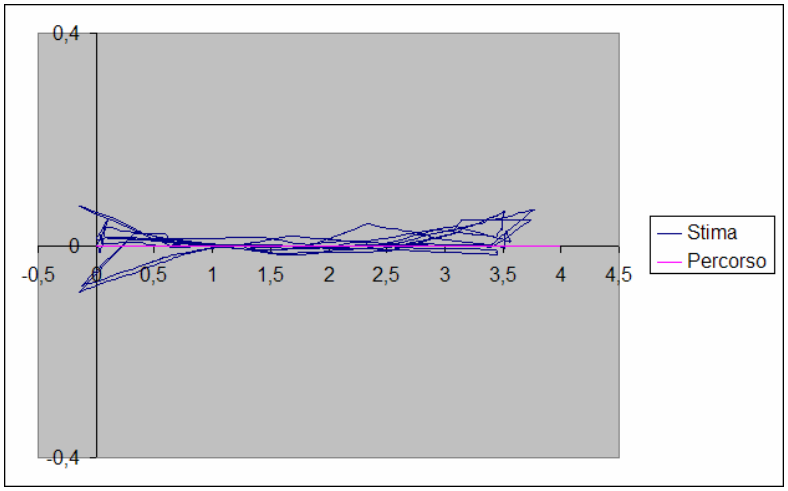


Fig. 66 – Andamento della stima della posizione del robot. La linea chiara rappresenta il percorso prefissato da seguire.

L'andamento della stima è nel complesso quella che ci si aspettava come risposta del sistema. Il robot ha eseguito in totale 10 volte il percorso, seguendo con buona approssimazione la traiettoria prefissata. Anche in questo caso naturalmente non è stato possibile utilizzare i marker artificiali, in quanto avrebbero alterato il pattern impedendo la corretta individuazione dei corner.

Come nel caso del percorso quadrato, anche questa volta l'errore di stima massimo registrato in un vertice del percorso è stato pari a circa 30 centimetri, corrispondente a circa mezza piastrella.

6.4 – Conclusioni e sviluppi futuri

Nel complesso il sistema ha mostrato delle buone capacità di localizzazione. Durante il movimento in ambiente reale è stato

mostrato che la ricostruzione odometrica non è in grado di localizzare da sola il robot.

Durante il test con robot fermo è stato mostrato come il sistema di localizzazione sia in grado di stimare correttamente la posizione del robot anche dopo una lunga serie di disturbi e rumori nell'acquisizione delle features. Questa prestazione positiva non sembra essere così marcata durante il movimento. Questo fatto appare essere dovuto alle ridotte capacità di inquadratura della webcam di bordo. Durante il movimento la webcam è in grado di inquadrare una zona di pavimento di soli 260×180 pixel, pari a circa 34×24 centimetri in base alla posizione della webcam rispetto al pavimento. Ciò significa che la webcam è in grado di inquadrare un'area di pavimento pari a meno della metà dell'area di una piastrella. Di conseguenza, durante la navigazione sono molte le zone in cui non si è in grado di inquadrare corner, e queste superano nettamente le zone in cui i corner sono visibili. In tali zone la stima della posizione viene affidata interamente alla ricostruzione odometrica (che fornisce la stima a priori nel Filtro di Kalman), influendo negativamente sulla stima stessa data l'alta quantità di errori che la affliggono. Il filtraggio avviene in corrispondenza di ogni *frame* acquisito, e mediamente si è rilevato che per ogni corner visibile vengono acquisiti solo 7 *frame* durante la navigazione, mentre tra una zona in cui è visibile un corner e l'altra vengono acquisiti invece circa 30 *frame*. Il contributo dell'odometria è quindi durante il movimento molto più elevato rispetto a quello del filtraggio. Non si dimentichi inoltre che talvolta è possibile che impurità sul pavimento ingannino il riconoscitore in zone in cui è visibile anche il corner vero, causando il rigetto della correzione operata dal Filtro di Kalman.

Le considerazioni precedenti pongono in pratica le basi per lo sviluppo futuro di questo progetto. La scelta dell'incrocio tra le fughe delle piastrelle come feature di lavoro si è confermata una buona idea, in quanto l'incrocio è facilmente riconoscibile, non risulta affetto da problemi di occlusione, e il riconoscimento non è strettamente legato alle condizioni di illuminazione. Il problema dei falsi riconoscimenti può essere risolto rafforzando l'estrattore di Harris, inserendo ulteriori controlli e tecniche che consentano di distinguere tra un incrocio tra le fughe e le zone di sporco.

Riguardo al frequente ricorso alla ricostruzione odometrica a causa della ridotta area inquadrata dalla webcam, fonte di un accumulo considerevole di errori nella stima della posizione, si pone la necessità di migliorare il sistema di visione. La webcam si è rivelata uno strumento utile, semplice e poco costoso, ma ha dei limiti in definitiva non trascurabili. Le possibilità di *upgrade* del sistema possono prevedere quindi l'introduzione di uno strumento di acquisizione video in grado di inquadrare un'area più grande di pavimento, oppure l'introduzione di ulteriori webcam, in modo da poter cercare più landmark contemporaneamente durante la navigazione. Avere a disposizione frequenti informazioni sulle posizioni delle feature si è rivelato essere di fondamentale importanza per migliorare l'accuratezza delle stime, e localizzare con sempre più precisione il robot.

Concludendo, l'obiettivo di autolocalizzare un robot mediante il riconoscimento degli incroci delle fughe del pavimento di navigazione è stato raggiunto. I risultati sperimentali sono più che incoraggianti, e sicuramente il superamento delle limitazioni imposte dalla webcam e dal riconoscimento di corner in senso lato non potrà che portare migliorie a questo sistema, tali da fornire performances ancora più soddisfacenti di quelle attuali.

Appendici

Appendice A – Ambiente di sviluppo ETHNOS

ETHNOS (Expert Tribe in a Hybrid Network Operating System) è un ambiente di sviluppo in grado di gestire la comunicazione e lo sviluppo di sistemi real time [SGO].

Programmato in c++ e sviluppato in ambiente Linux, ETHNOS incorpora funzionalità di schedulazione di processi e di distribuzione del carico computazionale su diversi calcolatori.

ETHNOS rappresenta l'implementazione del modello cognitivo HEIR [PIA], un modello concettuale per l'integrazione dei diversi livelli di un sistema robotico.

Il sistema è composto essenzialmente da due unità:

- *Esperti*: sono parti di codice predisposte per eseguire compiti specifici. Gli esperti vengono eseguiti ripetutamente, si distinguono in *periodici* se vengono eseguiti a scadenza temporale prefissata, *sporadici* se vengono eseguiti solo al verificarsi di eventi particolari, *di background* se vengono eseguiti solo mentre il processore non è impegnato nell'esecuzione di altri esperti.

- *Kernel*: è un programma di interfaccia con lo schedulatore del sistema operativo, attraverso il quale gestisce la schedulazione e la comunicazione tra i processi. Attraverso l'analisi del tempo di esecuzione degli esperti presenti nel sistema il kernel effettua un'analisi di schedulabilità, in base a cui decide se eseguire l'insieme di esperti.

Esperti e kernel comunicano tra loro attraverso lo scambio di *messaggi*, ovvero strutture dati contenenti informazioni. Lo scambio avviene tramite zone di memoria condivise, risparmiando occupazione di memoria e proteggendo i dati da modifiche. Il kernel diffonde infine l'informazione nel sistema realizzando una sorta di *broadcasting*. Istanziando più kernel su diversi elaboratori è possibile inoltre realizzare sistemi distribuiti.

Appendice B – Webcam e spazio di colore YUV

Nella realizzazione del sistema descritto nel capitolo 5 si è reso necessaria l'implementazione di un esperto ETHNOS destinato all'acquisizione di immagini da webcam. La realizzazione dell'esperto, che è stato chiamato Webcam Expert, ha portato alla luce alcuni problemi di trattamento dei dati acquisiti. Infatti tipicamente le webcam sono in grado di acquisire immagini solamente nel formato nativo YUV, in cui il valore Y rappresenta la luminanza (luminosità del pixel), i valori U (componente differenza del blu) e V (componente differenza del rosso) rappresentano invece la crominanza, ovvero la differenza di valore tra le quantità di colore:

$$Y = 0{,}2990\,R + 0{,}5870\,G + 0{,}1140\,B$$
$$U = 0{,}5635\,(B - Y) + 128 = -\,0{,}1684\,R - 0{,}3316\,G + 0{,}5\,B + 128$$
$$V = 0{,}7133\,(R - Y) + 128 = 0{,}5\,R - 0{,}4187\,G - 0{,}0813\,B + 128$$

Le webcam (ma non solo) utilizzano inoltre formati YUV compressi, allo scopo di risparmiare la quantità di byte da elaborare. Tipicamente il formato di compressione scelto è il 4:2:0, che sta a indicare che viene descritto un gruppo di 4 pixel (2

verticali e 2 orizzontali) utilizzando 6 byte di informazione: ogni pixel del gruppo possiede il proprio valore di luminanza mentre i valori U e V sono identici. Ne consegue che ogni pixel dell'immagine è descritto da 8 bit di luminanza Y, 2 bit di crominanza U e 2 bit di crominanza V, ovvero 12 bit. La scelta di risparmiare informazioni sul colore è dovuta al fatto che l'occhio umano è molto più sensibile ai cambi di luminosità rispetto a quelli di colore. In pratica l'occhio distingue facilmente tra diverse tonalità di grigio ma non è in grado di fare altrettanto con diverse sfumature cromatiche di rosso, verde e blu.

Il risparmio di byte è evidente (pari al 50% del formato classico RGB), tuttavia sorgono alcuni problemi. I dati sono acquisiti dalla webcam secondo la sequenza $Y_1 \ldots Y_n U_1 \ldots U_m V_1 \ldots V_m$. 12 bit di dati per pixel, ovvero 1,5 byte, è un valore scomodo da trattare, poiché per allocare buffer di memoria è necessario in linguaggio c++ specificare dimensioni intere in byte; di conseguenza, i buffer di memorizzazione devono essere più grandi di quanto effettivamente richiesto, con spreco di risorse. Anche la strutturazione dei dati in sequenza comporta difficoltà nel trattamento, poiché occorrono accorgimenti particolari per recuperare tutti i bit di dato relativi a un determinato pixel dell'immagine.

Per quanto riguarda l'analisi di immagini in toni di grigio il compito è facile, in quanto è sufficiente memorizzare in un buffer solo i dati relativi alla luminanza, troncando tutti i dati di crominanza. Tuttavia, nell'ambito della visione artificiale è generalmente necessario poter lavorare anche con i più usuali valori RGB (Red Green Blue) o HSV (Hue Saturation Value). Si è quindi deciso di implementare la conversione da YUV a RGB, la cui difficoltà risulta in realtà in un problema di efficienza di esecuzione, in quanto la complessità dei calcoli richiesti rischia di

gravare in maniera molto negativa sulle CPU, nell'ipotesi di rispettare i requisiti di *real time* dei sistemi in cui il Webcam Expert si trova ad operare.

Servendosi di alcuni cosiddetti *tricks* (letteralmente *trucchi*) informatico-matematici è stato possibile realizzare una funzione c++ di conversione particolarmente efficiente e veloce, in grado di ridurre al minimo i calcoli necessari alla conversione. I *tricks* hanno come scopo principale la limitazione delle operazioni di moltiplicazione e divisione. Le CPU sono infatti molto più lente nell'eseguire tali operazioni rispetto ad addizioni, sottrazioni e *shift* di bit. Per riuscire quindi a velocizzare le funzioni di conversione la funzione implementata esegue diversi *shift* di bit e si serve di tre buffer di accumulazione per ognuno dei 3 canali Y,U e V, e di un buffer per memorizzare i valori RGB convertiti; la funzione riceve in ingresso le dimensioni dell'immagine *width* ed *height* (larghezza ed altezza rispettivamente), il buffer **yuvbuf* contenente la sequenza di dati YUV, e il buffer **rgbbuf* su cui memorizzare l'mmagine codificata in RGB.

Di seguito viene presentato il codice c++ relativo alla funzione di conversione, chiamata *YUVtoRGB*:

```
void YUVtoRGB(int width, int height, unsigned char *yuvbuf,
unsigned char *rgbbuf)
{

    int line, col, linewidth;
    int y, u, v, yy, vr, ug, vg, ub;
    int r, g, b;

    const unsigned char *py, *pu, *pv;

    linewidth = width >> 1;
    py = yuvbuf;
    pu = py + (width * height);
    pv = pu + (width * height) / 4;

    y = *py++;
    yy = y << 8;
```

```
        u = *pu - 128;
        ug =   88 * u;
        ub =  454 * u;
        v = *pv - 128;
        vg =  183 * v;
        vr =  359 * v;



        for (line = 0; line < height; line++)
        {
              for (col = 0; col < width; col++)
              {
                    r = (yy +      vr) >> 8;
                    g = (yy - ug - vg) >> 8;
                    b = (yy + ub     ) >> 8;

                    if (r < 0)   r = 0;
                    if (r > 255) r = 255;
                    if (g < 0)   g = 0;
                    if (g > 255) g = 255;
                    if (b < 0)   b = 0;
                    if (b > 255) b = 255;

                    *rgbbuf++ = b;
                    *rgbbuf++ = g;
                    *rgbbuf++ = r;

                    y = *py++;
                    yy = y << 8;
                    if (col & 1)
                    {
                          pu++;
                          pv++;

                          u = *pu - 128;
                          ug =   88 * u;
                          ub =  454 * u;
                          v = *pv - 128;
                          vg =  183 * v;
                          vr =  359 * v;
                    }
              }
              if ((line & 1) == 0)
              {
                    pu -= linewidth;
                    pv -= linewidth;
              }
        }
}
```

Appendice C – L'applicazione CrossFinder

CrossFinder è un'applicazione di test dell'algoritmo di estrazione di corner di Harris. L'applicazione è scritta in c++ e sviluppata in ambiente Linux, e visualizza attraverso un pannello grafico (programmato con l'uso delle librerie QT) il corner rilevato nell'immagine corrente acquisita dalla webcam.

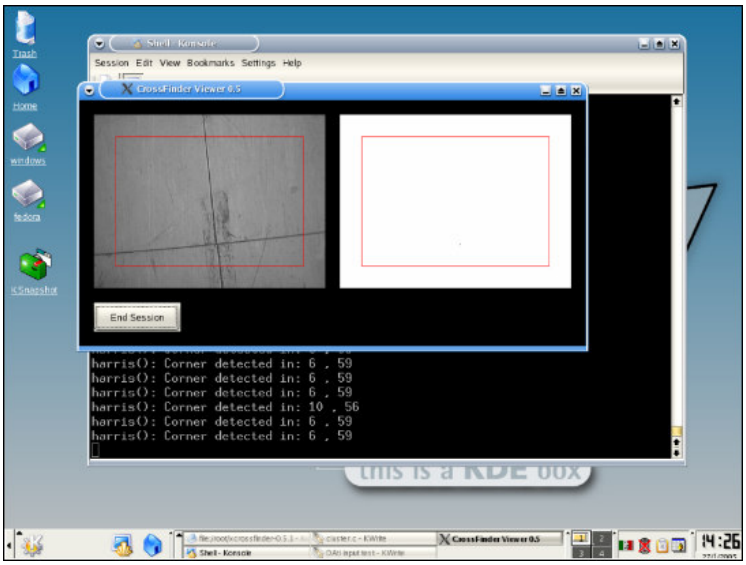


Fig. 67 - Esempio di esecuzione da shell Linux di CrossFinder Viewer, in ambiente grafico KDE 3.1.

Il pannello grafico presenta 2 finestre affiancate. In quella di sinistra viene visualizzata l'immagine corrente e l'eventuale corner individuato viene evidenziato. Nella finestra di destra viene evidenziato il corner su sfondo bianco. Lo sfondo corrisponde alla dimensione dell'immagine corrente. Il pannello di destra serve a rendere meglio visibile all'utente la posizione del corner nell'immagine in cui viene rilevato. La *shell* di esecuzione restituisce invece le coordinate del corner rilevato nel piano immagine.

CrossFinder si basa sull'architettura ETHNOS, ed è composta essenzialmente da 3 esperti:

- Esperto *Webcam* : è l'esperto che si occupa dell'interfacciamento con la webcam, è in grado di restituire immagini in YUV, RGB e in toni di grigio. All'interno di CrossFinder serve a inviare 2 messaggi ad ogni sua esecuzione. Il primo messaggio (MSG_RGB) contiene un array dei valori RGB dell'immagine corrente, utilizzati per visualizzarla nel Viewer. Il secondo messaggio (MSG_WEBCAM) contiene un array con i valori in toni di grigio dell'immagine corrente da far analizzare alla funzione che realizza l'estrattore di Harris.

- Esperto *ClusterAcq* : è l'esperto che si occupa di estarre il corner dall'immagine corrente. Riceve il messaggio MSG_HARRIS, su cui applica la funzione *harris()*, che realizza il filtraggio dell'immagine e l'estrazione del corner. L'esperto invia il messaggio MSG_CROSSHARRIS se viene individuato un corner.

- Esperto *ModuloAcq* : è l'esperto che si occupa di visualizzare il pannello grafico CrossFinder Viewer. Riceve il messaggio MSG_RGB e visualizza l'immagine corrente nella finestra di sinistra del Viewer. Quando riceve il messaggio MSG_CROSSHARRIS visualizza un cerchio rosso centrato nella posizione del corner sia nella finestra di sinistra, sia nella finestra di destra.

L'applicazione è particolarmente utile per verificare la posizione iniziale del robot. Conoscendo a priori la posizione della webcam,

è facile prevedere il punto esatto in cui deve essere visto il corner nel piano immagine. Aiutandosi con la visualizzazione dell'immagine corrente acquisita dalla webcam si può posizionare adeguatamente il robot, facendo coincidere il corner rilevato con quello previsto, un'operazione necessaria ad esempio per porre il robot nella corretta posizione iniziale.

L'applicazione gestisce una grande quantità di dati, sui quali esegue calcoli decisamente complessi e onerosi in termini di CPU. La semplice implementazione dell'algoritmo di Harris non consentiva durante le prime versioni di prova dell'applicazione di raggiungere prestazioni accettabili per un utilizzo del codice in *real time*. Si è reso quindi necessario il ricorso a tecniche di ottimizzazione del codice, in particolare nell'esperto ClusterAcq, la cui esecuzione comprende l'estrazione del corner tramite l'algoritmo di Harris. Riuscire a raggiungere velocità ragguardevoli di esecuzione (pari a 30 FPS) è stato reso possibile essenzialmente riducendo al minimo i calcoli della CPU e gli accessi in memoria. Si sono innanzitutto ridotti al minimo i cicli "for", accorpando nel frattempo assieme, ove possibile, più operazioni differenti in modo da ridurre anche gli accessi alle matrici di lavoro. Un notevole incremento di prestazioni si è verificato inoltre riducendo al minimo l'utilizzo di puntatori e *array*. Sebbene tali strutture siano particolarmente pratiche per il programmatore, la loro sostituzione con variabili statiche ha consentito di incrementare il *frame rate* di oltre 10 FPS. Si può notare nel codice a tal proposito l'utilizzo di variabili chiamate "zerozero", "zerouno", "unodue", ecc., dove prima erano presenti accessi a celle degli *array*.

Di seguito viene presentata un frammento della parte significativa del codice finale della funzione *harris()*, fulcro delle ottime prestazioni fornite dall'applicazione CrossFinder.

```
int cluster::harris()
{
  //[…]
  //costruzione matrice di lavoro *my
  for (ry = 0; ry < myheight; ry++)
  {
    for (rx = 0; rx < mywidth; rx++)
    {
          my[rx][ry] = (int)y[rx + ry * mywidth];
    }
  }

  //Sobel Filtering
  for (ry = 1; ry < (myheight-1); ry++)
  {
    for (rx = 1; rx < (mywidth-1); rx++)
    {
          zerozero = my[rx-1][ry-1];
          zerouno = my[rx-1][ry];
          zerodue = my[rx-1][ry+1];
          unozero = my[rx][ry-1];
          unouno = my[rx][ry];
          unodue = my[rx][ry+1];
          duezero = my[rx+1][ry-1];
          dueuno = my[rx+1][ry];
          duedue = my[rx+1][ry+1];

          my[rx][ry] = (zerozero + 2 * unozero + duezero -
zerodue - 2 * unodue - duedue)/4;
    }
  }
```

```
for (ry = 1; ry < (myheight-1); ry++)
  {
    for (rx = 1; rx < (mywidth-1); rx++)
    {
          zerozero = my[rx-1][ry-1];
          zerouno = my[rx-1][ry];
          zerodue = my[rx-1][ry+1];
          unozero = my[rx][ry-1];
          unouno = my[rx][ry];
          unodue = my[rx][ry+1];
          duezero = my[rx+1][ry-1];
          dueuno = my[rx+1][ry];
          duedue = my[rx+1][ry+1];

          my[rx][ry] = (zerozero + 2 * zerouno + zerodue -
duezero - 2 * dueuno - duedue)/4;
    }
  }

  //Roberts-Cross Operator, per ottenere i gradienti velocemente,
viene calcolato subito anche il quadrato
  for (ry = 1; ry < (myheight-1); ry++)
  {
    for (rx = 1; rx < (mywidth-1); rx++)
    {
          Grad_x = my[rx][ry] - my[rx+1][ry+1];
          Grad_y = my[rx][ry+1] - my[rx+1][ry];
          Gradient_x[rx][ry] = Grad_x * Grad_x;
          Gradient_y[rx][ry] = Grad_y * Grad_y;
          Gradient_xy[rx][ry] = Grad_x * Grad_y;
    }
  }

  //reset ricerca
  last = Hthr;
```

```
  //[…]
    //harris corner detection su neighbourhood 3x3
  for (ry = 30; ry < (myheight-30); ry++)
  {
    for (rx = 30; rx < (mywidth-30); rx++)
    {
          sommax = Gradient_x[rx-1][ry-1] + Gradient_x[rx][ry-1]
+     Gradient_x[rx+1][ry-1]     +     Gradient_x[rx-1][ry]     +
Gradient_x[rx][ry]   +   Gradient_x[rx+1][ry]   +   Gradient_x[rx-
1][ry+1] + Gradient_x[rx][ry+1] + Gradient_x[rx+1][ry+1];

          sommaxy = Gradient_xy[rx-1][ry-1] + Gradient_xy[rx][ry-
1]   +   Gradient_xy[rx+1][ry-1]   +   Gradient_xy[rx-1][ry]   +
Gradient_xy[rx][ry]  +  Gradient_xy[rx+1][ry]  +  Gradient_xy[rx-
1][ry+1] + Gradient_xy[rx][ry+1] + Gradient_xy[rx+1][ry+1];

          sommay = Gradient_y[rx-1][ry-1] + Gradient_y[rx][ry-1]
+     Gradient_y[rx+1][ry-1]     +     Gradient_y[rx-1][ry]     +
Gradient_y[rx][ry]   +   Gradient_y[rx+1][ry]   +   Gradient_y[rx-
1][ry+1] + Gradient_y[rx][ry+1]+ Gradient_y[rx+1][ry+1];

          //calcolo H per il punto (rx,ry)
          lambda1  =  0.5  *  (sommax  +  sommay  -  sqrt((sommax  -
sommay) * (sommax - sommay) + 4 * (sommaxy * sommaxy)));

          lambda2  =  0.5  *  (sommax  +  sommay  +  sqrt((sommax  -
sommay) * (sommax - sommay) + 4 * (sommaxy * sommaxy)));

          H[rx][ry] = (lambda1 * lambda2) - alpha * (lambda1 +
lambda2) * (lambda1 + lambda2);

          // memorizza il punto a valore maggiore
          if (H[rx][ry] > last)
          {
                last = H[rx][ry];
                temp_rx = rx;
                temp_ry = ry;
```

```
            }
        }
    }

    //prendo solo il valore più alto
    if (last > Hthr)
    {
        harriscross[temp_rx + temp_ry * mywidth] = nero;
    }
    else
    {
        cout << "harris(): No corner" << endl;
    }

    return 1;
}
```

Bibliografia

[BAL] D.H.Ballard, C.M.Brown. *Computer Vision*. Prentice-Hall, Englewood Cliffs. 1982

[BOF] J.Borenstein, L.Feng, *Measurement and correction of systematic odometry errors in mobile robots*. IEEE Transactions on Robotics and Automation 12. 1996

[BOR] J.Borenstein, H.Everett, L.Feng, D.Wehe. *Mobile robot positioning: Sensors and techniques*. Journal of Robotic Systems 14. 1997

[CAN] J.Canny. *A computational approach to edge detection*. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1986

[CAS] K.R.Castleman. *Digital Image Processing*. Prentice Hall. 1996.

[COX] I.J.Cox. *Position estimation for an autonomous robot vehicle*. Autonomous Mobile Robots: Control, Planning, and Architecture (Vol. 2). 1991

[COW] I.J.Cox, G. Wilfong. *Autonomous Robot Vehicles*. Springer-Verlag. 1990

[CSE] D.Csetverikov *Basic Algorithms for Digital Image Analysis*. PDF. 2003

[DAN] P.H.Dana. *The Global Positioning System*. PDF. 2000

[GRO] P.Gros, O.Bournez, E.Boyer. *Using Local Planar Geometric Invariants to Match and Model Images of Line Segments*. PDF. 1998

[GRW] M.Grewal, A. Andrews. Kalman filtering: theory and practice. Prentice-Hall. 1993

[HAR] C.Harris. *Geometry from visual motion*. Active Vision Vol.8. 1992

[HRA] R.M.Haralick, K.Shanmugam, I.Dinstein. *Texture features for image classification*. PDF. 1973

[HRS] R.M.Haralick, L.G.Shapiro. *Survey: Image Segmentation*. Computer Vision, Graphics, Image Processing. Vol.29. 1985

[JIA] J.Wang, W.Yang, R.Acharya. *Color clustering techniques for color-content based image retrieval from image databases*. Proc. IEEE Conf. on Multimedia Computing and Systems. 1997

[JOH] J.R.Smith, S.Chang. *Automated binary texture feature sets for image retrieval*. PDF. 1996

[KAL] R.Kalman. *A new approach to linear filtering and prediction problems*. Transactions ASME Journal of Basic Engineering 82. 1960

[LAM] B.Lamiroy, P.Gros. *Rapid Object Indexing and Recognition Using Enhanced Geometric Hashing*. Proc. of the 4th European Conf. on Computer Vision Vol. 1. 1996.

[LAU] F.Launay, A.Ohya, S.Yuta. *A Corridors Lights based Navigation System including Path Definition using a Topologically Corrected Map for Indoor Mobile Robots*. PDF. 2002

[LEI] B.J.Lei, E.A.Hendriks, M.J.T.Reinders. *On Feature Extraction from Images*. PDF. 1999

[LEO] J.Leonard, H. Durrant-Whyte. *Mobile robot localization by tracking geometric beacons*. IEEE Transactions on Robotics and Automation. 1991

[LUC] L.J. van Vliet. *Grey-Scale Measurements in Multi-Dimensional Digitized Images*. PDF. 1993

[MAY] P.S.Maybeck. *Stochastic models, estimation and control*. Academic Press Inc. 1979

[MAR] D.Marr. *Vision*. Freeman. 1982

[MAT] M.Mata, J.M.Armingol, A.de la Escalera, M.A.Salichs. A *Visual Landmark Recognition System for Topological Navigation of Mobile Robots*. PDF. 2002

[MEI] G.Meini. *Elaborazione delle Immagini Digitali*. Computer Programming Vol. 44. Infomedia. 1996

[MOR] H.P.Moravec. *Towards automatic visual obstacle avoidance*. PDF. 1979

[NEG] R.Negenborn. *Robot Localization and Kalman Filters*. PDF. 2003

[PIA] M.Piaggio. *HEIR: A non hierarchical Hybrid Architecture for Intelligent Robot*. PDF. 1998.

[RAN] K.Rangarajan, M.Shah, D.Van Brackle. *Optimal Corner Detector*. Computer Vision, Graphics, and Image Processing 48. 1989.

[SIG] A.Singhal. *Issues in Autonomous Mobile Robot Navigation*. PDF. 1997

[SIN] A.Singh, M.Shneier. *Grey level corner detection: A generalization and a robust real time implementation*. Computer Vision, Graphics and Image Processing 51. PDF. 1990

[SGA] A.Raggi, A.Sgorbissa, R.Zaccaria. *AE-SIM: Simulating Intelligent Robots in Intelligent Environments*. PDF. 2004

[SGO] M.Piaggio, A.Sgorbissa, R.Zaccaria. *A Programming Environment for Real Time Control of Distributed Multiple Robotic Systems*. PDF. 1999

[SMI] S.M.Smith. *SUSAN -- a new approach to low level image processing.* Internal Technical Report TR95SMS1. PDF. 1995.

[TAK] Y.Takeuchi, P.Gros, M.Hebert, K.Ikeuchi. *Visual Learning for Landmark Recognition*. PDF. 1997

[THF] S.Thrun, D.Fox, W.Burgard, F.Dellaert. *Robust Monte Carlo localization for mobile robots*. Artificial Intelligence Vol.128. 2001

[THR] S.Thrun. *Bayesian Landmark Learning for Mobile Robot Localization*. PDF. 1997

[WEI] A.Sanderson, A.Weiss. *Visual Servoing*. PDF. 1980

[WEL] M.Welling. *The Kalman Filter*. PDF. 2000

Per la ricerca di materiale bibliografico in formato elettronico (*identificato in Bibliografia dalla sigla "PDF", che sta a indicare un documento in formato Adobe PDF™ distribuito liberamente dall'autore*) sono risultati inoltre di fondamentale importanza i seguenti siti web:

Google Italia - *www.google.it*

Laboratorium - *www.laboratorium.dist.unige.it*

Citeseer - Scientific Digital Library - *http://citeseer.ist.psu.edu*

The Robotics Insitute - *http://www.ri.cmu.edu*

L'immagine mostrata negli esempi di filtraggio del capitolo 3 è "Mountain" di John Avon, ©1993-2004 Wizards of The Coast Inc., ed è utilizzata per gentile concessione dell'ufficio italiano di Wizards of The Coast Inc. – Hasbro.