UNIVERSITÀ DEGLI STUDI DI ROMA
"TOR VERGATA"

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA
DELL'AUTOMAZIONE

A.A. 2009/2010

Tesi di Laurea

Realizzazione e Localizzazione mediante Kalman di un robot mobile

RELATORE

Ing. Francesco Martinelli

CANDIDATO

Stefano Al Zanati

CORRELATORE

Ing. Daniele Carnevale

Se vedi una freccia volare in cielo verso l'orizzonte, troverai a terra un arco ben saldo da cui tale freccia è partita.

A mia madre e mio padre, Lidia ed Ali, per il sostegno e gli insegnamenti ricevuti da tutta una vita.

Indice

Ringraziamenti

Un ringraziamento preliminare va all'Università degli Studi di Roma "Tor Vergata", nello specifico al Dipartimento di Informatica Sistemi e Produzione e al Corso di Studi di Ingegneria dell'Automazione, per aver messo a disposizione l'uso del laboratorio di robotica pesante, con la relativa attrezzatura, e l'acquisto dei componenti utilizzati.

In particolar modo un ringraziamento all'ingegner Francesco Martinelli, che fin dall'inizio mi ha fornito articoli e testi, dai quali poter trovare le fonti teoriche necessarie, e si è sempre reso disponibile a chiarire con spiegazioni e delucidazioni i concetti più complessi.

In secondo luogo ringrazio l'ingegner Daniele Carnevale, che ha seguito ogni fase del progetto con passione, e ha avuto la pazienza di seguire la realizzazione del robot e di ovviare agli errori commessi, fornendo sia le spiegazioni teoriche sia gli aiuti manuali.

Un ulteriore ringraziamento al professor Luca Zaccarian per aver divulgato l'utilità del LaTeX, indispensabile nella stesura di questa tesi scientifica.

Infine, un ultimo grazie va dato ai miei colleghi, in particolar modo Alessandra, Giordano, Luca, Marco e Moreno, che hanno condiviso con me questo percorso dalla partenza al traguardo, fornendomi un ambiente di studio stimolante e produttivo durante l'intero corso di studi.

Introduzione

La localizzazione robotica

L'automazione ha tra le sue esigenze principali la sostituzione dell'uomo con la macchina in determinati ambiti operativi. In questo contesto si colloca la necessità di far sì che le macchine automatiche siano capaci di poter correggere autonomamente eventuali problematiche, non prevedibili in fase di progettazione, che si presentano in fase esecutiva.

Il problema della localizzazione robotica ha come obiettivo l'individuazione del percorso realmente effettuato da un sistema in movimento non guidato da utente. Tramite tale informazione è possibile apportare una correzione sugli errori di posizionamento di tale apparato mobile. Durante la movimentazione di un robot, in un ambiente noto e lungo un percorso prefissato, sembrerebbe a primo impatto non rilevante la realizzazione di alcun sistema di individuazione e correzione del tragitto. Ma la presenza di slittamenti, attriti con il pavimento, urti con eventuali ostacoli, irregolarità dei sistemi di movimentazione, fa sì che la macchina potrebbe in ogni momento deviare dalla sua traiettoria originaria, effettuando un erroneo tragitto che causa un mancato raggiungimento della posizione finale assegnata dall'utente. Anche avendo a disposizione i dati odometrici la semplice integrazione numerica produce il disallineamento tra la posizione reale e la posizione misurata (si veda [6]).

Tale problematica, non prevedibile e di conseguenza non eliminabile a priori durante il progetto, ha portato alla realizzazione di varie metodologie capaci di stimare la posizione effettiva del robot mobile nell'ambiente assegnato.
Lo studio effettuato in questa tesi riguarda l'applicazione del filtro di Kalman (si veda a titolo d'esempio [15]). Esso, infatti, oltre ai dati propriocettivi provenienti dalle informazioni odometriche, utilizza dati esterocettivi provenienti da sensori di distanza posti sulla struttura. Si rende quindi necessario l'impiego di tecniche di fusione sensoriale. L'impiego di tale filtraggio tiene conto anche dalla presenza di rumore nei dati inviati dai sensori acustici a basso costo, a cui va aggiunta la poca affidabilità della diretta integrazione dei dati tachimetrici.

Struttura della relazione

La presente relazione consta di tre capitoli principali.

- Nel primo capitolo viene descritto il robot costruito con i mezzi forniti dall'università. Si descrive la progettazione della struttura e le problematiche incontrate in fase realizzativa (si confronti [7]). Vengono quindi descritti i sistemi di movimentazione e l'apparato sensoriale ed infine il loro assemblaggio.

- Dalla fase concreta di costruzione, si passa allo studio teorico del problema della localizzazione. Si affronta inizialmente la modellizzazione del sistema, descrivendo il modello cinematico e le equazioni dei rilevamenti odometrici, per poi descrivere il modello di misura con le equazioni dei rilevamenti acustici. La relazione si occupa poi dell'esposizione de-

gli algoritmi del filtro di Kalman extended (EKF) e unscented (UKF), scrivendo le relative equazioni e i nomi assegnati alle variabili utilizzate.

- Nell'ultima parte ci si occupa dell'implementazione pratica su Matlab. Nella prima sezione si descrive la procedura di taratura dei sensori; viene poi descritta una fase simulativa di vari percorsi del robot per meglio disporre i sensori lungo la circonferenza dello stesso; segue infine l'adattamento del filtro alla reale movimentazione, la comunicazione tra Matlab ed Arduino e il resoconto delle prove effettuate con i risultati sperimentali ottenuti.

Capitolo 1

Realizzazione del robot

Viene di seguito descritto il robot realizzato: la struttura fisica, i componenti per la movimentazione e i sensori.

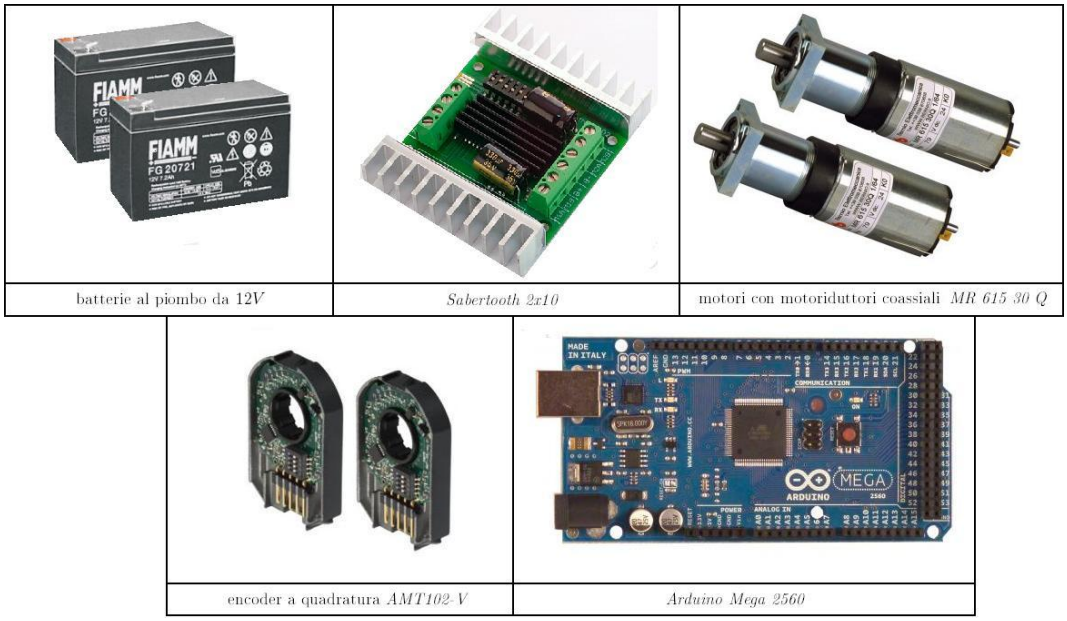
1.1 La struttura fisica

Il robot realizzato è un robot rover a due ruote motrici con guida differenziale, ovvero il controllo dei due motori a cui sono solidali le ruote è indipendente.
Il robot è stato realizzato in lega di alluminio con forma ottagonale non regolare, formata da quattro lati maggiori, di $20cm$, e quattro minori, di $15cm$.
Consta di due vani interni per il posizionamento dei componenti, mentre la struttura trasversale è posta solamente sui quattro lati minori per consentire l'assemblaggio, l'inserimento e l'estrazione della componentistica interna.
La struttura ha un altezza di $50cm$ e la diagonale maggiore misura $40cm$.

1.2 Componenti per la movimentazione

Per la movimentazione e il controllo della stessa sono stati utilizzati i seguenti componenti (rappresentati in figura 1.1):

- due batterie al piombo da $12V$;



batterie al piombo da $12V$ | *Sabertooth 2x10* | motori con motoriduttori coassiali *MR 615 30 Q*

encoder a quadratura *AMT102-V* | *Arduino Mega 2560*

Figura 1.1: I componenti utilizzati per la movimentazione del robot

- modulo di potenza *Sabertooth 2x10*;
- due motori con motoriduttori coassiali *MR 615 30 Q*;
- due encoder a quadratura *AMT102-V*;
- scheda a microcontrollore *Arduino Mega 2560*.

1.2.1 Descrizione dei componenti

Le batterie

Le batterie utilizzate sono al piombo, con tensione nominale di uscita a $12V$ ciascuna, e capacità di $3Ah$. Esse sono state collegate in serie, in maniera tale da avere un potenziale in uscita ai capi delle batterie di $24V$, ma mantenere la stessa corrente. In questa maniera nel circuito è presente una tensione massima di $24V$, con cui sono stati alimentati i motori, tramite il circuito di potenza Sabertooth.

Sabertooth

Il modulo di potenza Sabertooth è utilizzato per il controllo dei due motori in corrente continua. Di dimensioni compatte, esso sostituisce, migliorandone le performance, il ponte H, circuito utilizzato in robotica per permettere ai motori in corrente continua di poter girare in entrambi i versi. Tale modulo è dotato di limitatori di corrente e protezione termica, limitando i danni in caso di stalli accidentali dei motori del robot e di picchi di tensione. Esso presenta un'uscita a $5V$, che è utilizzata per l'alimentazione degli encoder e dei sensori[1].

I motori

I motori utilizzati nella costruzione del robot presentano due motoriduttori coassiali con rapporto di riduzione a 64, capaci di compiere 79 giri al minuto a vuoto, che scendono fino a 58 giri al minuto a seconda del carico applicato. Ricevono un'alimentazione in corrente continua ad una tensione di $12V$ o $24V$. Nell'applicazione effettuata l'alimentazione è stata fornita a $24V$ dalle batterie tramite Sabertooth, per consentire velocità maggiori.

Gli encoder

Due encoder ottici incrementali (encoder di quadratura) sono stati posizionati sui motori. Tali encoder supportano un'alimentazione in continua in un range compreso tra $3.6V$ e $5.5V$, con un minimo assorbimento di corrente, dell'ordine dei $10mA$. L'alimentazione degli encoder è fornita dalla Sabertooth, che garantisce una sicurezza contro i picchi di potenza. Gli encoder sono stati impostati con 48 impulsi ogni giro motore, che grazie al motoriduttore, impostato a 64, produce un totale di $48 \cdot 64 = 3072$ impulsi per ogni giro. Tale

[1]Per il datasheet completo si veda [5].

risoluzione è la minima disponibile, si è deciso di utilizzarla poiché un numero troppo elevato di impulsi avrebbe complicato la gestione degli stessi da parte del microcontrollore.

Arduino

La scheda Arduino Mega è basata su un microcontrollore ATmega2560[2]. Essa dispone di 54 porte digitali di input/output, delle quali 14 posso essere utilizzate come generatori in output di PWM (la modulazione a larghezza di impulso utilizzata per regolare la velocità dei motori in corrente continua). Inoltre possiede 16 input analogici, utilizzati per i segnali inviati dai sensori infrarossi. È stato collegato al computer tramite cavo USB con connessione di tipo A/B presente sulla scheda, da tale collegamento riceve l'alimentazione. La memoria consiste in: una memoria Flash da $256KB$, un modulo SRAM da $8KB$ e una EEPROM da $4KB$. Tali capacità hanno permesso una gestione agevole dei dati analogici degli 8 sensori, e dei contatori digitali dei passi encoder.

1.2.2 Assemblaggio dei componenti

I motori e gli encoder su di essi posizionati, sono stati assemblati nella parte inferiore del basamento del robot. I due motori risultato pertanto in posizione specchiata l'uno rispetto all'altro. Tale configurazione, a causa delle caratteristiche dei motori, provoca un andamento leggermente differente nel controllo degli stessi. Si supponga, ad esempio, che si voglia mandare in avanti il robot rover al massimo della velocità, e pertanto i motori si trovino a dover essere pilotati in versi opposti. Data una non elevata qualità dei motori, l'assegnazione, tramite PWM, della velocità non corrisponde per un motore al valore

[2]Per il datasheet completo si veda [2].

minimo 0 e per l'altro al valore massimo di 255. Se, infatti, tale assegnazione viene effettuata, il risultato ottenuto è una curvatura del moto, poiché vi è un motore che in tale verso di rotazione è facilitato rispetto all'altro. Nella situazione opposta, di movimento all'indietro, si è osservato il fenomeno complementare. Per tale motivazione si è deciso di rallentare il motore più veloce, assegnando un valore inferiore a quello massimo, per permettere allo stesso di ruotare ad una velocità comparabile al motore più lento, ed avere un moto risultante rettilineo.

Le batterie sono state posizionate, in un apposito slot, nel vano inferiore del robot, in posizione decentrata posteriormente, per facilitare la movimentazione in avanti dello stesso.

Nello stesso compartimento sono stati posizionati le schede Sabertooth e Arduino, per permettere un agevole collegamento tra i vari componenti. Un ulteriore accorgimento adottato, nell'invio da parte della scheda Arduino delle onde PWM per controllare i motori, è il filtraggio di tali segnali mediante un filtro passa-basso (filtro R-C), come rappresentato in figura 1.2, per avere un segnale meno soggetto a rumore.

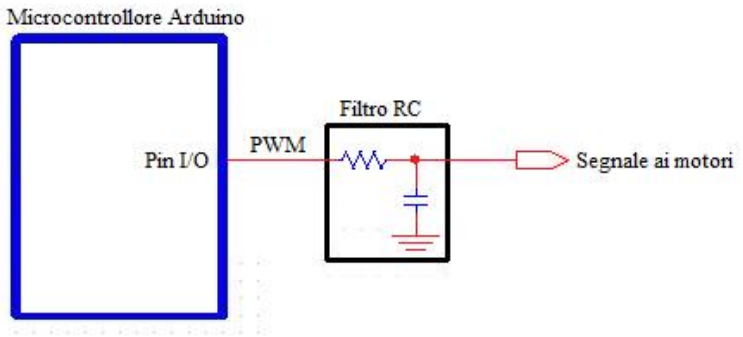


Figura 1.2: Il filtraggio utilizzato per il segnale PWM

1.2.3 Controllo dei componenti da computer

Il programma per il controllo dei motori, grazie all'uso della scheda Arduino, è stato realizzato nell'opportuno ambiente di sviluppo che implementa un linguaggio molto simile al linguaggio C/C++, il Processing/Wiring. Il listato completo del programma è riportato in Appendice B, di seguito si descrivono le funzioni principali e la codifica scelta per la comunicazione con la scheda.

La funzione *SetPwmFrequency*

L'impostazione della frequenza dell'onda PWM, generata da alcuni pin digitali della scheda Arduino Mega, è stata effettuata tramite un'apposita funzione, *SetPwmFrequency*. Tale funzione permette di modificare la frequenza con cui viene inviata l'onda quadra di modulazione a larghezza di impulso. Impostati i pin digitali, $m1 = 9$ e $m2 = 10$, cui sono collegati i motori destro e sinistro, è stato comandato un $divisor = 8$. Attraverso tali impostazioni di pin e divisor, la frequenza risultante, data la frequenza del clock di sistema a $16MHz$, corrisponde a $3906.25Hz$. Tale settaggio ha ridotto notevolmente il rumore acustico che si era riscontrato nel caso di impostazione di tensione nulla ai motori (valore medio del PWM, pari a $var1 = 127$ e $var2 = 127$, ove le variabili $var1$ e $var2$ rappresentano il valore scritto sui pin $m1$ e $m2$).

La funzione *AttachInterrupt*

Per il conteggio dei passi encoder si è sfruttato l'utilizzo di un ulteriore funzione disponibile per la scheda Arduino, la funzione *AttachInterrupt*. Per ogni encoder vengono inviati due segnali alla scheda Arduino. Durante il passaggio da basso a alto di uno dei due segnali, si verifica lo stato dell'altro segnale, sfa-

sato di 90°. In questa maniera si ha un controllo non solo del numero di step effettuati, ma anche del verso di rotazione del motore. La funzione *AttachInterrupt(pin, funzione, opzione)* facilita estremamente tale controllo. Infatti, quando il pin definito nel primo argomento della funzione cambia di stato da basso ad alto (opzione 'RISING'), viene invocata la funzione definita nel secondo argomento. Nel programma realizzato, quando i pin $encoder1 = 18$ e $encoder3 = 19$ (che corrispondono ai numeri 5 e 6 scritti in prima posizione in *AttachInterrupt*) passano da basso ad alto, vengono rispettivamente invocate le funzioni *LeggiEncoder1* e *LeggiEncoder2*. Queste ultime funzioni, controllato lo stato dell'altro segnale (rispettivamente letto dai pin $encoder2 = 4$ e $encoder4 = 7$) inviato dagli encoder, decrementano o incrementano, opportunamente, di un'unità le variabili $step1$ e $step2$, contatori dei passi encoder del motore destro e sinistro.

La lettura della stringa

La scheda Arduino è attaccata ad un netbook tramite cavo USB, l'invio dei dati da computer al microcontrollore è pertanto seriale. Il programma su pc invia sulla porta seriale una stringa di caratteri, che vengono letti da Arduino ed immagazzinati in un array s solo se i caratteri sono numerici, o lettere maiuscole. Per verificare la correttezza della stringa letta, è stato stabilito che l'ultimo carattere da inviare sia la lettera 'Q': un controllo sull'ultimo carattere immagazzinato in $s[i-1]$ è la prima condizione che deve essere, pertanto, soddisfatta.

Le stringhe inviate rispondono a comandi differenti a seconda del primo carattere.

- Se il primo carattere è la lettera 'A', il numero che segue tale lettera indica la velocità che si assegna al motore destro nel verso avanti, se invece la stringa comincia con la lettera 'D', il motore dovrà girare nel verso opposto alla velocità indicata dal numero successivo. Segue il verso di rotazione, indicato dalle lettere 'A' o 'D', per il motore sinistro, e il numero indicante la velocità assegnata a tale motore. Ad esempio la stringa 'A100A100Q' assegna sia al motore destro sia al sinistro una velocità di 100[3], ed il robot compirà un moto rettilineo nel verso avanti. La stringa 'A100D100Q', invece, produce un movimento rotazionale in senso antiorario. Rientra in questa categoria la stringa 'A0A0Q', utilizzata per fermare i motori.

- Se il carattere iniziale è la lettera 'G', eventualmente seguito dalla lettera 'I', assegna il numero, che segue, di giri al motore destro. La presenza o meno della lettera 'I' indica se tali giri vanno effettuati indietro o in avanti. L'assegnazione di un giro comanda al motore di far effettuare un giro alla ruota[4]. Segue una sequenza di caratteri analoga per l'assegnazione dei giri da far effettuare, in avanti o all'indietro, al motore sinistro. La sequenza continua con la lettera 'A', seguita o meno dalla lettera 'I' per considerazioni analoghe alle precedenti, per assegnare al motore destro un angolo da effettuare posteriormente all'ultimo giro di ruota effettuato. L'angolo è comunicato in gradi. Successivamente vi è un'analoga sequenza di caratteri per far effettuare l'angolo assegnato al

[3]Per informazioni sull'utilizzo del numero assegnato, si veda la seguente sezione *Il controllo della velocità proporzionale.*

[4]Per informazioni sul calcolo di tali giri, si veda la successiva sezione *Il controllo dei giri proporzionale integrale con scarica e desaturazione.*

motore sinistro. Per chiarezza esplicativa si dà il seguente esempio: la stringa 'G2GI2A90AI90Q' fa fare alla ruota, collegata sul motore destro, due giri e un angolo di 90° nel verso avanti, valore analogo ma nel verso opposto per la ruota sinistra, ne consegue, quindi, una rotazione del robot.

- Se la stringa comincia con la lettera 'L', il programma stampa sulla seriale i valori letti sui pin analogici, cui sono collegati gli 8 sensori infrarossi, seguiti dagli attuali passi encoder del motore destro e sinistro. Ogni valore inviato è mandato a capo, per consentire una separazione tra un numero e il successivo. La stringa inviata è pertanto 'LQ'.

Il controllo della velocità proporzionale

Come già descritto nella sezione precedente, l'utente comunica un numero corrispondente alla velocità che si vuole assegnare, con l'opportuno segno, ad entrambi i motori. Di seguito viene riportata la porzione di codice che effettua tale calcolo.

```
if (millis() - previousMillis > interval) {
    // Trasformazione in passi encoder al secondo
    vel1=(-(step1-oldstep1)*20)*0.1171875;
    vel2=((step2-oldstep2)*20)*0.1171875;
    // Memorizzazione per il successivo campionamento
    oldstep1=step1;
    oldstep2=step2;
    // Calcolo dell'errore
    deltavel1=(velrichiesta1-vel1);
    deltavel2=(velrichiesta2-vel2);
    previousMillis = millis();

    // Controllo proporzionale delle velocità
    var1=var1-deltavel1*0.17;
    var2=var2-deltavel2*0.17;
    // Assegnazione nel caso di velocità richiesta nulla
    if (velrichiesta1==0)
        var1=127;
```

```
    if (velrichiesta2==0)
        var2=127;
    // Assegnazione nel caso di velocità maggiori di quelle
       massime
    if(var2>255)
        var2=255;
    else if(var2<0)
        var2=0;
    if(var1>255)
        var1=255;
    else if(var1<0)
        var1=0;
}
```

La velocità è comunicata in step al secondo, cioè, assegnando ad esempio il valore di 100, si vuole che il robot si muova ad una velocità tale da contare 100 passi encoder ogni secondo. Il controllo di tale velocità è di tipo proporzionale. Ogni secondo si calcola la velocità attuale dal motore leggendo i passi encoder effettuati dall'ultimo campionamento, da tale velocità si calcola l'errore con la velocità richiesta dall'utente, e tale errore, moltiplicato per una costante k opportunamente calcolata, viene assegnato alle variabili $var1$ e $var2$ scritte sui pin $m1$ e $m2$ dei motori. Una schematizzazione del comportamento di tale legge di controllo è riportato in figura 1.3

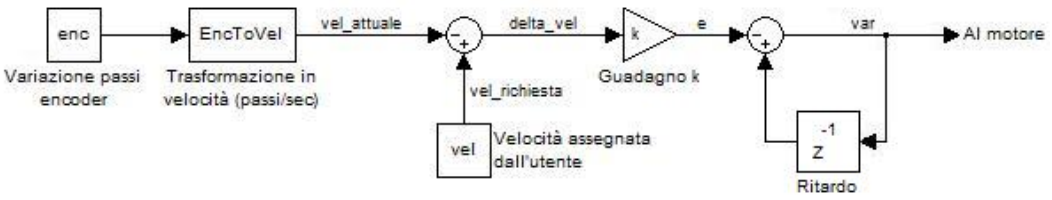


Figura 1.3: Diagramma a blocchi del controllo proporzionale della velocità

Il controllo dei giri proporzionale integrale con scarica e desaturazione

Con le lettere 'G' ed 'A' l'utente può invece comunicare quanti giri e frazioni (espressi in angoli) la ruota deve effettuare. Il controllo sui giri viene effettuato implementando una legge di controllo di tipo proporzionale ed integrale, il codice che effettua tale calcolo è il seguente.

```
if (millis() - previousMillis3 > interval) {
      relstep1=-(stepiniz1-step1);
      relstep2=(stepiniz2-step2);
      deltaenc1=steprich1-relstep1;
      deltaenc2=steprich2-relstep2;
      //Integrale
      intstep1+=deltaenc1*1;
      intstep2+=deltaenc2*1;
      //Scarica dell'integrale
      if((deltaenc1>0 && olddeltaenc1<0)||(deltaenc1<0 &&
         olddeltaenc1>0))
        intstep1=0;
      if((deltaenc2>0 && olddeltaenc2<0)||(deltaenc2<0 &&
         olddeltaenc2>0))
        intstep2=0;
      olddeltaenc1=deltaenc1;
      olddeltaenc2=deltaenc2;
      //Desaturazione dell'integrale
      if(intstep1>127)
        intstep1=127;
      else if(intstep1<-127)
        intstep1=-127;
      if(intstep2>127)
        intstep2=127;
      else if(intstep2<-127)
        intstep2=-127;
      // Assegnazione ai motori
      var1=127+deltaenc1*0.096+intstep1*0.05;
      var2=127+deltaenc2*0.096+intstep2*0.05;
      // Saturazione uscita
      if(var1>190)
        var1=190;
      else if(var1<77)
        var1=77;
      if(var2>190)
        var2=190;
      else if(var2<77)
        var2=77;
}
```

Dalla lettura degli encoder si leggono i passi attuali. Gli step richiesti sono calcolati, invece, con opportune conversioni, in base al numero di giri ed all'ampiezza dell'angolo assegnati dall'utente. Dalla differenza tra gli step richiesti e quelli attuali, si ha l'errore su cui viene effettuato un controllore di tipo proporzionale ed integrale, con opportune costanti k e kI determinate. La legge integrale prevede la scarica dell'integrale e la sua desaturazione. Vengono così calcolate le variabili $var1$ e $var2$ da assegnare ai motori. Uno schema riassuntivo è riportato in figura 1.4.

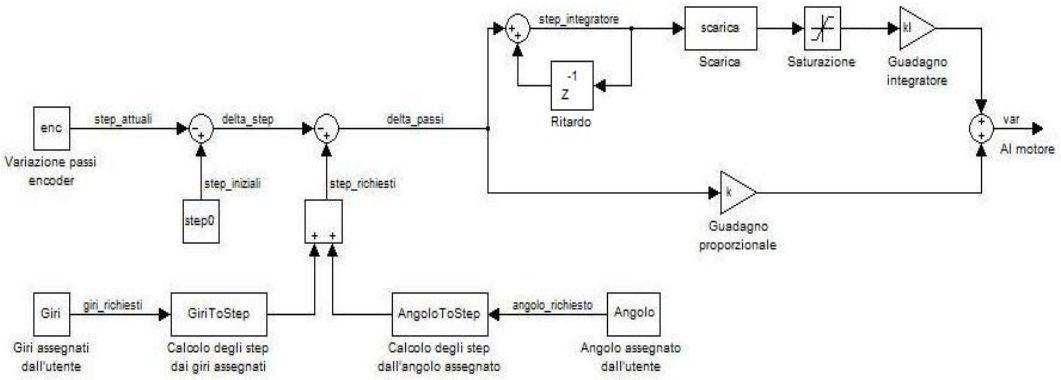


Figura 1.4: Diagramma a blocchi del controllo dei giri

1.3 Sensori

La componentistica utilizzata per i rilevamenti acustici consiste in 8 sensori a infrarossi *Sharp 2Y0A21*. Il controllo di tali sensori è gestito dalla scheda a microcontrollore Arduino Mega, mentre l'alimentazione dal modulo Sabertooth, descritti in precedenza. Una raffigurazione di tale sensore è presente in figura 1.5.

Figura 1.5: Il sensore infrarosso utilizzato

1.3.1 Caratteristiche dei sensori infrarossi

I sensori utilizzati sono dei rilevatori di distanza che utilizzano raggi infrarossi per calcolare la prossimità di un ostacolo. La parte trasmettente del sensore emette un raggio a luce infrarossa, che colpisce l'ostacolo e da esso viene riflesso verso la parte ricevente del sensore. Per tale motivazione, collegando alla porta analogica della scheda Arduino tale sensore, si legge un numero tanto più grande quanto più l'oggetto colpito è vicino. I sensori utilizzati misurano distanze tra i $10cm$ e gli $80cm$. L'alimentazione da fornire a questi sensori deve essere in un range compreso tra i $4.5V$ e i $5.5V$. I sensori sono stati alimentati dall'uscita a $5V$ del modulo di potenza Sabertooth descritto in precedenza. L'assorbimento di corrente medio di tali sensori è di $30mA$.
Il tempo di risposta è di $38 \pm 10s$, valore di cui si è tenuto conto durante la movimentazione del robot, per essere certi di leggere sui pin analogici il risultato effettivo della misurazione[5].

1.3.2 Assemblaggio dei sensori

I sensori infrarossi sono stati posizionati su una cinta perimetrale esterna alla struttura del robot. Data la previsione di collocare sul robot una torretta rotante con un'unico sensore, di qualità maggiore, e di sviluppare un sistema

[5]Per il datasheet completo si veda [3].

di visione con telecamere, è stato effettuato l'assemblaggio dei sensori su un corpo esterno rimovibile, per permettere di effettuare in futuro i rilevamenti acustici con strumenti più affidabili.

I sensori, dopo simulazioni effettuate[6], sono stati posizionati ad intervalli regolari di 45° lungo tutta la circonferenza del robot. Si è scelto di definire sensore 1 il sensore in posizione 45°, e di conseguenza il sensore 8, in posizione 360°, è il sensore frontale in direzione di movimento del robot.

Ogni sensore presenta 3 pin: alimentazione a $5V$, terra e segnale inviato. I pin dei segnali sono stati collegati direttamente alla scheda Arduino, posta nel vano inferiore del robot. I pin dell'alimentazione e della terra sono stati collegati, tramite un apposito circuito situato nel vano superiore del robot, all'alimentazione e alla terra della Sabertooth presente nel vano inferiore. Tale circuito è protetto da un interruttore e da un fusibile, per evitare dispersione di corrente, durante il non utilizzo dei sensori, che potrebbe danneggiare gli stessi.

1.3.3 Controllo dei sensori da computer

Come già detto nella sezione precedente, inviando sulla seriale la stringa 'LQ', il computer comanda al programma dell'Arduino la stampa sulla seriale dei valori dei sensori. Tale sezione di programma legge ogni sensore per 10 volte, ed invia sulla seriale il valore medio letto, per attenuare eventuali errori. Dagli esperimenti risulta, comunque, che il valore letto è costante su tutto il range operativo di tali sensori. Di seguito si riporta la parte di codice che effettua tali operazioni.

```
//Comando letture
if(s[0]=='L'){
```

[6]Per maggiori dettagli su tali simulazioni si veda il paragrafo 3.2.

```
   for(int cont=0;cont<10;contIR+){
     analogValueAverage[0]+=analogRead(ir1);
     analogValueAverage[1]+=analogRead(ir2);
     analogValueAverage[2]+=analogRead(ir3);
     analogValueAverage[3]+=analogRead(ir4);
     analogValueAverage[4]+=analogRead(ir5);
     analogValueAverage[5]+=analogRead(ir6);
     analogValueAverage[6]+=analogRead(ir7);
     analogValueAverage[7]+=analogRead(ir8);
  }
  for(int contIR=0;contIR<NumSensori;contIR++)
     Serial.println(analogValueAverage[contIR]/10);
}
```

L'interpretazione del valore analogico letto sulla porta di Arduino è affidata al programma su computer. Si è effettuata tale scelta per evitare di appesantire eccessivamente il programma presente sul microcontrollore. La conversione da valore analogico comunicato dal sensore a centimetri è stata effettuata perfezionando la legge di controllo fornita dal produttore (per maggiori dettagli si veda il paragrafo 3.1). Il robot assemblato è raffigurato in figura 1.6.

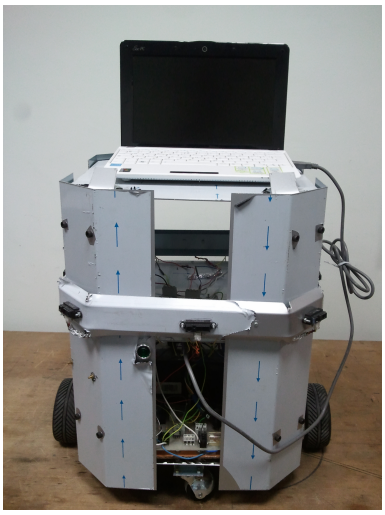


Figura 1.6: Il robot rover realizzato

Capitolo 2

Localizzazione mediante il filtro di Kalman

Questo capitolo si occupa di dare una modellizzazione del robot descritto nel precedente capitolo e della sua localizzazione tramite il filtro di Kalman esteso (EKF) e il filtro di Kalman unscented (UKF).

2.1 Modellizzazione del sistema

Lo studio è stato sviluppato sul robot descritto nel capitolo 1. Il robot acquisisce informazioni dall'ambiente esterno mediante gli encoder sui motori (dati propriocettivi) e gli 8 sensori infrarossi posti lungo la circonferenza del robot (dati esterocettivi). La fusione sensoriale (si prenda [8] come riferimento) avviene pertanto su due tipi di rilevamenti: odometrici ed acustici[1].

2.1.1 Modello cinematico e rilevamenti odometrici

Si consideri la dinamica del robot in uno spazio bidimensionale con coordinate (x_t, y_t) e orientamento θ_t. Il sistema di riferimento adottato è illustrato in figura 2.1.

Assumendo un modello discretizzato nel tempo, la dinamica di movimenta-

[1]Per maggiori dettagli si confronti [12].

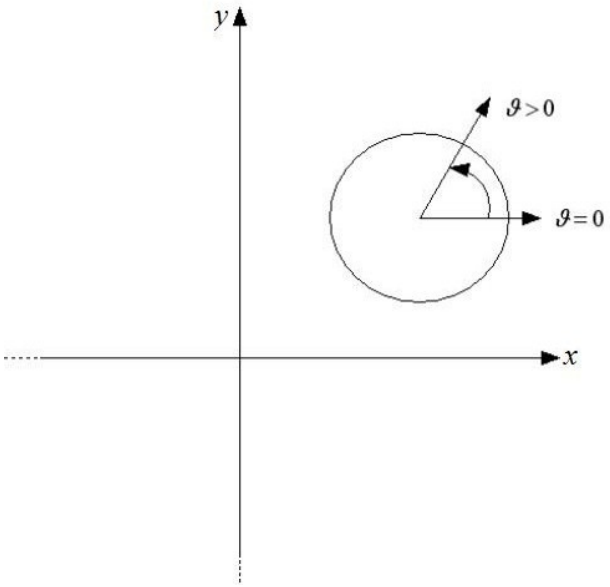


Figura 2.1: Sistema di riferimento del robot.

zione del robot è data dalle seguenti equazioni:

$$x_{t+1} = x_t + \delta\rho_t \cdot \cos\theta_t; \tag{2.1.1}$$

$$y_{t+1} = y_t + \delta\rho_t \cdot \sin\theta_t; \tag{2.1.2}$$

$$\theta_{t+1} = \theta_t + \delta\theta_t; \tag{2.1.3}$$

dove $\delta\rho_t$ e $\delta\theta_t$ sono rispettivamente lo spostamento e la rotazione durante il tempo di campionamento, collegati alle letture odometriche ed al sistema di guida del robot. Sotto l'assunzione di una guida sincrona, l'errore sulle letture odometriche[2] può essere scritto come:

$$\delta\rho_t = \delta\rho_t^e + n_{\rho,t}; \tag{2.1.4}$$

$$\delta\theta_t = \delta\theta_t^e + n_{\theta,t}; \tag{2.1.5}$$

nelle quali equazioni $\delta\rho_t^e$ e $\delta\theta_t^e$ derivano dalla lettura degli encoder del robot; mentre i rumori $n_{\rho,t}$ e $n_{\theta,t}$, la cui varianza è, per assunzione, linearmente

[2] [11].

crescente con la distanza percorsa[3], sono dati dalle relazioni:

$$n_{\rho,t} \sim \mathcal{N}\left(0, K_\rho \cdot \delta\rho_t^e\right); \tag{2.1.6}$$

$$n_{\theta,t} \sim \mathcal{N}\left(0, K_\theta \cdot \delta\rho_t^e\right); \tag{2.1.7}$$

ove K_ρ e K_θ sono due costanti positive. Si noti la distribuzione di probabilità dei rumori sul processo assunta come variabile casuale Gaussiana a media nulla. Per semplificare la notazione, si riscrivano le equazioni (2.1.1), (2.1.2), (2.1.3) come:

$$x_{r,t+1} = f\left[x_{r,t}, \delta\rho_t^e, \delta\theta_t^e, n_{\rho,t}, n_{\theta,t}\right]; \tag{2.1.8}$$

in cui si è definito con:

$$x_r \triangleq [x, y, \theta]^\top; \tag{2.1.9}$$

e con:

$$f\left[x_r, \delta\rho^e, \delta\theta^e, n_\rho, n_\theta\right] \triangleq \begin{cases} x + (\delta\rho^e + n_\rho) \cdot \cos\theta \\ y + (\delta\rho^e + n_\rho) \cdot \sin\theta \\ \theta + \delta\theta^e + n_\theta \end{cases} \tag{2.1.10}$$

2.1.2 Modello di misura e rilevamenti acustici

Il modello di misura adottato è caratterizzato da sensori infrarossi disposti lungo tutta la circonferenza del robot ad intervalli regolari (tale scelta verrà giustificata nel paragrafo 3.2). Attraverso tali sensori il robot riceve informazioni sulla distanza dalle pareti della stanza, nota, in cui si trova. In tale ambito, l'associazione dei dati attesi con quelli misurati può essere falsata dalla presenza di ostacoli non previsti lungo il percorso. Si è soliti perciò procedere ignorando dati che sono troppo distanti dai valori attesi. La funzione che descrive il modello di misura è:

$$z_{i,t} = h_i\left[x_{r,t}, \Theta\right] + \nu_{i,t}, \text{ con } i = 1, \ldots, m; \tag{2.1.11}$$

[3]Per un approfondimento si legga [10].

in cui m è il numero dei sensori infrarossi disposti lungo il robot; $z_{i,t}$ rappresenta la misura dell'i-esimo sensore al tempo t; Θ indica l'ambiente, ovvero le pareti della stanza e $\{\nu_{i,t}\}$ sono sequenze indipendenti di variabili gaussiane a media nulla e varianza $\zeta_{i,2}$. Si chiami z il vettore colonna di tutte le misurazioni z_i, e siano h e ν tali da soddisfare l'equazione:

$$z_t = h\left[x_{r,t}, \Theta\right] + \nu_t. \tag{2.1.12}$$

La lettura $z_{i,t}$ dell'i-esimo sensore al tempo t è data dalla relazione:

$$z_{i,t} = \sqrt{\left(x_t - x_{p_{i,t}}\right)^2 + \left(y_t - y_{p_{i,t}}\right)^2} + \nu_{i,t}, \text{ con } i = 1, \ldots, m; \tag{2.1.13}$$

ove $\left(x_{p_{i,t}}, y_{p_{i,t}}\right)$ rappresenta l'intersezione del raggio del sensore i-esimo con la parete colpita al tempo t, come illustrato in figura 2.2.

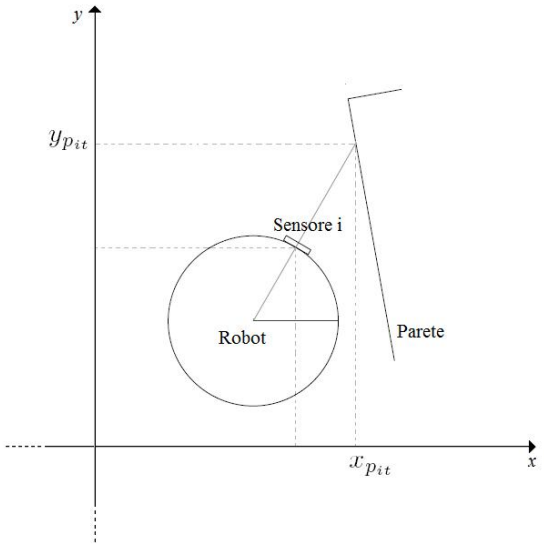


Figura 2.2: Intersezione tra raggio del sensore i-esimo e la parete.

2.2 Extended Kalman Filter

Il filtro di Kalman esteso è stato implementato apportando come unica modifica significativa l'imposizione di un valore di soglia, che permette di scartare letture che provocano un'innovazione troppo grande.

L'algoritmo si basa su un ciclo di predizione e stima: nella fase predittiva, in base allo stato precedente, viene stimato, a priori, lo stato successivo e la sua matrice di covarianza; nella fase di stima, tramite le misure z, si determina l'innovazione, si calcola il guadagno di Kalman e si aggiorna la matrice di covarianza.

In dettaglio le operazioni dell'algoritmo sono le seguenti:

- si assuma uno stato iniziale stimato $\hat{x}_{r,0}$, con matrice di covarianza data P_0 e si ponga il tempo $t = 1$;

- si calcola a priori (si noti l'apice "$-$" nelle equazioni (2.2.1) e (2.2.2)) lo stato e la sua matrice di covarianza secondo le seguenti equazioni:

$$\hat{x}_{r,t}^{-} = f\left[\hat{x}_{r,t-1}, \delta\rho_{t-1}^{e}, \delta\theta_{t-1}^{e}, 0, 0\right]; \tag{2.2.1}$$

$$P_t^{-} = A_{t-1} \cdot P_{t-1} \cdot A_{t-1}^{\top} + W_{t-1} \cdot Q_{t-1} \cdot W_{t-1}^{\top}; \tag{2.2.2}$$

  ove A_{t-1} è la Jacobiana di f rispetto a x_r al tempo $t-1$; W_{t-1} è la Jacobiana di f rispetto a $[n_\rho, n_\theta]$ al tempo $t-1$; $Q_t = diag\,(K_\rho, K_\theta)$ è la matrice di covarianza del rumore sulla movimentazione;

- si calcola l'innovazione $\text{In}_{\,i,t} \triangleq z_{i,t} - h_i\left[\hat{x}_{r,t}^{-}, \Theta\right]$ e si scarta la misura i-esima se $|\text{In}_{\,i,t}| \,/\, \max\left(z_{i,t}, h_i\left[\hat{x}_{r,t}^{-}, \Theta\right]\right)$ è più grande di una soglia prestabilita, $\forall i = 1, \ldots, m$. Si denoti quindi con $\bar{z}$ il vettore ridotto delle

misure e si indichino con una soprasegnatura tutte le variabili dopo la stessa riduzione;

- si calcola il guadagno di Kalman tramite l'equazione:

$$K_t = P_t^- \cdot \bar{H}_t^\top \cdot \left(\bar{H}_t \cdot P_t^- \cdot \bar{H}_t^\top + \bar{V}_t \cdot \bar{R}_t \cdot \bar{V}_t^\top\right)^{-1}; \tag{2.2.3}$$

dove $\bar{H}_t$ e $\bar{V}_t$ sono, rispettivamente, le Jacobiane di $\bar{h}$ rispetto a x_r e $\bar{\nu}$, mentre $\bar{R} = \overline{diag\left(\zeta_{i,2}\right)}$ è la matrice ridotta di covarianza del rumore sui dati acustici;

- si calcolano lo stato e la sua matrice di covarianza secondo le seguenti relazioni:

$$\hat{x}_{r,t} = \hat{x}_{r,t}^- + K_t \cdot \left(\bar{z}_t - \bar{h}\left[\hat{x}_{r,t}^-, \Theta\right]\right); \tag{2.2.4}$$

$$P_t = \left(I - K_t \cdot \bar{H}_t\right) \cdot P_t^-; \tag{2.2.5}$$

- sia $t = t + 1$ e si iteri nuovamente dal secondo passo.

Come si può notare dai passi dell'algoritmo, l'EKF estende il filtro di Kalman tramite il calcolo delle Jacobiane, dal momento che le funzioni f e h sono relazioni non lineari.

2.3 Unscented Kalman Filter

Il filtro di Kalman unscented è un filtro specifico per i sistemi non lineari: esso evita però le linearizzazioni, ottenute con l'ausilio delle Jacobiane, attuate dal modello extended.

L'idea di fondo del modello unscented è la generazione di un insieme deterministico di Sigma Point intorno al valor medio. Tali Sigma Point sono quindi

propagati attraverso le funzioni non lineari f ed h, dalle quali la media e la covarianza dello stato sono state trovate, generando un insieme di punti trasformati.

Durante il movimento del robot in prossimità delle pareti perimetrali, è possibile che un Sigma Point cada al di fuori della mappa, persino se la stima è abbastanza buona. In tale evenienza, il valore letto viene scartato, come nel caso dell'EKF, poiché produce un'innovazione troppo elevata.

In dettaglio, le operazioni effettuate dall'algoritmo, opportunamente modificato con il valore di soglia, sono le seguenti:

- lo stato (equazione (2.1.8)) e il rumore sulla movimentazione (equazioni (2.1.6) e (2.1.7)) hanno rispettivamente tre e due componenti, si sceglie di prendere uno stato esteso di dimensione $n_a = 5$;

- all'istante $t = 0$ si assuma uno stato iniziale stimato $\hat{x}_{r,0}$, con matrice di covarianza data P_0;

- si definisca lo stato esteso a priori come segue:

$$\hat{x}_{a,t-1} = \begin{bmatrix} \hat{x}_{r,t-1} \\ 0 \end{bmatrix}; \tag{2.3.1}$$

  e la sua matrice di covarianza come:

$$P_{a,t-1} = \begin{bmatrix} P_{t-1} & 0 \\ 0 & Q_{t-1} \end{bmatrix}; \tag{2.3.2}$$

  ove Q_t è la matrice di covarianza del rumore sulla movimentazione;

- si calcoli l'insieme di Sigma Point come:

$$[\Gamma_{t-1}, \Xi_{t-1}] = \text{ sigmaPoints}\left(\hat{x}_{a,t-1}, P_{a,t-1}\right); \tag{2.3.3}$$

in cui $\Gamma_t \in \Re^{2n_a+1}$ è un vettore di pesi; $\Xi_t = \{\zeta_{j,t}\}_{\{j=1,\ldots,2n_a+1\}}$ è l'insieme di $2n_a + 1$ Sigma Point $\xi_{j,t} \in \Re^{n_a}$ generato dalla funzione *sigmaPoints*. Per il dettaglio sulla funzione generatrice dei sigma Point si veda [9].

- per ogni Sigma Point si calcola a priori (si noti l'apice "$-$" nelle equazioni (2.3.4), (2.3.5) e (2.3.6)):

$$\xi^-_{j,t}(1:3) = f\left[\xi_{j,t-1}(1:3), \delta\rho^e_{t-1}, \delta\theta^e_{t-1}, \xi_{j,t-1}(4:5)\right]; \tag{2.3.4}$$

in cui $j = 1, \ldots, 2n_a + 1$ e con la notazione $v(i:l)$ si vogliono indicare i componenti da i a l del vettore v;

- si calcola, tramite $\xi^-_{j,t}$ appena trovato, lo stato a priori:

$$\hat{x}^-_{r,t} = \sum_{j=1}^{2n_a+1} \Gamma_{j,t} \cdot \zeta^-_{j,t}(1:3); \tag{2.3.5}$$

e la relativa matrice di covarianza:

$$P^-_t = \sum_{j=1}^{2n_a+1} \Gamma_{j,t} \cdot \left(\xi^-_{j,t}(1:3) - \hat{x}^-_{r,t}\right) \cdot \left(\xi^-_{j,t}(1:3) - \hat{x}^-_{r,t}\right)^\top; \tag{2.3.6}$$

- per ogni Sigma Point $\xi^-_{j,t-1}$, con $j = 1, \ldots, 2n_a + 1$, si calcola il vettore di misure atteso:

$$Z_{j,t} = h\left[\xi^-_{j,t}(1:3), \Theta\right]; \tag{2.3.7}$$

- da (2.3.7), si trova il vettore con i valori attesi delle misure a priori:

$$\hat{z}^-_t = \sum_{j=1}^{2n_a+1} \Gamma_{j,t} \cdot Z_{j,t}; \tag{2.3.8}$$

- si calcola l'innovazione $\text{In}_{i,t} \triangleq z_{i,t} - \hat{z}^-_{i,t}$ e si scarta la misura i-esima se $|\text{In}_{i,t}| / \max\left(z_{i,t}, h_i\left[\hat{x}^-_{r,t}, \Theta\right]\right)$ è più grande di una soglia prestabilita, $\forall i = 1, \ldots, m$. Si denoti quindi con $\bar{z}$ il vettore ridotto delle misure

e si indichino con una soprasegnatura tutte le variabili dopo la stessa riduzione.

- si calcolano quindi le matrici di covarianza come segue:

$$P_z = \sum_{j=1}^{2n_a+1} \Gamma_{j,t} \cdot \left(\bar{Z}_{j,t} - \bar{\hat{z}}_t^-\right) \cdot \left(\bar{Z}_{j,t} - \bar{\hat{z}}_t^-\right)^\top + \bar{R}; \tag{2.3.9}$$

$$P_{xz} = \sum_{j=1}^{2n_a+1} \Gamma_{j,t} \cdot \left(\xi_{j,t}^-(1:3) - \hat{x}_{r,t}^-\right) \cdot \left(\bar{Z}_{j,t} - \bar{\hat{z}}_t^-\right)^\top; \tag{2.3.10}$$

dove $\bar{R} = \overline{diag\left(\zeta_{i,2}\right)}$ è la matrice ridotta di covarianza del rumore sui dati acustici;

- si determina il guadagno di Kalman come:

$$K_t = P_{xz} \cdot P_z^{-1}; \tag{2.3.11}$$

- si determina lo stato e si aggiorna la matrice di covarianza:

$$\hat{x}_{r,t} = \hat{x}_{r,t}^- + K_t \cdot \left(\bar{z}_t - \bar{\hat{z}}_t^-\right); \tag{2.3.12}$$

$$P_t = P_t^- - K_t \cdot P_z \cdot K_t^\top; \tag{2.3.13}$$

- sia $t = t + 1$ e si iteri dal terzo passo.

Nonostante, a primo impatto, il calcolo tramite la funzione *sigmaPoints* può sembrare più complesso rispetto alla linearizzazione tramite Jacobiane, per funzioni non lineari molto complesse tale maniera deterministica risulta più semplice. Inoltre i risultati ottenuti sia in ambito simulativo sia operativo hanno evidenziato un miglioramento delle performance tramite UKF piuttosto che con la semplice estensione dell'EKF.

2.4 Schema riassuntivo sulla localizzazione

Mediante la stima a priori, attraverso i dati provenienti da encoder e sensori, tramite il filtro di Kalman si può attuare una correzione del successivo segnale di controllo da inviare ai motori. Di seguito viene riportato in figura 2.3, per chiarezza espositiva, uno schema logico riassuntivo[4] per meglio chiarire il funzionamento del filtro. Tale schema è valido per entrambe le estensioni EKF e

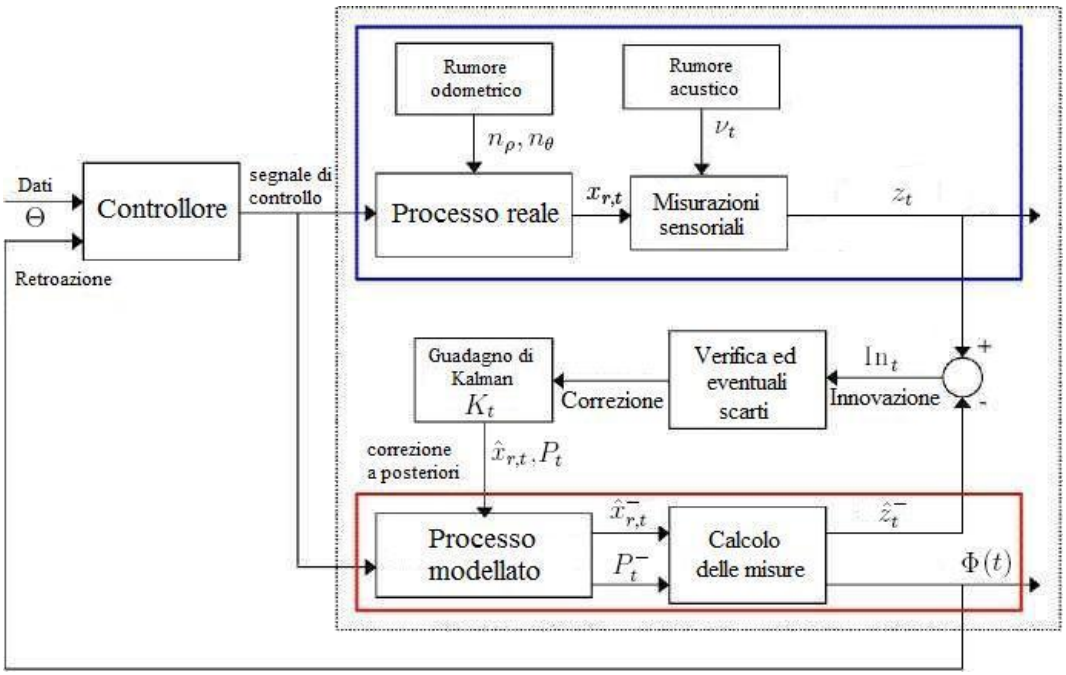


Figura 2.3: Schema generale del funzionamento del filtro

UKF. La variabile $\hat{z}_t^-$, già dichiarata per il modello unscented tramite l'equazione (2.3.8), rappresenta nel modello extended $\hat{z}_t^- = h\left[\hat{x}_{r,t}^-, \Theta\right]$. Si è inoltre dichiarato con $\Phi(t)$ lo stato filtrato da utilizzare nell'iterazione successiva.

[4]In tale schema non sono riportati tutti i collegamenti necessari per evitare un'eccessiva complicazione della figura.

Capitolo 3

Implementazione del filtro

In questo capitolo verrà esposta l'implementazione del filtro di Kalman, modellizzato nel capitolo 2, sul sistema robot descritto nel capitolo 1. Verranno enunciate le strategie attuate per la taratura dei sensori, le simulazioni effettuate su Matlab per un posizionamento degli stessi che meglio localizzasse il robot, ed infine l'implementazione pratica sul robot realizzato con i relativi programmi elaborati.

3.1 Taratura dei sensori

La prima problematica riscontrata durante la messa a punto, sul robot rover, del filtro di Kalman è stata la conversione dal valore letto dalla scheda Arduino, sulla porta analogica, e il corrispettivo della distanza in centimetri. Dal datasheet[1] fornito dal produttore del sensore vi è una curva (figura 3.1) che lega il valore letto in volt con il corrispettivo in centimetri. Dato che il massimo valore letto sulla porta analogica di Arduino è il valore 1024, che corrisponde ad una tensione di $5.0V$, con la seguente relazione:

$$\text{Valore}_{\,Volt} = \frac{\text{Valore}_{\,analogico} \cdot 5}{1024}; \tag{3.1.1}$$

si ricava il valore letto in $Volt$.

[1] [3].

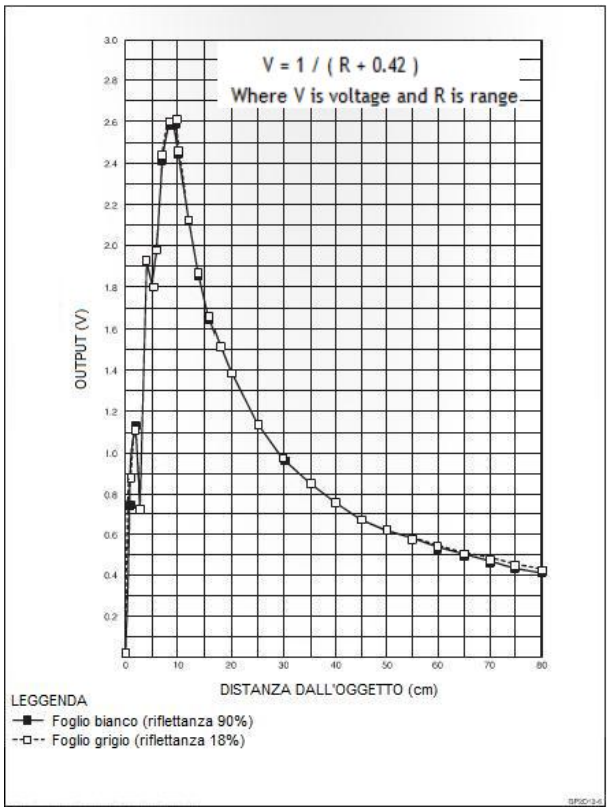


Figura 3.1: Curva output-distanza del sensore GP2D12 e legge output-distanza

La prima osservazione effettuata dalla visione della curva, è l'utilizzo di tale sensore per determinare una distanza compresa tra $10cm$ e $80cm$, come inoltre riportato sulla scheda di descrizione del prodotto. Infatti, le letture a distanze inferiori ai $10cm$, producono dei valori letti corrispondenti a misure superiori a tale soglia, e pertanto non distinguibili dalle prime. D'altra parte, si può notare che l'andamento della curva per valori superiori agli $80cm$ tende ad una costante, e pertanto non vi è una distinzione oltre questo limite superiore.
L'applicazione di un sensore su un carrello mobile, con sistema di rilevamento della distanza da una parete bianca opaca, ha, però, evidenziato che, con la legge di output-distanza fornita, vi era, in alcuni intervalli, un errore medio di $\pm 5cm$, con picchi di errore di $10cm$. In nessun caso si è osservata una corrispon-

denza ritenuta accettabile tra valore letto e valore misurato sperimentalmente. Si è riscontrato che una maggiore imprecisione avveniva nei punti ove la curva cambiava di andamento. Tale fenomeno è riconducibile al fatto che la curva è stata ottenuta dal produttore facendo una media dei valori letti, tramite riflessione da pareti bianca e grigia opache, ad un numero ridotto di intervalli, per poi approssimare con un'interpolazione polinomiale al secondo grado. È perciò plausibile dedurre che si verifichi un errore, dovuto a tale interpolazione, che produca delle uscite sbagliate in determinate regioni di piano. Inoltre, il datasheet fornito dal produttore corrisponde all'ultimo modello di sensori disponibili, successivi ai sensori utilizzati.

Per tale motivazione, inizialmente, si è pensato di procedere alla lettura di valori ogni $5cm$ (aumentando pertanto il numero di campioni a disposizione), per poi effettuare un'approssimazione tramite interpolazione polinomiale, con l'ausilio della funzione *polyfit* di Matlab, che per l'appunto calcola i coefficienti del polinomio di grado n rappresentante la funzione interpolante (e.g. [4]). Di seguito è riportata parte dell'algoritmo Matlab che effettua tale approssimazione[2].

```
%% Calcolo funzione di trasferimento

% Totale delle misurazioni effettuate: da 10 a 80 cm ogni
   5 cm
totmisure=29;
% Trasformo in volt il valore letto sulla porta analogica
volt=5.0*analogValue/1024.0;
% Grado dell'approssimazione
grado=2;
% Azzero le variabili
y=0;
valorecm(j)=zeros(1,totmisure);
```

[2]Per il listato completo si veda l'Appendice B.

```
% riempimento array cm per misure a corta gittata (10-80):
    totmisure=29
for i=1:totmisure
          cm(i)=7.5+2.5*i;    % 10; 12.5; 15; 17.5; 20;
             ...; 77.5; 80
end
% Coefficienti dell'interpolazione polinomiale
coefficienti=vpa(polyfit(volt,cm,grado),8)
```

Dai test effettuati ne è risultato che, effettivamente, con una curva di secondo grado si hanno conversioni in cm migliori che con approssimazioni di grado superiore al secondo.

La curva risultante è differente dalla precedente, ma il margine di errore era ancora superiore in modulo ai $3cm$, valore ritenuto accettabile come imprecisione su un range che va dai $10cm$ agli $80cm$. Pertanto si è proceduto approssimando la curva con una spezzata, ponendo i punti di non derivabilità ove l'errore cresceva maggiormente (figura 3.2). Con tale seconda approssimazione si è ottenuta una legge di taratura da valore in $Volt$ a valore in cm soddisfacente, con errore mai superiore in modulo ai $2cm$ e sensibilità a variazioni di $0.5cm$ su quasi tutto l'intervallo. La funzione di conversione $VoltageToRange$ appositamente definita è descritta come segue.

```
function [range]=VoltageToRange(voltage)
     % Metà inferiore della curva
     if(voltage < 0.8181650)
       if(voltage<0.5860420)
         range = -1.431295831 * voltage + 1.371201638;
       else
         range = -0.751946517 * voltage + 0.973074437;
       end
     % Metà superiore della curva
     else
       if(voltage < 1.1492463)
         range = -0.349412673 * voltage + 0.643735317;
       elseif(voltage < 1.6363301)
```

```
          range = -0.177086187 * voltage + 0.445689738;

        else
          range = -0.076191604 * voltage + 0.280592896;
        end
      end
      % Conversione in cm
      range = range*100;
end
```

Come si può notare dal precedente listato, vengono prodotte 5 porzioni di rette che meglio approssimano la curva nelle regioni di piano indicate.

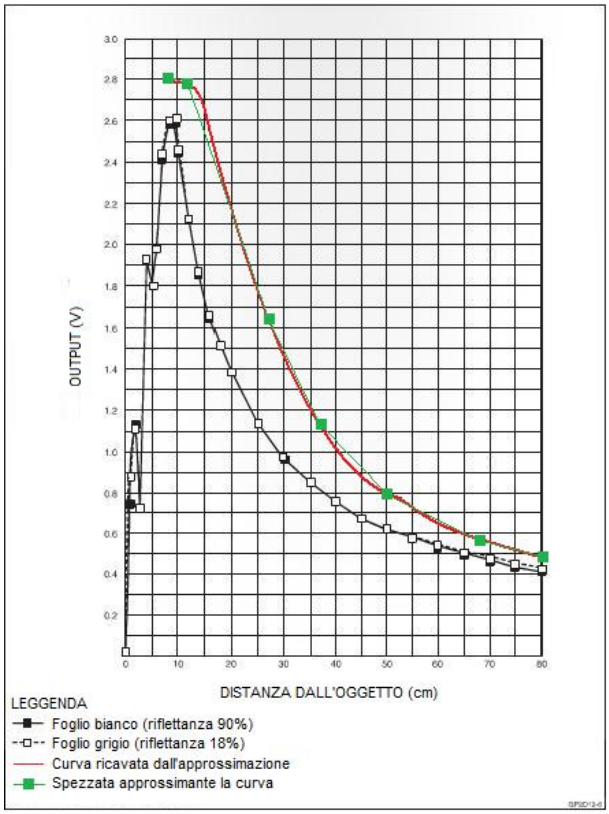


Figura 3.2: Curva output-distanza utilizzata

3.2 Simulazioni per il posizionamento dei sensori

Prima di procedere all'assemblaggio dei sensori, si è deciso di effettuare delle simulazioni, in condizioni ambientali differenti, per determinare se esistesse un posizionamento ottimale degli stessi. Sono stati realizzati cinque stanze e percorsi differenti, con peculiarità diverse, in maniera da meglio testare la scelta effettuata dei posizionamenti. Le ambientazioni, con i relativi movimenti al-

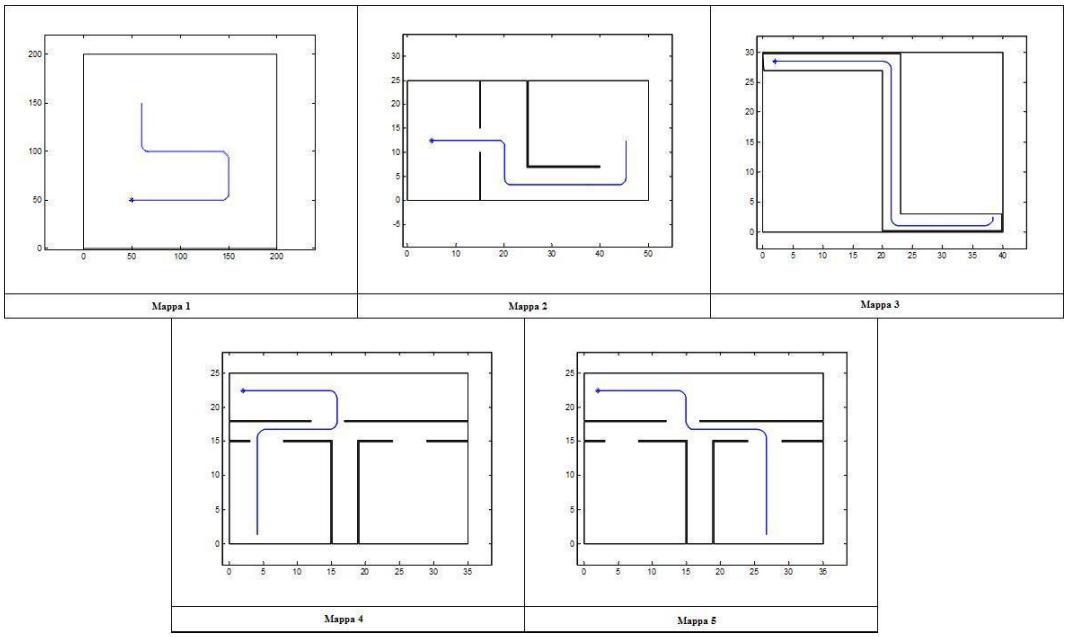


Figura 3.3: Stanze e percorsi utilizzati per la simulazione

l'interno di esse, sono rappresentate in figura 3.3.

I posizionamenti testati sono stati quattro: ad intervalli regolari sulla sola semicirconferenza anteriore del robot (in direzione di movimento), ad intervalli regolari su tutta la circonferenza del robot, ad intervalli irregolari sulla semicirconferenza anteriore ed infine ad intervalli irregolari su tutta la circonferenza del robot. In figura 3.4 sono riassunti i quattro tipi di posizionamenti.

L'ambiente simulativo è stato realizzato in Matlab, e i disturbi sulle letture

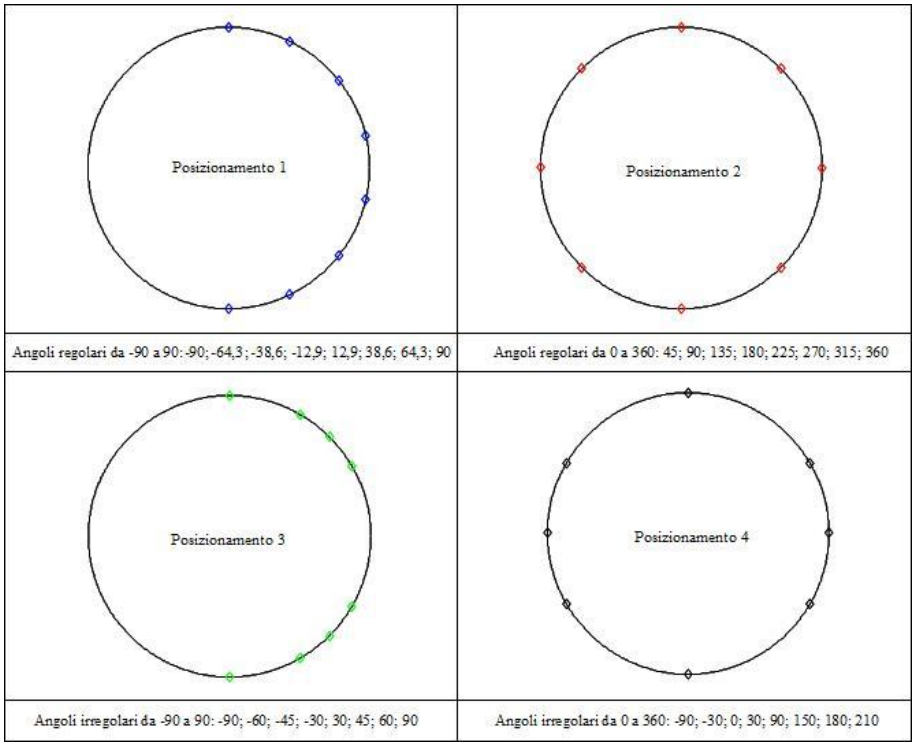


Figura 3.4: I posizionamenti considerati nella simulazione

odometriche e acustiche, essenziali per l'applicazione del filtro, sono stati generati in maniera casuale.

L'algoritmo prevede, infatti, l'assegnazione di una posizione iniziale descritta dalle variabili x_0, y_0 e θ_0, che viene corrotta aggiungendo un Δx_0, Δy_0 e $\Delta\theta_0$ calcolato tramite prodotto tra la deviazione standard degli errori della stima della posizione iniziale (σ_{x_0}, σ_{y_0}, σ_{θ_0}) e un numero pseudo-casuale normale standardizzato generato dalla funzione Matlab *randn(n)*, ove, nel caso in esame, $n = 1$ è la dimensione del vettore di numeri casuali necessari:

$$x(1) = x_0 + \sigma_{x_0} \cdot \text{ randn}(1); \tag{3.2.1}$$

$$y(1) = y_0 + \sigma_{y_0} \cdot \text{ randn}(1); \tag{3.2.2}$$

$$\theta(1) = \theta_0 + \sigma_{\theta_0} \cdot \text{ randn}(1). \tag{3.2.3}$$

In maniera analoga, sono state generate due variabili ν_ρ e ν_θ come prodotto tra la deviazione standard del rumore sulla dinamica (le varianze K_ρ e K_θ sono supposte costanti per tutta la dinamica del robot) e un vettore riga $1 \times Npassi$ di numeri casuali *randn(1,Npassi)*:

$$\nu_\rho = \text{ randn}(1, \text{Npassi}) \cdot \sqrt{K_\rho}; \tag{3.2.4}$$

$$\nu_\theta = \text{ randn}(1, \text{Npassi}) \cdot \sqrt{K_\theta}. \tag{3.2.5}$$

Infine, detta ζ la varianza del rumore acustico, per assunzione uguale per tutti i *NumSensoriIR* presenti, il rumore sulle letture sensoriali è dato dal prodotto dello scarto quadratico medio $\sqrt{\zeta}$ per una matrice $NumSensoriIR \times Npassi$ di numeri casuali *randn(NumSensoriIR,Npassi)*:

$$\text{IRNoise} = \sqrt{\zeta} \cdot \text{ randn(numSensoriIR,Npassi)}. \tag{3.2.6}$$

La presenza di variabili casuali ha reso necessaria l'esecuzione di 100 simulazioni per poter avere delle risposte statisticamente accettabili. I risultati ottenuti ad ogni iterazione sono stati memorizzati, e il risultato finale, su cui si sono fatte le osservazioni di seguito descritte, è la media delle 100 simulazioni condotte. Il seguente algoritmo esegue la stima nella maniera appena descritta.

```
sommaEKF=0;
sommaUKF=0;
Nsimulazioni=100;
% 100 simulazioni memorizzate nell'array
for i=1:Nsimulazioni
    % Lancia i dati e il percorso relativi alla stanza in
       esame
    dati_P
    % Lancia il filtro di Kalman esteso
    EKFSGL
    % Salva il valore di J_EKFsgl
    EKF(i)=J_EKFsgl;
```

```
    % Lancia il filtro di Kalman unscented
    UKFsgl
    % Salva il valore di J_UKFsgl
    UKF(i)=J_UKFsgl;
end
% Somma dei 100 elementi dell'array
for j=1:Nsimulazioni
    sommaEKF=sommaEKF+EKF(j);
    sommaUKF=sommaUKF+UKF(j);
end
% Media dei 100 elementi dell'array
mediaEKF=sommaEKF/Nsimulazioni;
mediaUKF=sommaUKF/Nsimulazioni;
% Media degli elementi scartando il valore più grande
sommaEKF=sommaEKF-max(EKF);
sommaUKF=sommaUKF-max(UKF);
mediaEKFscarto=sommaEKF/(Nsimulazioni-1);
mediaUKFscarto=sommaUKF/(Nsimulazioni-1);
```

Come si può notare dall'algoritmo, la media viene fatta sulle variabili *J_ EKFsgl* e *J_ UKFsgl*, prodotte lanciando, rispettivamente, i filtri di Kalman esteso ed unscented. Tali variabili rappresentano degli indici di prestazione normalizzati indicanti la distanza, per ognuno degli *Npassi*, della posizione stimata dagli algoritmi dal percorso effettuato dal robot (che differisce dal percorso assegnato grazie alle corruzioni dei dati odometrici precedentemente descritte).

```
% Prestazione normalizzata
J_EKFsgl=sqrt(sum((x-xHatEKFsgl).^2+(y-yHatEKFsgl).^2)/
   Npassi);
J_UKFsgl=sqrt(sum((x-xHatUKF).^2+(y-yHatUKF).^2)/Npassi);
```

L'algoritmo dettagliato che effettua l'implementazione della stanza, del percorso e la localizzazione tramite filtro di Kalman esteso e unscented, è presente in Appendice B.

Si vuole inoltre mettere in evidenza il calcolo di un valore medio approssimato,

in cui non si tiene conto del valore più alto presente. Tale valor medio con scarto è stato preso in considerazione, in luogo della reale media, nel caso in cui, a causa della casualità dei disturbi generati, vi fosse una sola occorrenza molto distante dalla distribuzione degli altri valori. La verifica di tale eventualità è stata effettuata graficando degli istogrammi delle occorrenze di ciascun valore misurato[3].

I risultati delle simulazioni sono riportati in due grafici in figura 3.5: in ascissa è presente la stanza, in ordinata il valore dell'indice di prestazione medio sulle 100 simulazioni condotte. Dalla lettura di tali risultati si deduce che i posizionamenti sulla sola semicirconferenza anteriore danno prestazioni peggiori. Tra i due posizionamenti su tutta la circonferenza, la distribuzione dei sensori ad intervalli regolari è mediamente migliore di quella ad intervalli irregolari, portando perciò a scegliere tale configurazione. Per questa motivazione, come già descritto nel paragrafo 1.3.2, i sensori sono stati assemblati sul robot rover lungo tutta la circonferenza, a distanza costante l'uno dall'altro.

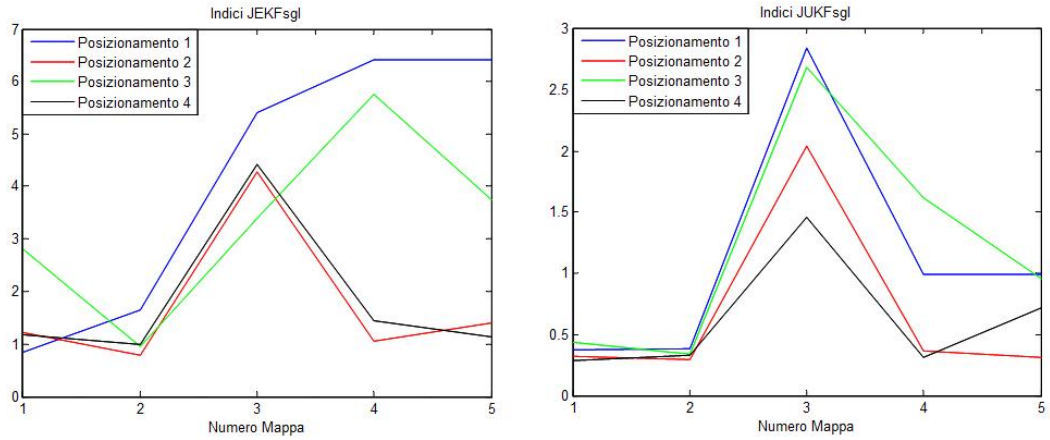


Figura 3.5: Risultati ottenuti dalle simulazioni

[3]L'algoritmo completo del calcolo della media è presente in Appendice B.

3.3 Implementazione sul robot

Dopo la taratura dei sensori, e la disposizione degli stessi sul robot rover in maniera ottimale, si è proceduto con l'adattamento del codice utilizzato in ambiente simulativo al caso reale. Le variabili che, in fase simulativa, erano corrotte da errori random, sono state sostituite con le letture dei dati odometrici e sensoriali del robot, che già presentano intrinsicamente un errore di misura. Pertanto la prima fase dell'implementazione sul caso reale consiste nella comunicazione tra la scheda Arduino e il software Matlab, su cui è implementato il filtraggio dei dati.

3.3.1 Comunicazione tra Matlab e Arduino

La comunicazione tra Matlab e Arduino avviene tramite porta seriale, collegando l'apposita porta di Arduino tramite cavo USB al netbook. La scheda Arduino è in costante ascolto sulla seriale dei comandi in arrivo da Matlab: tramite un'opportuna codifica la scheda è in grado di controllare la correttezza della stringa in arrivo, per poi eseguire il movimento comandato o inviare al calcolatore i dati richiesti[4].

In maniera speculare, il programma che codifica il filtro di Kalman richiede e riceve i dati odometrici e sensoriali dalla scheda, utilizza tali dati in combinazione con i dati precedentemente elaborati in fase di predizione, elabora un nuovo comando di movimentazione da inviare alla scheda Arduino.

Nella fase di inzializzazione il programma memorizza innanzitutto il valore dei passi encoder attuali, per poi inviare ad Arduino il primo movimento da effettuare. La generica rototraslazione viene divisa per meglio controllare il

[4]Il dettaglio sulla lettura della stringa è presente nel paragrafo 1.2.3.

robot rover. Pertanto viene assegnato per primo l'angolo, e a movimentazione terminata viene assegnata la traslazione da effettuare. Alla fine dell'intero movimento vengono richiesti e salvati i dati degli encoder e dei sensori. Uno schema riassuntivo dei passi principali della comunicazione è presente in figura 3.6. Una volta inizializzate le variabili inizia un ciclo iterativo su tutti

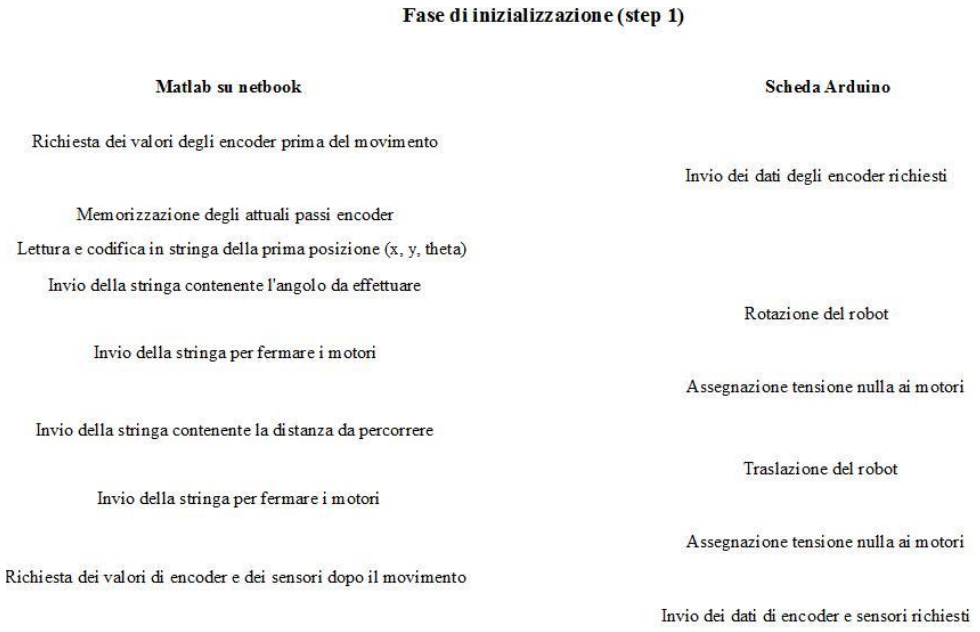


Figura 3.6: Schema riassuntivo della comunicazione tra Matlab e Arduino in fase di inizializzazione

gli *Npassi* da effettuare. Ad ogni iterazione, memorizzati i passi encoder e i valori dei sensori alla fine di un movimento, si calcola la posizione successiva tramite una predizione, corretta dai valori degli encoder. Tramite l'ausilio dei dati acustici, si può invece stimare la posizione attuale come distanza dalle pareti, e pertanto calcolare un'innovazione che corregge la posizione predetta. In tale maniera si ha una stima a posteriori, e il programma invia alla scheda le stringhe, opportunamente generate, per le movimentazioni di rotazione e traslazione. Alla fine del movimento i dati degli encoder e dei sensori vengono

aggiornati e si itera di nuovo la procedura appena descritta. Uno schema rias-

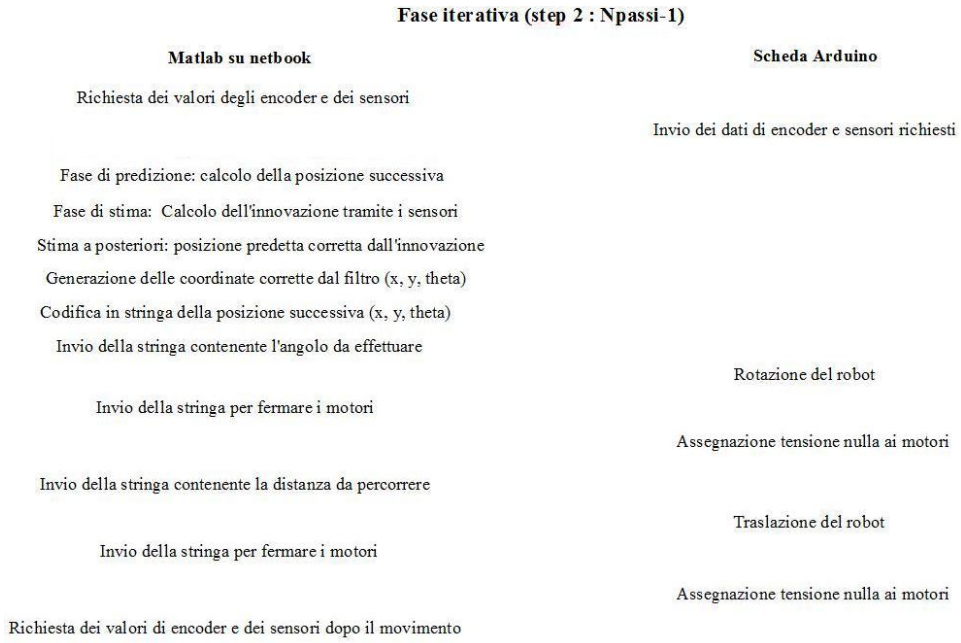


Figura 3.7: Schema riassuntivo della comunicazione tra Matlab e Arduino ad ogni step iterato

suntivo della stessa è presente in figura 3.7.

Gli algoritmi che effettuano tale filtraggio, sia per il filtro di Kalman esteso, sia per il filtro unscented, sono riportati in Appendice B, insieme alle funzioni chiamate in tali algoritmi.

La comunicazione principale che avviene da Matlab ad Arduino consiste pertanto nell'invio di stringhe di movimentazione, una volta che si sa quale sia la posizione attuale, opportunamente filtrata, e quale deve essere la posizione successiva. In questo ambito va messa in evidenza l'esigenza di una conversione da gradi o centimetri a giri da assegnare al motore. Come già descritto nel paragrafo 1.2.3, i comandi di movimento al robot rover vengono assegnati in funzione del numero di giri (o frazione di giro, data in angolo) che le ruote devono fare, in maniera indipendente l'una dall'altra. Si è cercato, pertanto,

una corrispondenza tra lo spazio o l'angolo che il robot doveva compiere, e il numero di giri che le ruote dovevano fare. Tale conversione è effettuata dalla funzione *MuoviRobot*[5], la quale, in base alle misurazioni effettuate, riesce ad operare tale conversione. Si è rilevato, infatti, che assegnando ai motori un giro in avanti (o indietro) per entrambi le ruote, il robot compiva uno spazio di $28cm$. Per multipli di un giro, lo spazio effettuato era esattamente multiplo di $28cm$: il numero di giri assegnati è pertanto calcolato come:

$$numero_{giri} = \left\lfloor \frac{centimetri}{28} \right\rfloor . \tag{3.3.1}$$

Si sono invece riscontrati problemi sulle frazioni di giro, comunicate al robot tramite angolo compreso tra 1° e 360°. Se, ad esempio, si assegna un angolo di 180°, il robot compie un percorso di $12cm$, anziché i $14cm$ attesi. Pertanto la frazione di giro attesa è calcolata a tratti in base alle misurazioni effettuate in laboratorio come segue:

$$angolo = \begin{cases} \frac{centimetri \mod 28}{26} \cdot 360, & 1 \leq centimetri \mod 28 \leq 7 \\ \frac{centimetri \mod 28}{24} \cdot 360, & 8 \leq centimetri \mod 28 \leq 14 \\ \frac{centimetri \mod 28}{28} \cdot 360, & 15 \leq centimetri \mod 28 \leq 27 \end{cases} ; \tag{3.3.2}$$

ottenendo un risultato soddisfacente, con errore massimo misurato di $0.5cm$.
Il calcolo della quantità di giri da assegnare alle ruote per compiere l'angolo di rotazione del robot è risultato, invece, molto più semplice. La spiegazione di questa differenziazione del comportamento sta nel fatto che i motori, posti in posizione speculare, lavorano alla stessa maniera se il verso di rotazione è lo stesso, e ciò si verifica in fase di rotazione, mentre, viceversa, in fase di traslazione essi girano l'uno in un verso e l'altro nel verso opposto[6]. Dagli

[5]Il listato della funzione Matlab è riportato in Appendice B.
[6]Tale problematica è stata precedentemente descritta nel paragrafo 1.2.2.

esperimenti effettuati la conversione da angolo a giri assegnati è la seguente:

$$numero_{giri} = \left\lfloor \frac{(gradi \cdot 4) + 15}{360} \right\rfloor ; \tag{3.3.3}$$

mentre la frazione di giro da assegnare in angolo è data dalla relazione:

$$angolo = (gradi \cdot 4) + 15 \mod 360. \tag{3.3.4}$$

Altre fasi di comunicazione da Matlab ad Arduino sono l'invio della stringa per fermare i motori[7] al termine dei movimenti di rotazione e traslazione (per evitare eventuali vibrazioni dovute ad un non esatto raggiungimento della posizione assegnata), e la richiesta dei dati dei sensori e degli encoder[8].
La comunicazione che è inviata dalla scheda Arduino a Matlab consiste nell'invio sulla seriale dei valori letti sulle porte analogiche a cui sono connessi i sensori, e dei valori attuali dei passi encoder per entrambi i motori.

3.3.2 Adattamento alla movimentazione

L'implementazione del filtro di Kalman utilizzata in ambiente simulativo ha necessitato di un adattamento per ovviare alle problematiche riscontrate durante le esecuzioni effettuate in laboratorio.
Il primo accorgimento che è stato effettuato è stata la riduzione del numero di *Npassi* che venivano effettuati in ambiente simulativo. La lettura dei valori degli encoder e dei sensori richiede l'interruzione della movimentazione, affinché si abbia la certezza dei valori letti. Inoltre i tempi di comunicazione con la Scheda Arduino non sono trascurabili, e si è riscontrato che l'invio di dati anche dei soli encoder durante la movimentazione del robot faceva sì che, a causa

[7] La stringa 'A0A0Q' descritta nel paragrafo 1.2.3.
[8] La stringa 'LQ' descritta nel paragrafo 1.3.3.

dei ritardi di comunicazione, portava all'elaborazione di dati non attendibili perché corrispondenti ad istanti precedenti. Per tale motivazione si è dovuto diminuire il numero dei passi in cui ricevere ed elaborare i dati, per ovviare a continue fermate del robot, e ad effettuare dopo ogni interruzione il filtraggio con i dati sia degli encoder, sia dei sensori[9].

A differenza dell'ambiente simulativo, si sono però eliminate tutte le variabili casuali generate in ambiente software, poiché le misurazioni a disposizioni presentano errori dovuti ad imprecisioni delle apparecchiature o alle approssimazioni adottate nelle conversioni effettuate da linguaggio robot a linguaggio dell'utente: dall'interpretazione del valore letto sui sensori, come descritto nel paragrafo 3.1, alla trasposizione da centimetri a giri ruota, come detto nella precedente sezione 3.3.1.

Un ulteriore discostamento dall'ambiente simulativo è presente nel non prevedere che, durante l'esecuzione del tragitto, il robot esegua una curva per raggiungere il segmento di percorso successivo a quello attuale. Ovvero, se il robot si sta muovendo dal punto P_i al punto P_j lungo il segmento $s_{i,j}$ e nella fase successiva deve raggiungere il punto P_k lungo il tratto $s_{j,k}$, in ambiente simulativo era prevista l'esecuzione di un tratto curvilineo antecedente il punto P_j che univa i percorsi $s_{i,j}$ e $s_{j,k}$. Nell'implementazione pratica, date le problematiche di gestione dei motori con comportamenti differenti nei due diversi sensi di rotazione, e data l'esigenza di fermare comunque il movimento per l'acquisizione dei dati odometrici ed acustici, si è preferito far percorrere al robot tutto il segmento $s_{i,j}$ sino al punto P_j, e da tale punto far avvenire la rotazione verso il punto P_k e la seguente traslazione lungo $s_{j,k}$.

[9] In fase simulativa si elaboravano i dati degli encoder ogni $Npassi$, mentre i dati dei sensori ogni $Nstep$, con $Nstep \leq Npassi$.

Si vuole inoltre mettere in evidenza come, in fase esecutiva, si è effettuata una scelta differente per il calcolo dell'indice di prestazione J. Infatti, mentre in simulazione l'indice calcolava quanto la stima si era discostata dal percorso effettuato, in fase esecutiva esso calcola quanto la stima si discosta dal percorso assegnato originariamente. Tale scelta è ritenuta preferibile per gli esperimenti in laboratorio, in quanto fornisce informazioni più significative sulla localizzazione dell'apparato mobile. Infatti, dalle prove effettuate, si è osservato che la posizione stimata non si discosta visivamente dalla posizione assunta realmente dal robot. Nonostante si potessero mettere in atto metodologie di stima della posizione realmente assunta dal robot[10], per semplicità operativa si è preferito operare su un indice di prestazione differente, e numericamente più rilevante. Dopo la localizzazione del robot, e quindi la stima della posizione in cui il robot si trova, si è proceduto con la correzione del percorso, affinché il sistema si riportasse sul tragitto originario. Se, ad esempio, il filtro determinasse che il robot, invece di trovarsi nel punto assegnato P_i, sia nel punto stimato $\hat{P}_i$, la movimentazione comandata ai motori è calcolata dal punto $\hat{P}_i$ al successivo P_j Sotto l'accorgimento di tali adattamenti, si è proceduto quindi all'esecuzione di prove in laboratorio.

3.3.3 Prove condotte e risultati ottenuti

Nel laboratorio di robotica pesante è stato realizzato un ambiente con pannelli di legno opachi non verniciati, di dimensioni $200cm \times 200cm$[11]. L'ambiente corrisponde alla prima mappa utilizzata in fase simulativa per il posizionamento dei sensori.

[10] Si veda a titolo d'esempio [1].

[11] Le misure e le coordinate riportate di seguito sono sempre da considerarsi in *cm*.

Il percorso assegnato al robot rover aveva come punto di partenza la posizione $P_1 = (50; 50)$ e angolo iniziale pari a $0°$[12]. Esso deve raggiungere lungo il tratto $s_{1,2} = 100cm$ il punto $P_2 = (150; 50)$, effettuare una rotazione di $+90°$ per dirigersi lungo $s_{2,3} = 50cm$ verso il punto $P_3 = (150; 100)$. Da tale posizione effettuare un'ulteriore rotazione di $+90°$ per orientarsi verso il punto $P_4 = (60; 100)$ e traslare lungo $s_{3,4} = 90cm$, ed infine, con una rotazione di $-90°$, raggiungere il "goal" in posizione $P_5 = (60; 150)$ percorrendo $s_{4,5} = 50cm$. L'ambiente e il percorso sono riportati in figura 3.8. Nonostante la semplicità

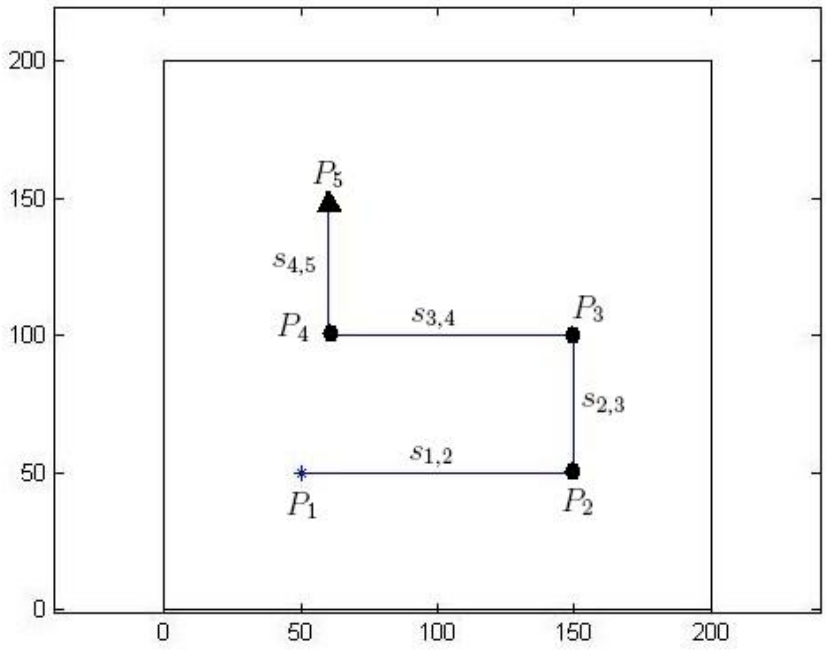


Figura 3.8: Il percorso utilizzato negli esperimenti pratici

del percorso, tenendo conto delle dimensioni fisiche del robot, il cui raggio è di $40cm$, il robot si trova ad effettuare il primo e l'ultimo tratto molto vicino alle pareti perimetrali della stanza. Inoltre tale percorso è stato pensato per verificare la correttezza di rotazioni sia positive sia negative.

[12]Per la convenzione utilizzata per le coordinate x, y, θ si veda la figura 2.1.

Gli esperimenti effettuati sono stati condotti fermando ed acquisendo i dati nei punti P_i, con $i = 1, \ldots, 5$, del percorso, e dividendo i tratti $s_{i,j}$ in proporzione alla loro lunghezza, in segmenti di $50cm$ $s^m_{i,j}$, con $m = 1, \ldots, \lceil \frac{s_{i,j}}{50} \rceil$.
Il percorso risultante, ove si è indicato con $P^k_{i,j}$, con $k = 1, \ldots, m-1$ il numero dei punti intermedi sul tratto $s_{i,j}$, è raffigurato in figura 3.9. Il maggior pro-

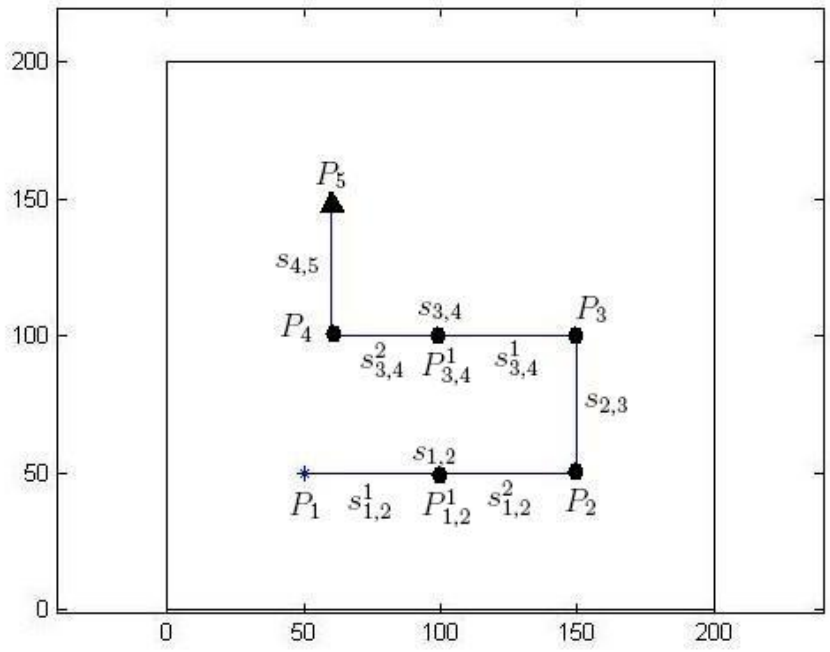


Figura 3.9: Il percorso utilizzato diviso in tratti

blema riscontrato è stato il differente comportamento dei motori che girano, durante la traslazione in avanti, in versi opposti. Data la scarsa qualità degli stessi, le due rotazioni delle ruote sono differenti, pertanto il robot si trova a non percorrere un tratto rettilineo, ma in diagonale, data l'iniziale prevalenza del motore destro rispetto all'altro, per poi ritrovarsi orientato nella giusta posizione, poiché il motore sinistro ha un ritardo anche in decelerazione. Il percorso effettuato, su un qualunque tratto $s^m_{i,j}$ è riportato in figura 3.10. Ulteriormente a tale errore di traslazione, si sono verificati slittamenti dovuti alla scarsa ade-

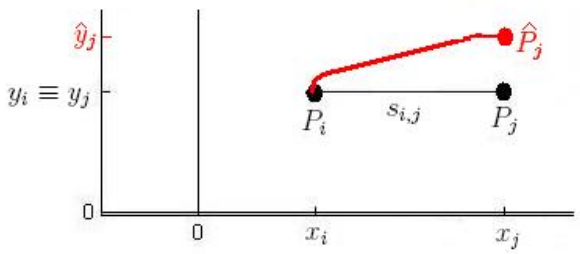


Figura 3.10: L'errore in traslazione causato dal differente comportamento dei motori

renza delle ruote sul pavimento, specialmente nei punti di interruzione tra una mattonella e l'altra e deviazioni dalla traiettoria originale causate dall'iniziale differente orientazione delle rotelle non motrici.

La presenza di tali fenomeni è stata stimata sia con il filtro di Kalman esteso, sia con il filtro di Kalman unscented. Tramite tale localizzazione è stata effettuata una correzione della movimentazione per poter permettere un ritorno sul percorso originario ed il conseguente raggiungimento dell'obiettivo finale. L'algoritmo di controllo utilizzato per tale correzione è di tipo proporzionale con guadagno unitario. Tale risultato è stato soddisfatto con una buona approssimazione, si è ottenuta un'imprecisione minore nel caso di filtraggio tramite il modello unscented.

Il calcolo delle posizioni assunte dal robot e dell'indice di prestazione J_EKF e J_UKF è stato effettuato prendendo il percorso originario assegnato al robot rover, e mettendolo in correlazione con le posizioni stimate dagli algoritmi di localizzazione durante l'esecuzione pratica del percorso in laboratorio. Il percorso, con entrambe le tipologie di filtro, è stato fatto eseguire al robot 5 volte, per verificare la distribuzione dei risultati ottenuti. I dati raccolti sono graficati in figura 3.11. Sono stati fatti effettuare al robot anche 2 percorsi ponendo

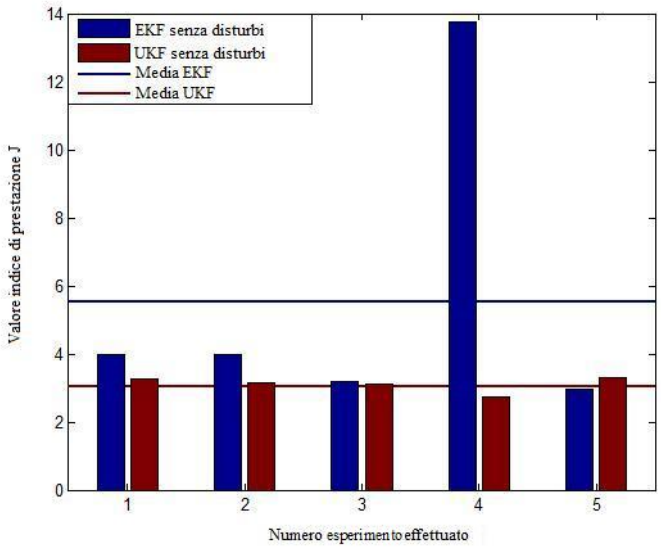


Figura 3.11: Grafico dei risultati ottenuti su 5 esecuzioni con entrambe le tipologie di filtro

dei fogli di carta a terra, che ne diminuiscono l'aderenza con il terreno, e 2 percorsi con un ostacolo nell'ambiente non previsto. Nel primo caso i risultati ottenuti, nonostante il successo della localizzazione, hanno prodotto correzioni del tragitto peggiori dei precedenti, che hanno comunque soddisfatto il raggiungimento della posizione finale con un margine di errore superiore. Con la presenza di ostacoli, invece, il raggiungimento del "goal" è fallito, poiché i dati stessi della localizzazione sono risultati erronei. In figura 3.12 sono riportati i dati raccolti con entrambi i disturbi cui è stato sottoposto il percorso del robot. Dall'analisi dei risultati si deduce che il filtro di Kalman unscented funziona mediamente meglio del filtro di Kalman esteso in caso di percorso normale. Infatti, mentre l'UKF ha riscontrato nelle prove in laboratorio un indice di prestazione pressoché costante, l'EKF, benché a volte migliore del modello unscented, presenta tuttavia notevoli discostamenti dal valore medio, risultando

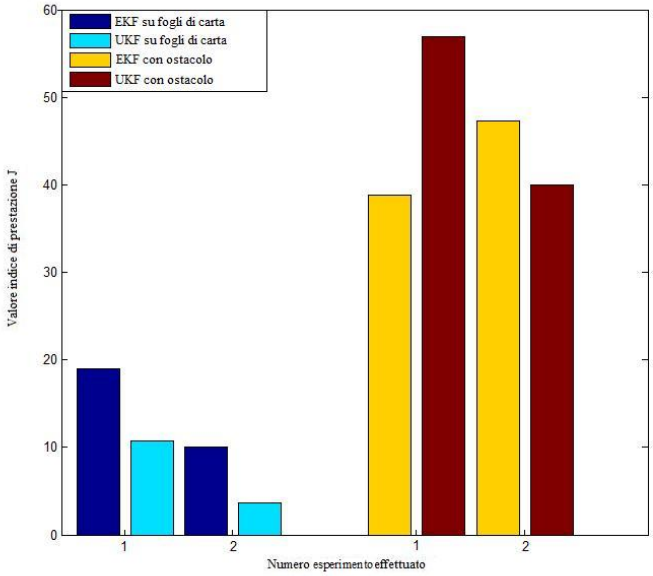


Figura 3.12: Grafico dei risultati ottenuti su 2 esecuzioni ciascuno con fogli di carta a terra e ostacolo nell'ambiente

pertanto meno affidabile del primo[13]. Anche in presenza di forti slittamenti, dovuti al posizionamento di fogli di carta sul pavimento che limitavano l'aderenza delle ruote, la correzione del tragitto tramite la stima derivata dall'UKF è risultata migliore dell'EKF. Invece, nel caso di presenza di ostacoli non noti, dal momento che i dati sensoriali non sono più attendibili, poiché essi misurano le distanze dall'ostacolo e non dalle pareti dell'ambiente in cui il robot è in movimento, il filtraggio opera su dati non realistici. Di conseguenza fallisce la stessa localizzazione, pertanto le prestazioni di entrambi i filtri sono risultate fallimentari.

Il filtro di Kalman ha come scopo principale l'individuazione dell'effettiva posizione del robot, cui si è fatta susseguire una correzione del percorso. Esso non è,

[13]La tabella riportante i valori di discostamento da ogni posizione di misura, ed una ricostruzione del percorso effettuato per ogni simulazione è riportata in Appendice A.

pertanto, adatto a rilevare ed aggirare ostacoli non noti. Per tale problematiche devono essere attuate ed implementate altre strategie operative.

Capitolo 4

Conclusioni e sviluppi futuri

Con la presente relazione si è messo a punto un filtro abbastanza soddisfacente, nonostante la bassa qualità degli strumenti utilizzati. Infatti, anche se i rilevamenti acustici erano soggetti ad errore, grazie alla presenza di un numero sufficiente di sensori, e alle potenzialità del filtro di Kalman, nelle esecuzioni in laboratorio non soggette a disturbi, la localizzazione, ed il conseguente ritorno sul percorso originario, sono stati raggiunti ad ogni esecuzione, con un margine di errore trascurabile rispetto alle dimensioni fisiche del robot.

Lo sviluppo futuro del robot rover realizzato prevede l'installazione di una torretta rotante (si veda [13]), dotata di un unico sensore laser ad alta qualità. Con tale strumento, rimossa la cintura sensoriale ad infrarossi, si disporrà di dati più precisi ed affidabili, che potranno contribuire a migliori risultati sulla fusione sensoriale effettuata dal filtro di Kalman.

In aggiunta, si vuole installare sul robot un sistema di visione stereoscopica con telecamere (si confronti [14]). Anche tale dato può essere aggiunto ai rilevamenti acustici da fornire al sistema di filtraggio per ricondurre il robot sul percorso originario.

Appendice A
Tabelle e grafici degli esperimenti condotti

Vengono di seguito riportati in tabella le posizioni $(x;y)$ assunte dal robot, nei grafici il confronto tra le posizioni effettuate (in rosso) con le posizioni assegnate (in blu).

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	150.0	150.0	100.0	60.00	60.00
y	50.00	50.00	50.00	100.0	100.0	100.0	150.0

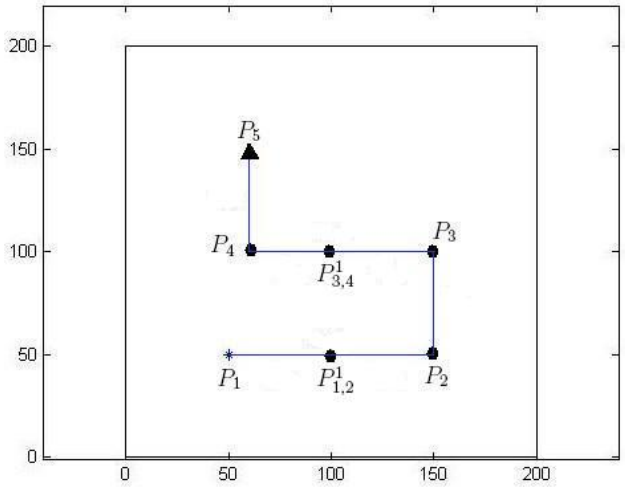


Figura A.1: Tabella e grafico delle posizioni assegnate al robot

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	146.9	149.5	100.2	65.10	66.30
y	50.00	57.00	49.60	100.1	100.5	101.3	148.4

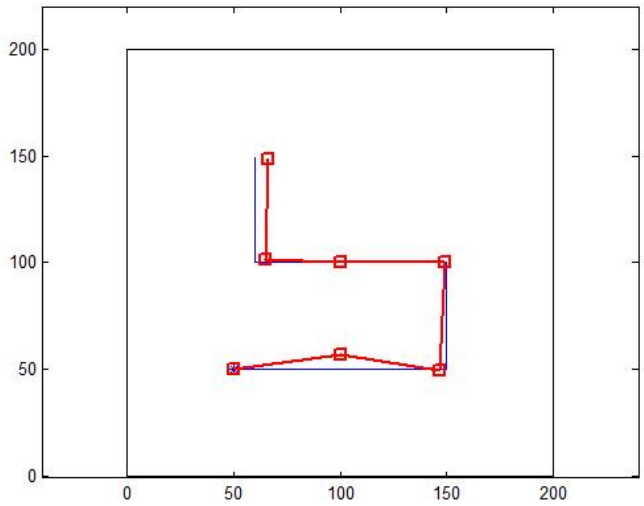


Figura A.2: Tabella e grafico della prima esecuzione di EKF

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	149.2	145.0	99.90	64.90	59.70
y	50.00	57.00	53.40	101.9	102.3	100.7	147.1

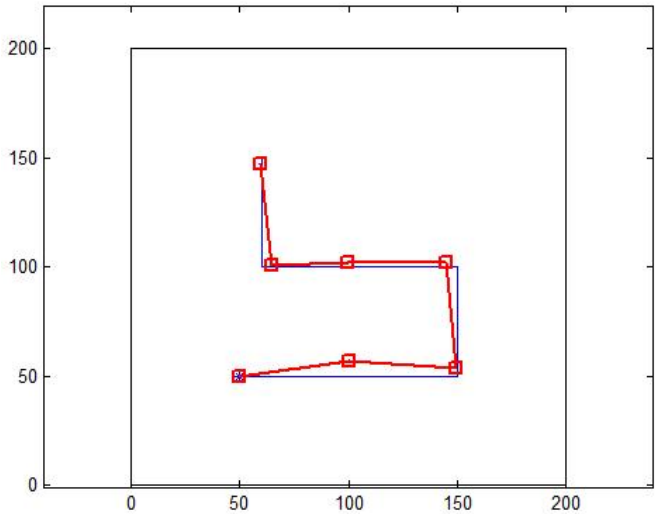


Figura A.3: Tabella e grafico della seconda esecuzione di EKF

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	151.7	150.4	100.2	60.80	61.70
y	50.00	57.00	51.00	100.5	104.0	101.7	154.9

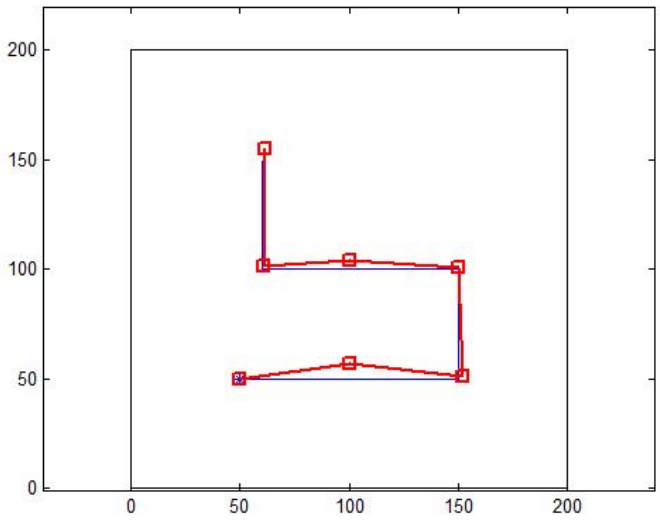


Figura A.4: Tabella e grafico della terza esecuzione di EKF

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	150.2	125.0	99.70	63.90	62.90
y	50.00	57.00	64.90	106.3	102.9	97.30	154.4

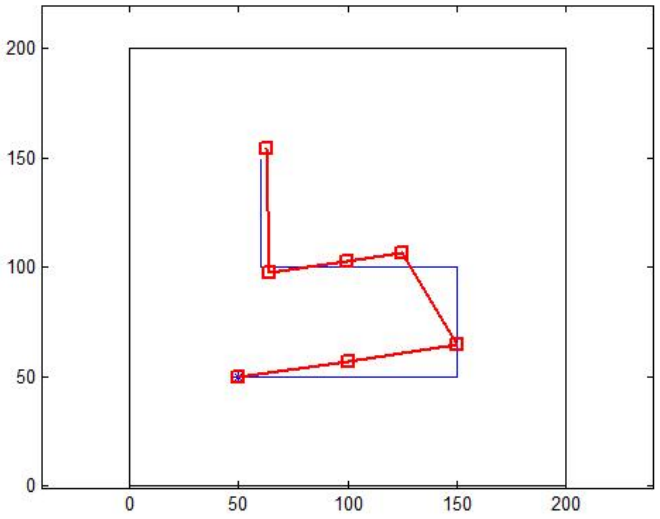


Figura A.5: Tabella e grafico della quarta esecuzione di EKF

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	147.8	150.4	102.2	62.40	61.20
y	50.00	57.00	51.10	96.30	97.60	101.6	148.1

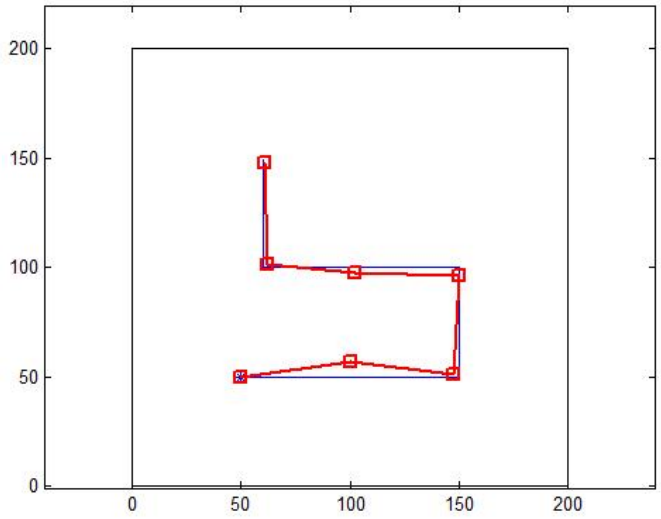


Figura A.6: Tabella e grafico della quinta esecuzione di EKF

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	150.1	149.2	103.8	59.80	61.90
y	50.00	57.00	51.40	102.0	103.5	103.1	152.6

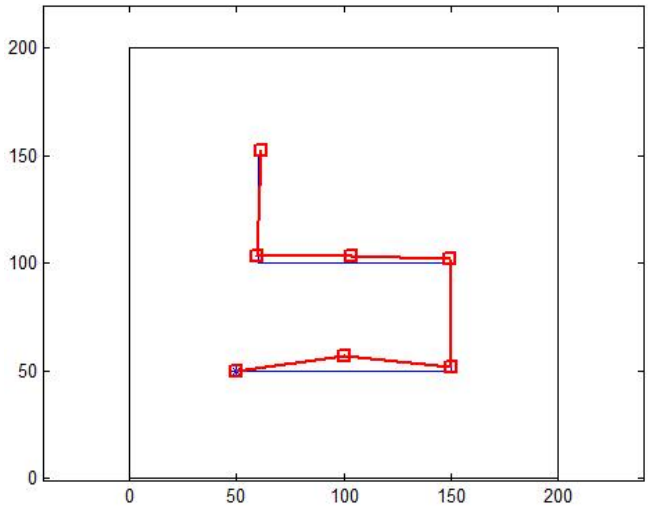


Figura A.7: Tabella e grafico della prima esecuzione di UKF

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	150.7	144.8	100.6	59.40	56.40
y	50.00	57.00	50.00	101.8	100.3	100.6	147.9

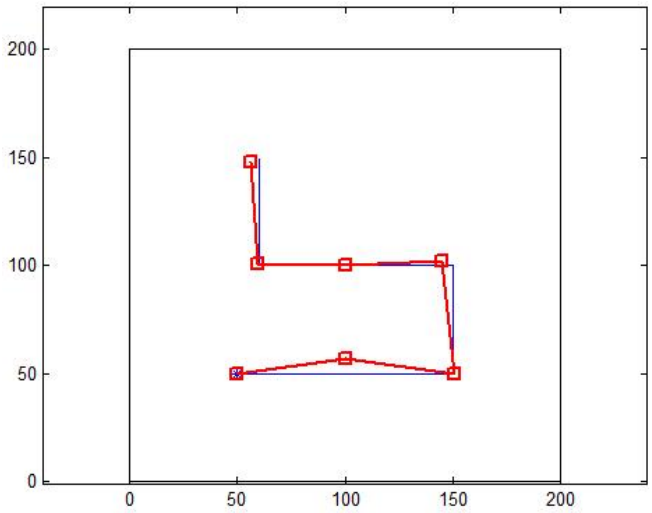


Figura A.8: Tabella e grafico della seconda esecuzione di UKF

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	150.6	147.7	102.0	59.90	58.30
y	50.00	57.00	51.00	99.40	98.70	104.4	146.2

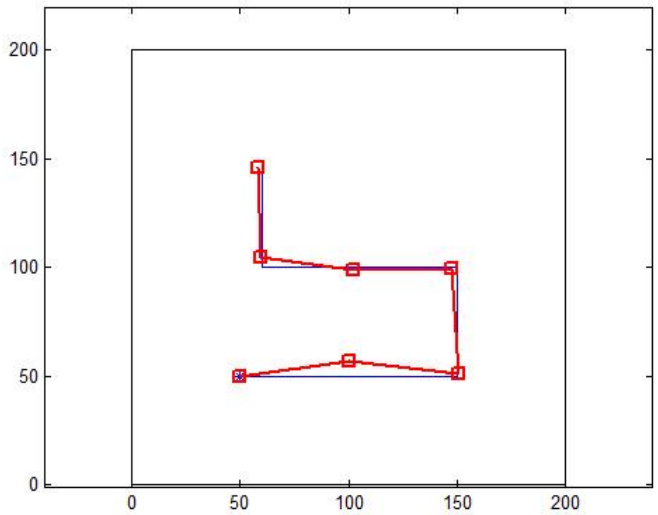


Figura A.9: Tabella e grafico della terza esecuzione di UKF

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	150.6	150.9	102.3	61.10	59.10
y	50.00	57.00	54.50	99.50	100.1	99.10	152.7

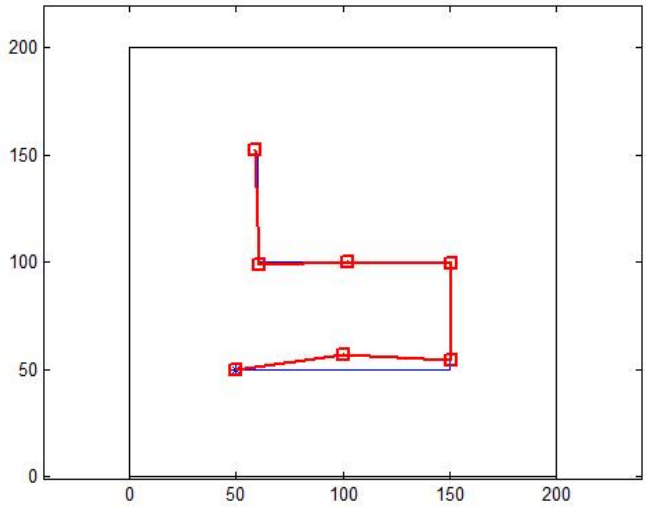


Figura A.10: Tabella e grafico della quarta esecuzione di UKF

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	152.1	149.6	101.8	60.90	56.40
y	50.00	57.00	48.30	98.30	101.1	97.20	145.7

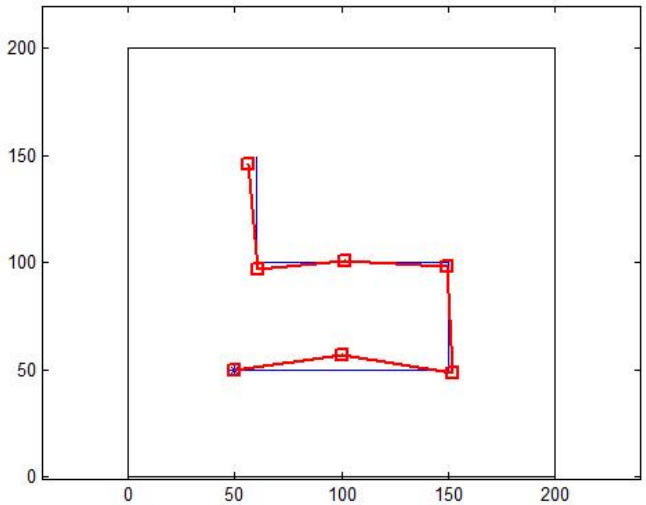


Figura A.11: Tabella e grafico della quinta esecuzione di UKF

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	148.4	138.7	114.7	64.40	69.80
y	50.00	57.00	81.43	117.8	95.50	101.3	151.8

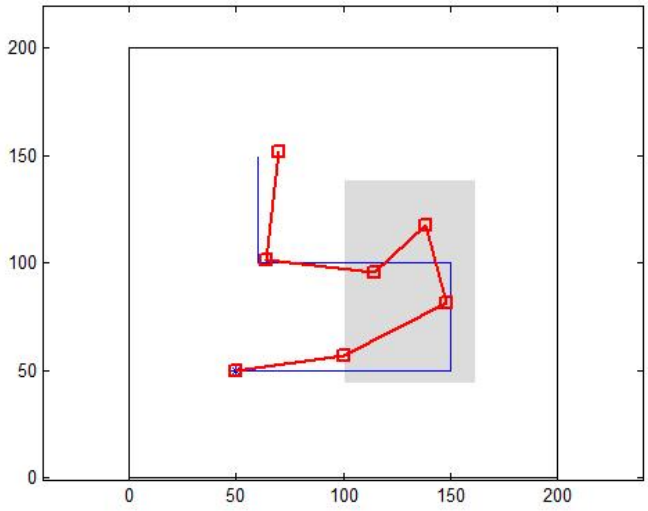


Figura A.12: Tabella e grafico della prima esecuzione di EKF su fogli di carta

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	141.0	150.2	103.9	60.20	60.70
y	50.00	57.00	64.40	110.9	100.7	108.6	152.8

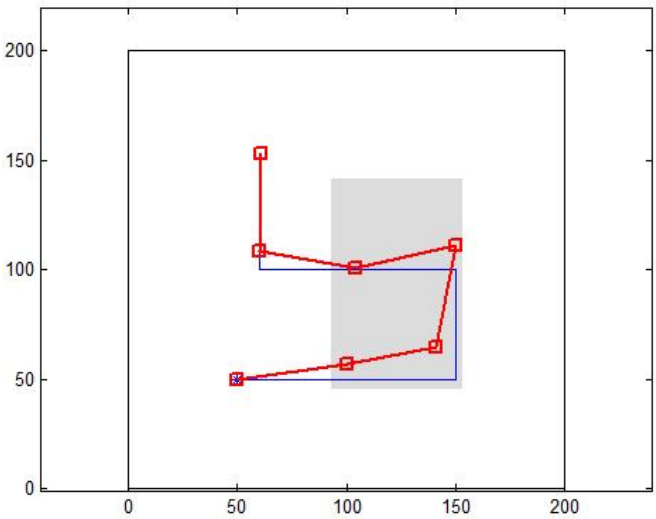


Figura A.13: Tabella e grafico della seconda esecuzione di EKF su fogli di carta

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	150.3	149.6	99.20	80.00	60.80
y	50.00	57.00	51.00	105.0	100.0	112.1	151.0

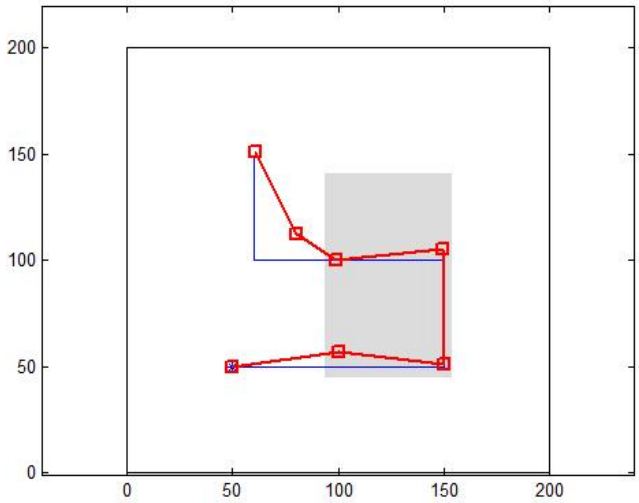


Figura A.14: Tabella e grafico della prima esecuzione di UKF su fogli di carta

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	150.7	149.6	103.9	60.70	56.30
y	50.00	57.00	50.00	99.50	105.4	100.8	147.5

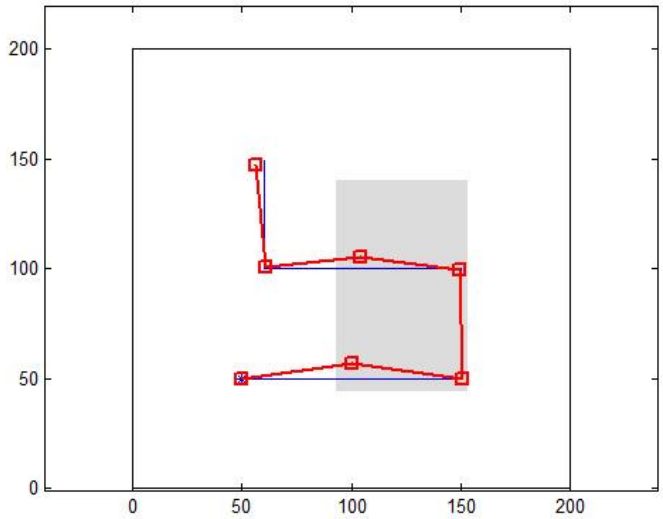


Figura A.15: Tabella e grafico della seconda esecuzione di UKF su fogli di carta

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	147.4	138.1	101.8	98.34	50.30
y	50.00	57.00	68.50	138.8	112.3	77.00	92.80

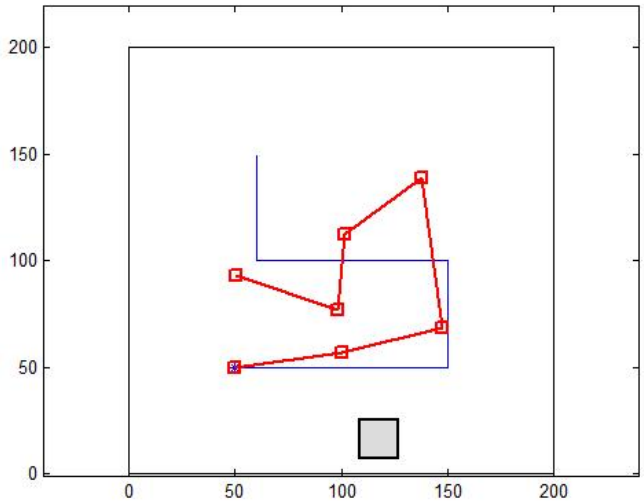


Figura A.16: Tabella e grafico della prima esecuzione di EKF con ostacolo

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	142.7	132.9	113.8	95.50	66.90
y	50.00	57.00	68.90	132.7	114.8	70.20	67.33

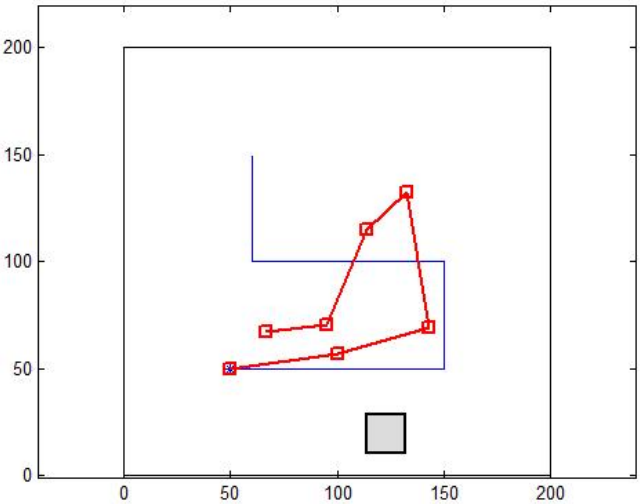


Figura A.17: Tabella e grafico della seconda esecuzione di EKF con ostacolo

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	144.7	148.9	99.20	118.8	129.8
y	50.00	57.00	50.60	89.60	85.60	115.8	64.60

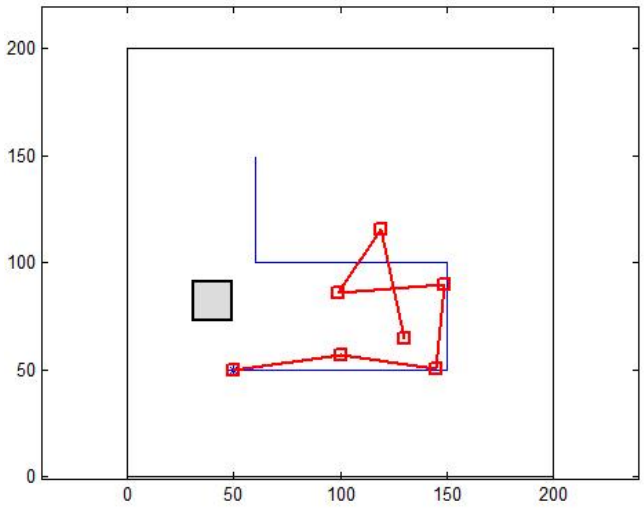


Figura A.18: Tabella e grafico della prima esecuzione di UKF con ostacolo

	P_1	$P^1_{1,2}$	P_2	P_3	$P^1_{3,4}$	P_4	P_5
x	50.00	100.0	144.0	145.0	95.80	94.50	124.1
y	50.00	57.00	52.60	90.40	87.80	126.5	109.4

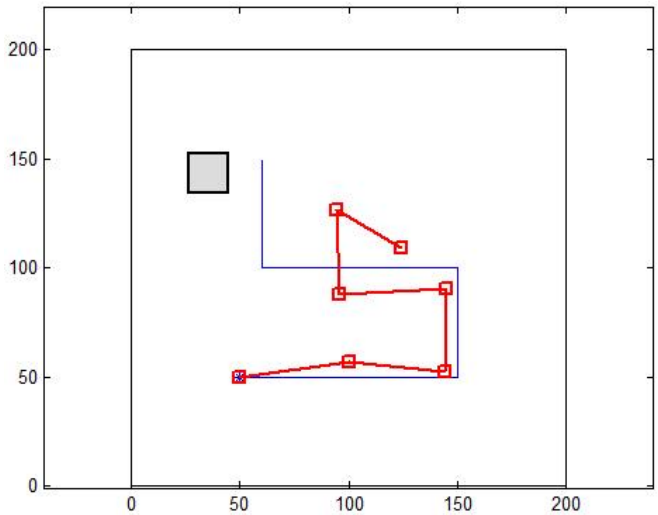


Figura A.19: Tabella e grafico della seconda esecuzione di UKF con ostacolo

Appendice B
Listati dei programmi realizzati

Programma per il controllo dei motori

```
// Definizione Pin Arduino
#define m1 9                //Motore destro           nero+blu
#define m2 10               //Motore sinistro         rosso+blu
#define encoder1 18         //Encoder motore destro   verde
#define encoder2 4          //Encoder motore destro   verde+
   bianco
#define encoder3 19         //Encoder motore sinistro arancione
#define encoder4 7          //Encoder motore sinistro bianco+
   giallo
#define ir1 2               //Infrarosso 1 (45ř)  -> AnalogIn 2
#define ir2 6               //Infrarosso 2 (90ř)  -> AnalogIn 6
#define ir3 8               //Infrarosso 3 (135ř) -> AnalogIn 8
#define ir4 10              //Infrarosso 4 (180ř) -> AnalogIn 10
#define ir5 12              //Infrarosso 5 (225ř) -> AnalogIn 12
#define ir6 14              //Infrarosso 6 (270ř) -> AnalogIn 14
#define ir7 0               //Infrarosso 7 (315ř) -> AnalogIn 0
#define ir8 4               //Infrarosso 8 (360ř) -> AnalogIn 4
// Variabili lettura seriale
char s[25]={"A0A0q\0"};
char ct;
int i=0,j,k;
int inizio,segno;
// Variabili scrittura seriale
int var1=127, var2=127;                          //velocità
   motori
boolean posizione=false;                         //controllo
   posizione/velocità
boolean reading1, reading2, reading3, reading4;  //lettura
   encoder
boolean oldreading1, oldreading3;
int deltavel1=0,deltavel2=0,velrichiesta1=0,velrichiesta2=0;
boolean avanti;
// Variabili intervalli di tempo
long previousMillis = 0, previousMillisStampa=0;
```

```
#define interval 50          //Intervallo calcolo velocità e
   posizioni
#define intervalstampa 100        //Intervallo lettura stringa
// Variabili controllo velocità
long step1=0, step2=0, oldstep1=0, oldstep2=0;
int vel1=0, vel2=0;
// Variabili controllo posizione
int giririch1=0, giririch2=0, angolirich1=0, angolirich2=0, sat
   =0;
long steprich1=0, steprich2=0, stepiniz1=0, stepiniz2=0,
   relstep1=0, relstep2=0;
long deltaenc1=0, deltaenc2=0, intstep1=0, intstep2=0,
   olddeltaenc1=0, olddeltaenc2=0;
// Variabili controllo sensori
#define numSensori 8
int analogValueAverage[numSensori];
// Funzione per la frequenza del PWM
void setPwmFrequency(int pin, int divisor) {
  byte mode;
  if(pin == 5 || pin == 6 || pin == 9 || pin == 10) {
    switch(divisor) {
    case 1:
      mode = 0x01;
      break;
    case 8:
      mode = 0x02;
      break;
    case 64:
      mode = 0x03;
      break;
    case 256:
      mode = 0x04;
      break;
    case 1024:
      mode = 0x05;
      break;
    default:
      return;
    }
    if(pin == 5 || pin == 6) {
      TCCR0B = TCCR0B & 0b11111000 | mode;
    }
    else {
      TCCR1B = TCCR1B & 0b11111000 | mode;
    }
  }
  else if(pin == 3 || pin == 11) {
    switch(divisor) {
    case 1:
      mode = 0x01;
      break;
```

```
    case 8:
      mode = 0x02;
      break;
    case 32:
      mode = 0x03;
      break;
    case 64:
      mode = 0x04;
      break;
    case 128:
      mode = 0x05;
      break;
    case 256:
      mode = 0x06;
      break;
    case 1024:
      mode = 0x07;
      break;
    default:
      return;
    }
    TCCR2B = TCCR2B & 0b11111000 | mode;
  }
}

// Setup
void setup() {
  // Definizioni porte e BaudRate
  Serial.begin(19200);
  pinMode(encoder1,INPUT);
  pinMode(encoder2,INPUT);
  pinMode(encoder3,INPUT);
  pinMode(encoder4,INPUT);
  pinMode(ir1,INPUT);
  pinMode(ir2,INPUT);
  pinMode(ir3,INPUT);
  pinMode(ir4,INPUT);
  pinMode(ir5,INPUT);
  pinMode(ir6,INPUT);
  pinMode(ir7,INPUT);
  pinMode(ir8,INPUT);
  setPwmFrequency(m1, 8);
  setPwmFrequency(m2, 8);
  //Inizializzazione variabili
  reading1=digitalRead(encoder1);
  reading2=digitalRead(encoder2);
  reading3=digitalRead(encoder3);
  reading4=digitalRead(encoder4);
  attachInterrupt(5, LeggiEncoder1, RISING);
  attachInterrupt(4, LeggiEncoder2, RISING);
  previousMillisIR = millis();
```

```
}
// Loop
void loop(){
  // Lettura stringa
  if (millis() - previousMillisStringa > intervalStringa) {
    if (Serial.available() > 0) {
      ct=char(Serial.read());
      if (((ct-'0'>=0) && (ct-'0')<=9) || (ct-'A'>=0) && (s[j]-
          'A')<=26){
        s[i] = ct;
        i++;
      }
    }
    if (s[i-1]=='Q') {
      // Comando letture
      if(s[0]=='L'){
        //Stampa valori medi degli numSensori sensori
        for(int cont=0;cont<10;contIR+){
           analogValueAverage[0]+=analogRead(ir1);
           analogValueAverage[1]+=analogRead(ir2);
           analogValueAverage[2]+=analogRead(ir3);
           analogValueAverage[3]+=analogRead(ir4);
           analogValueAverage[4]+=analogRead(ir5);
           analogValueAverage[5]+=analogRead(ir6);
           analogValueAverage[6]+=analogRead(ir7);
           analogValueAverage[7]+=analogRead(ir8);
        }
        for(int contIR=0;contIR<NumSensori;contIR++){
           Serial.println(analogValueAverage[contIR]/10);
                        }
        //Stampa passi encoder attuali
        Serial.println(step1);
        Serial.println(step2);
      }
      // Assegnazione velocità
      else if (s[0]=='A' || s[0]=='D'){
        velrichiesta1=0;
        velrichiesta2=0;
        // Assegnazione motore destro
        if (s[0]=='A'){
          segno=-1;
          avanti=true;
        }
        else if (s[0]=='D'){
          segno=1;
          avanti=false;
        }
        for (j=1;(s[j]-'0'>=0) && (s[j]-'0')<=9 ;j++)
          velrichiesta1=velrichiesta1*10 + (s[j]-'0');
        velrichiesta1=velrichiesta1*segno;
        // Assegnazione motore sinistro
```

```
  if (s[j]=='A')
    segno=1;
  else if (s[j]=='D')
    segno=-1;
  for (k=j+1;(s[k]-'0'>=0) && (s[k]-'0')<=9 ;k++)
    velrichiesta2=velrichiesta2*10 + (s[k]-'0');
  velrichiesta2=velrichiesta2*segno;
  // Assegnazione controllo velocità
  posizione=false;
}
// Assegnazione posizioni
else if(s[0]=='G'){
  giririch1=0;
  giririch2=0;
  angolirich1=0;
  angolirich2=0;
  intstep1=0;
  intstep2=0;
  if (s[1]=='I'){
    segno=-1;
    inizio=2;
  }
  else {
    segno=1;
    inizio=1;
  }
  // Assegnazione giri motore destro
  for (j=inizio;(s[j]-'0'>=0) && (s[j]-'0')<=9 ;j++)
    giririch1=giririch1*10 + (s[j]-'0');
  giririch1=giririch1*segno;
  // Assegnazione giri motore sinistro
  if (s[j]=='G'){
    if (s[j+1]=='I'){
      segno=1;
      inizio=j+2;
    }
    else{
      segno=-1;
      inizio=j+1;
    }
    for (k=inizio;(s[k]-'0'>=0) && (s[k]-'0')<=9 ;k++){
        giririch2=giririch2*10 + (s[k]-'0');
    }
    giririch2=giririch2*segno;
    // Assegnazione angolo motore destro
    if (s[k]=='A'){
      if (s[k+1]=='I'){
        segno=-1;
        inizio=k+2;
      }
      else {
```

```
            segno=1;
            inizio=k+1;
          }
          for (j=inizio;(s[j]-'0'>=0) && (s[j]-'0')<=9 ;j++){
            angolirich1=angolirich1*10 + (s[j]-'0');
          }
          angolirich1=angolirich1*segno;
          // Assegnazione angolo motore sinistro
          if (s[j]=='A'){
            if (s[j+1]=='I'){
              segno=1;
              inizio=j+2;
            }
            else {
              segno=-1;
              inizio=j+1;
            }
            for (k=inizio;(s[k]-'0'>=0) && (s[k]-'0')<=9 ;k
                ++)
            angolirich2=angolirich2*10 + (s[k]-'0');
            angolirich2=angolirich2*segno;
          }
        }
      }
      // Conversione di giri e angolo in passi encoder
      steprich1=giririch1*3072+angolirich1/0.1171875;
      steprich2=giririch2*3072+angolirich2/0.1171875;
      stepiniz1=step1;
      stepiniz2=step2;
      // Assegnazione controllo posizione
      posizione=true;
    }
    i=0;
  }
  previousMillis6 = millis();
}
// Calcolo delle velocità
if (millis() - previousMillis > interval) {
  // Trasformazione in passi encoder al secondo
  vel1=(-(step1-oldstep1)*20)*0.1171875;
  vel2=((step2-oldstep2)*20)*0.1171875;
  // Memorizzazione per il successivo campionamento
  oldstep1=step1;
  oldstep2=step2;
  // Calcolo dell'errore
  deltavel1=(velrichiesta1-vel1);
  deltavel2=(velrichiesta2-vel2);
  // Legge di controllo delle posizioni (giri e angolo)
  if(posizione==true){
    relstep1=-(stepiniz1-step1);
    relstep2=(stepiniz2-step2);
```

```
    deltaenc1=steprich1-relstep1;
    deltaenc2=steprich2-relstep2;
    // Integrale
    intstep1+=deltaenc1*1;
    intstep2+=deltaenc2*1;
    // Scarica dell'integrale
    if((deltaenc1>0 && olddeltaenc1<0)||(deltaenc1<0 &&
       olddeltaenc1>0)){
      intstep1=0;
    }
    if((deltaenc2>0 && olddeltaenc2<0)||(deltaenc2<0 &&
       olddeltaenc2>0)){
      intstep2=0;
    }
    olddeltaenc1=deltaenc1;
    olddeltaenc2=deltaenc2;
    // Desaturazione dell'integrale
    sat=127;
    if(intstep1>sat)
      intstep1=sat;
    else if(intstep1<-sat)
      intstep1=-sat;
    if(intstep2>sat)
      intstep2=sat;
    else if(intstep2<-sat)
      intstep2=-sat;
    // Assegnazione ai motori
    var1=127+deltaenc1*0.096+intstep1*0.05;
    var2=127+deltaenc2*0.096+intstep2*0.05;
    // Saturazione uscita
    if(var1>190)
      var1=190;
    else if(var1<77)
      var1=77;
    if(var2>190)
      var2=190;
    else if(var2<77)
      var2=77;
  }
  // Legge di controllo della velocità
  else{
    // Controllo proporzionale delle velocità
    var1=var1-deltavel1*0.17;
    var2=var2-deltavel2*0.17;
    // Assegnazione nel caso di velocità richiesta nulla
    if (velrichiesta1==0)
      var1=127;
    if (velrichiesta2==0)
      var2=127;
    // Assegnazione nel caso di velocità maggiori di quelle
       massime
```

```
      if(var1>255)
        var1=255;
      else if(var1<0)
        var1=0;
      if(var2>255)
        var2=255;
      else if(var2<0)
        var2=0;
     }
    previousMillis3 = millis();
  }
  // Assegnazione dei volt ai motori
  if (millis() - previousMillis2 > interval2) {
    analogWrite(m1,var1);
    analogWrite(m2,var2);
  }
} //fine loop
// Funzioni chiamate dall'AttachInterrupt
void LeggiEncoder1(){
  reading2=digitalRead(encoder2);
  if(reading2==1){
    step1=step1+1;
  }
  else if (reading2==0) {
    step1=step1-1;
  }
}
void LeggiEncoder2(){
  reading4=digitalRead(encoder4);
  if(reading4==0){
    step2++;
  }
  else {
    step2--;
  }
}
```

Interpolazione polinomiale

```
%% Memorizzazione analogValue da Arduino
% Impostazioni della seriale
s = serial('COM6');
OutputBufferSize = 1 + 3 + 1 + 3 + 1 + 3 + 1 + 3 + 1 + 1;
InputBufferSize = 6 + 1 + 6 + 1 + 3*NumSensoriLaser + 1*
   NumSensoriLaser;
set(s,'Terminator', 'LF');
set(s,'BaudRate', 19200);
set(s,'DataBits', 8);
set(s,'StopBits', 1);
```

```
set(s,'OutputBufferSize', OutputBufferSize);
set(s,'InputBufferSize', InputBufferSize);
%Apertura della seriale
fopen(s);
% Per i sensori a corta gittata totmisure=29
totmisure=29;
% Ciclo for per totmisure di letture seriali da fare
for indvalore=1:totmisure
% Il vettore A è aggiornato con la nuova lettura seriale
   con dei char
[A,count] = fread(s,512,'char');
% Il vettore A viene trasformato da char a int8
A=A-'0';
% Le variabili in uso nel ciclo vengono resettate allo
   stato originario
iesimo=1;
indvar=1;
primo = true;
primoelemento=0;
n=size(A);
totcampioni=0;
numcifre=0;
valoremedio=0;
varianza=0;
% Calcolo il numero di campioni contando il numero di \n
   ('\' - '0' = -35)
for w=1:n(1)
    if(A(w)==-35)
        totcampioni=totcampioni+1;
    end
end
% Scarto il primo elemento poiché potrebbe non essere
   letto dalla prima cifra
totcampioni = totcampioni-1;
% Indice del primo elemento con completezza certa (dopo il
    primo \n)
for w=1:n(1)
    if(A(w)==-35 && primo)
        primoelemento = w+2;
        primo = false;
    end
end
% Azzero il buffer
buffer=zeros(1,n(1));
% Azzero la raccolta di valori
```

```
valore=zeros(1,n(1));
% L'indice i parte dal primo elemento di cui è certa la
   completezza
i=primoelemento;
% Ciclo for su tutti i campioni trovati in  questa lettura
    seriale
for (iesimo=1:totcampioni)
    % Il ciclo si interrompe al valore letto negativo (\n)
        o alla fine di A
    while(A(i)>=0 && i<n(1))
        buffer(i)=A(i);
        numcifre=numcifre+1;
        i=i+1;
    end
    % Salto il carattere \ e il carattere n
    i=i+2;
    % Resetto le variabili
    voltefor=0;
    valore(iesimo)=0;
    % Ciclo che imposta in valore il numero in decimale
    for k=i-numcifre-2:i-3
        voltefor=voltefor+1;
        valore(iesimo)=valore(iesimo)+buffer(k)*10^(
           numcifre-voltefor);
    end
    % Riporto k all'ultima cifra letta
    k=k-1;
    % Memorizzo per calcolare il valore medio
    valoremedio=valoremedio+valore(iesimo);
    iesimo=iesimo+1;
    numcifre=0;
end
% Calcolo il valore medio
valoremedio=valoremedio/totcampioni;
% Memorizzo per calcolare la varianza
for indvar=1:totcampioni
    varianza= varianza + (valore(indvar)-valoremedio)^2;
end
% Calcolo la varianza
varianza=varianza/(totcampioni-1);
varianza = sqrt(varianza);
% Memorizzo la varianza
DeviazioneStandard(indvalore)=varianza;
% Memorizzo il valore analogico medio
analogValue(indvalore)=valoremedio;
```

```
% Attendo un comando da utente per leggere i valori
   successivi
pause
end

%% Calcolo funzione di trasferimento
% Totale delle misurazioni effettuate: da 10 a 80 cm ogni
   5 cm
totmisure=29;
% Trasformo in volt il valore letto sulla porta analogica
volt=5.0*analogValue/1024.0;
% Grado dell'approssimazione
grado=2;
% Azzero le variabili
y=0;
valorecm(j)=zeros(1,totmisure);
% riempimento array cm per misure a corta gittata (10-80):
    totmisure=29
for i=1:totmisure
          cm(i)=7.5+2.5*i;    % 10; 12.5; 15; 17.5; 20;
             ...; 77.5; 80
end
% Coefficienti dell'interpolazione polinomiale
coefficienti=vpa(polyfit(volt,cm,grado),8);

%% Chiusura seriale
fclose(s);
```

Conversione da Volt a cm

```
function [range]=VoltageToRange(voltage)
     % Metà inferiore della curva
     if(voltage < 0.8181650)
       if(voltage<0.5860420)
         range = -1.431295831 * voltage + 1.371201638;
       else
         range = -0.751946517 * voltage + 0.973074437;
       end
     % Metà superiore della curva
     else
       if(voltage < 1.1492463)
         range = -0.349412673 * voltage + 0.643735317;
       elseif(voltage < 1.6363301)
         range = -0.177086187 * voltage + 0.445689738;

```

```
        else
          range = -0.076191604 * voltage + 0.280592896;
        end
      end
      % Conversione in cm
      range = range*100;
end
```

Media dei risultati su 100 prove

```
%% 100 simulazioni per ogni posizionamento
sommaEKF=0;
sommaUKF=0;
Nfig=1;
Nsimulazioni=100;

% 100 simulazioni memorizzate nell'array
for i=1:Nsimulazioni
    % Lancia stanza e percorso in esame
    dati_P
    % Lancia il filtro di Kalman esteso
    EKFSGL
    % Salva il valore di J_EKFsgl
    EKF(i)=J_EKFsgl;
    % Lancia il filtro di Kalman unscented
    UKFsgl
    % Salva il valore di J_UKFsgl
    UKF(i)=J_UKFsgl;
end

% Somma dei 100 elementi dell'array
for j=1:Nsimulazioni
    sommaEKF=sommaEKF+EKF(j);
    sommaUKF=sommaUKF+UKF(j);
end

% Media dei 100 elementi dell'array
mediaEKF=sommaEKF/Nsimulazioni;
mediaUKF=sommaUKF/Nsimulazioni;

% Media degli elementi scartando il valore più grande
sommaEKF=sommaEKF-max(EKF);
sommaUKF=sommaUKF-max(UKF);

mediaEKFscarto=sommaEKF/(Nsimulazioni-1);
```

```
mediaUKFscarto=sommaUKF/(Nsimulazioni-1);

% istogrammi per verificare la validità degli scarti
figure(Nfig);
Nfig=Nfig+1;
hist(EKF,50)

figure(Nfig);
Nfig=Nfig+1;
hist(UKF,50)
```

Implementazione della stanza e del percorso

Il seguente algoritmo è quello utilizzato per la seconda stanza implementata, e dove è presente il quarto tipo di posizionamento dei sensore lungo la circonferenza del robot.

```
% Variabili globali e locali
global numSegmenti n q RaggioCirconfRobot numSensoriIR M
   thetaIRHome tol infinito xMin xMax yMin yMax Dx Dy
tol = 0.001; % tolleranza nei confronti di uguaglianza
Infinito = 9999; % infinito pratico

%% Dati ambiente
numOstacoli = 0; % numero degli ostacoli presenti nell'
   ambiente (per ora non previsto)
Px(1) = 0; Px(2) = 14.9; Px(3) = 15.1; Px(4) = 24.9; Px(5)
    = 25.1; Px(6) = 40; Px(7) = 50;
Py(1) = 0; Py(2) = 6.9; Py(3) = 7.1; Py(4) = 10.0; Py(5) =
    15.0; Py(6) = 25;
xMin = min(Px);
xMax = max(Px);
yMin = min(Py);
yMax = max(Py);
Dx = xMax - xMin;
Dy = yMax - yMin;
numVertici = 18;
P=zeros(numVertici,2);
P(1,:)=[Px(1) Py(1)];
P(2,:)=[Px(2) Py(1)];
P(3,:)=[Px(2) Py(4)];
P(4,:)=[Px(3) Py(4)];
```

```
P(5,:)=[Px(3) Py(1)];
P(6,:)=[Px(7) Py(1)];
P(7,:)=[Px(7) Py(6)];
P(8,:)=[Px(5) Py(6)];
P(9,:)=[Px(5) Py(3)];
P(10,:)=[Px(6) Py(3)];
P(11,:)=[Px(6) Py(2)];
P(12,:)=[Px(4) Py(2)];
P(13,:)=[Px(4) Py(6)];
P(14,:)=[Px(3) Py(6)];
P(15,:)=[Px(3) Py(5)];
P(16,:)=[Px(2) Py(5)];
P(17,:)=[Px(2) Py(6)];
P(18,:)=[Px(1) Py(6)];
for indVertice = 1:numVertici,
    [P(indVertice,1),P(indVertice,2)] = ruota(P(indVertice
       ,1),P(indVertice,2));
end
M = zeros(numVertici,4);
for indVertice = 1:numVertici-1,
    M(indVertice,:) = [P(indVertice,:) P(indVertice+1,:)];
end
M(numVertici,:) = [P(numVertici,:) P(1,:)];
numSegmenti = numVertici; % numero dei lati del perimetro
   dell'ambiente
n = ones (1,numSegmenti); % coefficienti angolari rette
q = ones (1,numSegmenti); % intercette rette
for lato = 1:numSegmenti,
   P1x = M(lato,1);
   P1y = M(lato,2);
   P2x = M(lato,3);
   P2y = M(lato,4);
   if P1x==P2x,
       n(lato) = Inf;
       q(lato) = P1x;
   else
       n(lato) = (P2y-P1y)/(P2x-P1x); % coefficiente
          angolare lato
       q(lato) = P1y - n(lato)*P1x; % costante lato
   end
end

%% Dati Robot
% Caratteristiche geometriche del robot
numSensoriIR = 8;
```

```
RaggioCirconfRobot = 0.1;
thetaIRHome = zeros(numSensoriIR,1);
%Posizionamento 4
thetaIRHome= thetaIRHome + [-pi/2 -pi/6 0 pi/6 pi/2 5*pi/6
    pi 7*pi/6]';

%% Dati simulazione
Npassi = 5000;
Nstep = 100;
Rx(1) = 5; Ry(1) = 12.5;
Rx(2) = 20; Ry(2) = 12.5;
Rx(3) = 20; Ry(3) = 3.5;
Rx(4) = 45; Ry(4) = 3.5;
Rx(5) = 45; Ry(5) = 12.5;
numVertR = numel(Rx);
for indVertR = 1:numVertR,
    [Rx(indVertR),Ry(indVertR)] = ruota(Rx(indVertR),Ry(
       indVertR));
end
x0 = Rx(1);
y0 = Ry(1);
theta0 = atan2(Ry(2)-Ry(1),Rx(2)-Rx(1));
deltaRho = zeros(1,Npassi);
deltaTheta = zeros(1,Npassi);
d1 = sqrt((Rx(2)-Rx(1))^2+(Ry(2)-Ry(1))^2);
d2 = sqrt((Rx(3)-Rx(2))^2+(Ry(3)-Ry(2))^2);
d3 = sqrt((Rx(4)-Rx(3))^2+(Ry(4)-Ry(3))^2);
d4 = sqrt((Rx(5)-Rx(4))^2+(Ry(5)-Ry(4))^2);
dTOT = d1+d2+d3+d4;
passiTratto1 = round(Npassi*d1/dTOT);
passiTratto2 = round(Npassi*(d1+d2)/dTOT)-passiTratto1;
passiTratto3 = round(Npassi*(d1+d2+d3)/dTOT)-passiTratto1-
   passiTratto2;
passiTratto4 = Npassi - passiTratto1-passiTratto2 -
   passiTratto3;
rhoC = 0.05*1000/Npassi;
nC = 40*Npassi/1000;
% Primo tratto
deltaRho=d1*ones(1,passiTratto1-nC/2)/(passiTratto1); %
   variazioni radiali
deltaTheta=ones(1,passiTratto1-nC/2)*0; % angolo costante
   che non varia nel tempo
% Curva
variazione=Ry(3)-Ry(2);
if variazione>=0
```

```
    segno=+1;
else segno=-1;
end
deltaRho = [deltaRho rhoC*ones(1,nC)];
deltaTheta = [deltaTheta (segno*pi/2)/nC*ones(1,nC)]; %
   rotazione di +/- 90 gradi
% Secondo tratto
deltaRho=[deltaRho d2*ones(1,passiTratto2-nC)/(
   passiTratto2)]; % variazioni radiali
deltaTheta=[deltaTheta ones(1,passiTratto2-nC)*0]; %
   angolo costante che non varia nel tempo
% Curva
variazione=Ry(4)-Ry(3);
if variazione>=0
    segno=+1;
else segno=-1;
end
deltaRho = [deltaRho rhoC*ones(1,nC)];
deltaTheta = [deltaTheta (segno*pi/2)/nC*ones(1,nC)]; %
   rotazione di +/- 90 gradi
% Terzo tratto
deltaRho=[deltaRho d3*ones(1,passiTratto3-nC)/(
   passiTratto3)]; % variazioni radiali
deltaTheta=[deltaTheta ones(1,passiTratto3-nC)*0]; %
   angolo costante che non varia nel tempo
% Curva
deltaRho = [deltaRho rhoC*ones(1,nC)];
angSucc = atan2(Ry(5)-Ry(4),Rx(5)-Rx(4));
angPrec = atan2(Ry(4)-Ry(3),Rx(4)-Rx(3));
angRot = angSucc - angPrec;
deltaTheta = [deltaTheta angRot/nC*ones(1,nC)]; %
   rotazione di angRot gradi
% Quarto tratto
deltaRho=[deltaRho d4*ones(1,passiTratto4-nC/2)/(
   passiTratto4)]; % variazioni radiali
deltaTheta=[deltaTheta ones(1,passiTratto4-nC/2)*0]; %
   angolo costante che non varia nel tempo
% Definizione dei vettori x,y e theta delle coordinate del
    robot durante la simulazione:
x = zeros(1,Npassi);
y = zeros(1,Npassi);
theta = zeros(1,Npassi);
% Assegnazione condizione iniziale assunta dal robot:
x(1)=x0;
y(1)=y0;
```

```
theta(1)=theta0;
% Deviazione standard degli errori della stima della
   posizione iniziale
sigmaX0 = 0.01;
sigmaY0 = 0.01;
sigmaTheta0 = 0.02;
% Deviazione standard del rumore sulla dinamica
Krho = 0.01;
Ktheta = 0.02;
% Errore stima iniziale
DeltaX0 = sigmaX0*randn(1);
DeltaY0 = sigmaY0*randn(1);
DeltaTheta0 = sigmaTheta0*randn(1);
x(1)=x0+DeltaX0;
y(1)=y0+DeltaY0;
theta(1)=theta0+DeltaTheta0;
% Assegnazione di tutte le posizioni assunte dal robot:
for k=1: (Npassi-1),
    x(k+1)= x(k)+ deltaRho(k)*cos(theta(k));
    y(k+1)= y(k)+ deltaRho(k)*sin(theta(k));
    theta(k+1)= theta(k)+ deltaTheta(k);
end
NiRho = randn(1,Npassi)*sqrt(Krho);
NiTheta = randn(1,Npassi)*sqrt(Ktheta);
% Letture odometriche corrotte
deltaRhoe = zeros(1,Npassi);
deltaThetae = zeros(1,Npassi);
for k = 1:Npassi,
    deltaRhoe(k) = deltaRho(k) + sqrt(deltaRho(k))*NiRho(k
       );
    deltaThetae(k) = deltaTheta(k) + sqrt(deltaRho(k))*
       NiTheta(k);
end
% Letture acustiche corrotte
ZitaIR = 0.1;
IRNoise = sqrt(ZitaIR)*randn(numSensoriIR,Npassi);
```

Algoritmo EKF

```
% Vettori di stima
xHatEKFsgl = zeros(1,Npassi);
xHatMenoEKFsgl = zeros(1,Npassi);
yHatEKFsgl = zeros(1,Npassi);
yHatMenoEKFsgl = zeros(1,Npassi);
thetaHatEKFsgl = zeros(1,Npassi);
```

```
thetaHatMenoEKFsgl = zeros(1,Npassi);
% Stima della posizione iniziale assunta dal robot
xHatEKFsgl(1) = x0 + DeltaX0;
yHatEKFsgl(1) = y0 + DeltaY0;
thetaHatEKFsgl(1) = theta0 + DeltaTheta0;
% Matrice di covarianza sulla stima iniziale
Pcov = diag([sigmaX0^2, sigmaY0^2, sigmaTheta0^2]);
numeroStime = 0;
% Matrice di covarianza del rumore sulla dinamica (NiRho e
    NiTheta)
Q = [Krho 0; 0 Ktheta];
% Matrice covarianza misure
R = ZitaIR*eye(numSensoriIR);
Jpepe_EKFsgl = 0;
for k = 1:Npassi-1,

%% Fase di predizione (fatta per tutti i k)
    xHatMenoEKFsgl(k+1) = xHatEKFsgl(k) + deltaRhoe(k)*cos
       (thetaHatEKFsgl(k)); % xHat(k+1|k)
    yHatMenoEKFsgl(k+1) = yHatEKFsgl(k) + deltaRhoe(k)*sin
       (thetaHatEKFsgl(k)); % yHat(k+1|k)
    thetaHatMenoEKFsgl(k+1) = thetaHatEKFsgl(k) +
       deltaThetae(k); % thetaHat(k+1|k)
    % jacobiano della dinamica rispetto allo stato
    F = [1 0 -deltaRhoe(k)*sin(thetaHatEKFsgl(k));
        0 1 deltaRhoe(k)*cos(thetaHatEKFsgl(k));
        0 0 1];
    % jacobiano della dinamica rispetto al disturbo
    W = [sqrt(deltaRhoe(k))*cos(thetaHatEKFsgl(k)) 0;
        sqrt(deltaRhoe(k))*sin(thetaHatEKFsgl(k)) 0;
        0 sqrt(deltaRhoe(k))];
    % Aggiornamento matrice di covarianza
    Pmeno = F*Pcov*F' + W*Q*W';

%% Fase di stima (fatta solo ogni Nstep passi)
    if mod(k,Nstep) == 0, % va fatta la stima con Kalman
       Esteso
       numeroStime = numeroStime + 1;
       % distanze IR corrotte (reali)
       [latiLetti,distanzeLette] = distanzeIR(x(k+1),y(k
          +1),theta(k+1));
       rhoiReali = distanzeLette + IRNoise(:,k); % era
          numeroStime
       % jacobiano H delle misure buone
       H = zeros(numSensoriIR,3);
```

```
45          ammessi = ones(numSensoriIR,1);
            [latiAttesi, rhoiAttese] = distanzeIR(
               xHatMenoEKFsgl(k+1),yHatMenoEKFsgl(k+1),
               thetaHatMenoEKFsgl(k+1)); % calcolo prima i rhoi
                e i lati con cui si intersecano gli IR
            if min(latiAttesi)==0, % stima fuori mappa
                [latiAttesi1,rhoiAttese1,latiAttesi,rhoiAttese]
                    = distanzeIRRob(xHatMenoEKFsgl(k+1),
                   yHatMenoEKFsgl(k+1),thetaHatMenoEKFsgl(k+1))
                   ;
                for IR = 1:numSensoriIR,
50                  if latiAttesi1(IR) == 0,
                        ammessi(IR)=0;
                    end
                end
            end
55          innovation = rhoiReali-rhoiAttese;

            xIR = xHatMenoEKFsgl(k+1);   % La x del robot
               stimata
            yIR = yHatMenoEKFsgl(k+1);   % La y del robot
               stimata
            for IR = 1: numSensoriIR,  % per ogni IR
60              if latiAttesi(IR)>0 && abs(innovation(IR))/max
                   ([rhoiReali(IR),rhoiAttese(IR)]) < 0.4, %
                   allora l'innovazione e' accettabile
                    nIR = n(latiAttesi(IR));     % determino il
                        coeff. angolare del lato con cui si
                       interseca l'IR
                    qIR = q(latiAttesi(IR));     % e l'
                       intercetta di tale lato (qi,j nelle
                       formule)
                    angIR = thetaHatMenoEKFsgl(k+1)+thetaIRHome
                       (IR);   % l'angolo del raggio IR (
                       theta_t + theta_i nelle formule)
                    cosIR = cos(angIR);    % il coseno di tale
                       angolo
65                  sinIR = sin(angIR);    % e il seno
                    if nIR == Inf, % la retta con cui si
                       interseca l'IR e' verticale
                        if abs(cosIR) < tol, % il raggio IR e'
                           paralello al lato
                            H(IR,:) = [Inf 0 Inf];
                        else                    % il raggio IR
                           NON e' parallelo al lato
```

```
                    H(IR,:) = [-1/cosIR 0 rhoiAttese(IR
                        )*sinIR/cosIR];
                end
            else  % retta obliqua (ossia non verticale)
                DEN = nIR*cosIR-sinIR;
                if abs(DEN) < 0.0001, % raggio IR
                    parallelo a lato
                    H(IR,:) = [Inf Inf Inf];
                else                  % il raggio IR
                    NON e' paralello al lato
                    H(IR,1) = -nIR/DEN;
                    H(IR,2) = 1/DEN;
                    H(IR,3) = -(nIR*xIR+qIR-yIR)*(cosIR
                        + nIR*sinIR)/DEN^2;
                end
            end
        else
            ammessi(IR) = 0;
        end
    end
    ammessiIR = sum(ammessi);
    H = riduciM(H,ammessi);
    innovation = riduciM(innovation,ammessi);
    R = ZitaIR*eye(ammessiIR);
    % Calcolo guadagno Kalman K
    K = Pmeno*H'*pinv(H*Pmeno*H'+R);
    % Calcolo della stima a posteriori:
    xHatEKFsgl(k+1) = xHatMenoEKFsgl(k+1) + K(1,:)*
        innovation;
    yHatEKFsgl(k+1) = yHatMenoEKFsgl(k+1) + K(2,:)*
        innovation;
    thetaHatEKFsgl(k+1) = thetaHatMenoEKFsgl(k+1) + K
        (3,:)*innovation;
    % Aggiornamento matrice di covarianza
    Pcov = (eye(3) - K*H)*Pmeno;
    Jpepe_EKFsgl = Jpepe_EKFsgl + (x(k+1)-xHatEKFsgl(k
        +1)).^2+(y(k+1)-yHatEKFsgl(k+1)).^2;
else    % Non si fa la stima con EKF ma si utilizza
    solo odometria
    xHatEKFsgl(k+1) = xHatMenoEKFsgl(k+1);
    yHatEKFsgl(k+1) = yHatMenoEKFsgl(k+1);
    thetaHatEKFsgl(k+1) = thetaHatMenoEKFsgl(k+1);
    % Aggiornamento matrice di covarianza
    Pcov = Pmeno;
end
```

```
end
% prestazione normalizzata
J_EKFsgl = sqrt(sum((x-xHatEKFsgl).^2+(y-yHatEKFsgl).^2)/
   Npassi) ;
```

Filtro EKF per la movimentazione del robot rover

```
%% Variabii e dati
% Dati seriale
s = serial('COM6');
OutputBufferSize = 1 + 3 + 1 + 3 + 1 + 3 + 1 + 3 + 1 + 1;
InputBufferSize = 6 + 1 + 6 + 1 + 3*NumSensoriLaser + 1*
   NumSensoriLaser;
set(s,'Terminator', 'LF');
set(s,'BaudRate', 19200);
set(s,'DataBits', 8);
set(s,'StopBits', 1);
set(s,'OutputBufferSize', OutputBufferSize);
set(s,'InputBufferSize',InputBufferSize);
second=1.5;
% Vettori di stima
xHatEKFsgl = zeros(1,Npassi);
xHatMenoEKFsgl = zeros(1,Npassi);
yHatEKFsgl = zeros(1,Npassi);
yHatMenoEKFsgl = zeros(1,Npassi);
thetaHatEKFsgl = zeros(1,Npassi);
thetaHatMenoEKFsgl = zeros(1,Npassi);
% Stima della posizione iniziale assunta dal robot
xHatEKFsgl(1) = x0 + DeltaX0;
yHatEKFsgl(1) = y0 + DeltaY0;
thetaHatEKFsgl(1) = theta0 + DeltaTheta0;
% Matrice di covarianza sulla stima iniziale
Pcov = diag([sigmaX0^2, sigmaY0^2, sigmaTheta0^2]);
% Conta il numero di volte che è stata effettuata la stima
    con la correzione
numeroStime = 0;
% Matrice di covarianza del rumore sulla dinamica (NiRho e
    NiTheta)
Q = [Krho 0; 0 Ktheta];
% Matrice covarianza misure
R = ZitaIR*eye(numSensoriIR);
% Vettore letture odometriche traslazionali e di
   orientamento
deltaRhoe = zeros(1,Npassi);
deltaThetae = zeros(1,Npassi);
```

```
35
%% Apertura seriale e inizializzazione
% Apertura della porta seriale
fopen(s);
pause(second);
40 % Attende un comando da tastiera per cominciare
pause
% Lettura dei sensori e degli encoder iniziali
[centimetriprec,encoderprec,status]=leggiSeriale(s,'LQ');
% Se la lettura è andata a buon fine
45 if status==1
    % Assegnazione prima posizione
    x=deltaX(1);
    y=deltaY(1);
    theta=deltaTheta(1);
50  % Generazione delle stringhe di movimentazione del
       robot
    [esistediagonale,esistetheta,stringaD,stringaTheta,
       tempoD,tempoTheta]=MuoviRobot(x,y,theta);
    % Se è stato generato un comando
    if(esistediagonale||esistetheta)
        % Se è stato generato un comando di rotazione
55      if esistetheta
            % Scrive l'angolo sulla seriale e attende un
               tempo sufficiente per la rotazione
            fwrite(s,stringaTheta);
            pause(tempoTheta)
            % Ferma i motori
60          fwrite(s,'A0A0Q');
            pause(second)
        end
        % Se è stato generato un comando di traslazione
        if esistediagonale
65          % Scrive l'angolo sulla seriale e attende un
               tempo sufficiente per la traslazione
            fwrite(s,stringaD);
            pause(tempoD)
            % Ferma i motori
            fwrite(s,'A0A0Q');
70          pause(second)
        end
    end
end
```

```
%% Continuo della movimentazione per tutti gli Npassi
   rimanenti
for k = 2:Npassi-1,
   % Lettura dei sensori e degli encoder attuali
   [centimetri,encoder,status]=leggiSeriale(s,'LQ');
   % Se la lettura è andata a buon fine
   if status==1
        % Calcolo i passi encoder da fare (per ogni motore
           )
        passi1 = encoder(1)-encoderprec(1);
        passi2 = encoder(2)-encoderprec(2);
        % Converto i passi encoder in centimetri (per ogni
            motore)
        spazioeffettuato1=passi1*28/3072;
        spazioeffettuato2=passi2*28/3072;
        % Calcolo della stima del percorso fatto leggendo
           i soli encoder
        [posfinale,thetafinale]=CorreggiPosizione(deltaRho
           (k), deltaTheta(k), spazioeffettuato1,
           spazioeffettuato2, deltaRho(k-1), deltaTheta(k
           -1));
        % Fase di predizione
        deltaRhoe(k) = deltaRho(k) + sqrt(min(
           spazioeffettuato1,spazioeffettuato2))*NiRho(k);
        deltaThetae(k) = deltaTheta(k) + sqrt(min(
           spazioeffettuato1,spazioeffettuato2))*NiTheta(k
           );
        xHatMenoEKFsgl(k+1) = xHatEKFsgl(k) + deltaRhoe(k)
           *cos(thetaHatEKFsgl(k));
        yHatMenoEKFsgl(k+1) = yHatEKFsgl(k) + deltaRhoe(k)
           *sin(thetaHatEKFsgl(k));
        thetaHatMenoEKFsgl(k+1) = thetaHatEKFsgl(k) +
           deltaThetae(k);
        % Jacobiano della dinamica rispetto allo stato
        F = [1 0 -deltaRhoe(k)*sin(thetaHatEKFsgl(k));
            0 1 deltaRhoe(k)*cos(thetaHatEKFsgl(k));
            0 0 1];
        % Jacobiano della dinamica rispetto al disturbo
        W = [sqrt(deltaRhoe(k))*cos(thetaHatEKFsgl(k)) 0;
            sqrt(deltaRhoe(k))*sin(thetaHatEKFsgl(k)) 0;
            0 sqrt(deltaRhoe(k))];
        % Aggiornamento matrice di covarianza
        Pmeno = F*Pcov*F' + W*Q*W';
        % Fase di stima
        numeroStime = numeroStime + 1;
```

```
H = zeros(numSensoriIR,3);
ammessi = ones(numSensoriIR,1);
[latiAttesi, rhoiAttese] = distanzeIR(
   xHatMenoEKFsgl(k+1),yHatMenoEKFsgl(k+1),
   thetaHatMenoEKFsgl(k+1)); % calcolo prima i
   rhoi e i lati con cui si intersecano gli IR
if min(latiAttesi)==0,
   [latiAttesi1,rhoiAttese1,latiAttesi,rhoiAttese]
       = distanzeIRRob(xHatMenoEKFsgl(k+1),
      yHatMenoEKFsgl(k+1),thetaHatMenoEKFsgl(k+1))
      ;
   for IR = 1:numSensoriIR,
       if latiAttesi1(IR) == 0,
             ammessi(IR)=0;
       end
   end
end
for sensore=1:numSensoriIR
    rhoiReali(sensore,1)=centimetri(sensore);
    if centimetri(sensore)<0
        rhoiReali(sensore,1)=rhoiAttese(sensore)+
           IRNoise(sensore,k);
    end
end
innovation = rhoiReali-rhoiAttese;
xIR = xHatMenoEKFsgl(k+1);
yIR = yHatMenoEKFsgl(k+1);
for IR = 1: numSensoriIR,
   if latiAttesi(IR)>0 && abs(innovation(IR))/max
      ([rhoiReali(IR),rhoiAttese(IR)]) < 0.4,
       nIR = n(latiAttesi(IR));
       qIR = q(latiAttesi(IR));
       angIR = thetaHatMenoEKFsgl(k+1)+thetaIRHome
          (IR);
       cosIR = cos(angIR);
       sinIR = sin(angIR);
       if nIR == Inf,
           if abs(cosIR) < tol,
               H(IR,:) = [Inf 0 Inf];
           else
               H(IR,:) = [-1/cosIR 0 rhoiAttese(IR
                  )*sinIR/cosIR];
           end
       else
           DEN = nIR*cosIR-sinIR;
```

```
                    if abs(DEN) < 0.0001,
                        H(IR,:) = [Inf Inf Inf];
                    else
                        H(IR,1) = -nIR/DEN;
                        H(IR,2) = 1/DEN;
                        H(IR,3) = -(nIR*xIR+qIR-yIR)*(cosIR
                            + nIR*sinIR)/DEN^2;
                    end
                end
            else
                ammessi(IR) = 0;
            end
        end
        ammessiIR = sum(ammessi);
        H = riduciM(H,ammessi);
        innovation = riduciM(innovation,ammessi);
        R = ZitaIR*eye(ammessiIR);
        % Calcolo guadagno Kalman K
        K = Pmeno*H'*pinv(H*Pmeno*H'+R);
        % Calcolo della stima a posteriori:
        xHatEKFsgl(k+1) = xHatMenoEKFsgl(k+1) + K(1,:)*
           innovation;
        yHatEKFsgl(k+1) = yHatMenoEKFsgl(k+1) + K(2,:)*
           innovation;
        thetaHatEKFsgl(k+1) = thetaHatMenoEKFsgl(k+1) + K
           (3,:)*innovation;
        % Aggiornamento matrice di covarianza
        Pcov = (eye(3) - K*H)*Pmeno;
        deltax=xHatEKFsgl(k+1)-xHatEKFsgl(k);
        deltay=yHatEKFsgl(k+1)-yHatEKFsgl(k);
        deltatheta=thetaHatEKFsgl(k+1)-thetaHatEKFsgl(k);
        % Calcolo stringa e comunicazione con Arduino
        [esistediagonale, esistetheta, stringaD,
           stringaTheta,tempoD, tempoTheta]=MuoviRobot(
           deltax,deltay,deltatheta);
        if(esistediagonale||esistetheta)
         if esistetheta
             fwrite(s,stringaTheta);
             pause(tempoTheta)
             fwrite(s,'A0A0Q');
             pause(second)
         end
         if esistediagonale
             fwrite(s,stringaD);
             pause(tempoD)
```

```
            fwrite(s,'AOAOQ');
            pause(second)
        end
        end
    % Aggiornamento matrice di covarianza e passi encoder
        Pcov = Pmeno;
        encoderprec(1)=encoder(1);
        encoderprec(2)=encoder(2);
  end

 end
% Indice di prestazione normalizzata
J_EKFsgl = sqrt(sum((x-xHatEKFsgl).^2+(y-yHatEKFsgl).^2)/
   Npassi) ;
% Chiusura della seriale
fclose(s);
```

Filtro UKF per la movimentazione del robot rover

```
%% Variabili e dati
% Dati seriale
s = serial('COM6');
OutputBufferSize = 1 + 3 + 1 + 3 + 1 + 3 + 1 + 3 + 1 + 1;
InputBufferSize = 6 + 1 + 6 + 1 + 3*NumSensoriLaser + 1*
   NumSensoriLaser;
set(s,'Terminator', 'LF');
set(s,'BaudRate', 19200);
set(s,'DataBits', 8);
set(s,'StopBits', 1);
set(s,'OutputBufferSize', OutputBufferSize);
set(s,'InputBufferSize',InputBufferSize);
second=1.5;
% Vettori di stima
xHatUKF = zeros(1,Npassi);
xHatMenoUKF = zeros(1,Npassi);
yHatUKF = zeros(1,Npassi);
yHatMenoUKF = zeros(1,Npassi);
thetaHatUKF = zeros(1,Npassi);
thetaHatMenoUKF = zeros(1,Npassi);
ennone = 3+2;
xaHat = zeros(ennone,1);
ZETA = zeros(numSensoriIR,2*ennone+1);
Pzeta = zeros(numSensoriIR);
Pxz = zeros(3,numSensoriIR);
% Stima della posizione iniziale assunta dal robot
```

```
xHatUKF(1) = x0+DeltaX0;
yHatUKF(1) = y0+DeltaY0;
thetaHatUKF(1) = theta0+DeltaTheta0;
% Matrice di covarianza sulla stima iniziale
Pcov = diag([sigmaX0^2, sigmaY0^2, sigmaTheta0^2]);
% Matrice di covarianza del rumore sulla dinamica (NiRho e
    NiTheta)
Q = [Krho 0; 0 Ktheta];
% Matrice covarianza misure
R = ZitaIR*eye(numSensoriIR);
% stato esteso per applicare UKF (ennone elementi, ennone
   = 3(stato) + 2
% (disturbo su dyn) + numSensoriIR (disturbo su misura)
xaHat(:) = [xHatUKF(1); yHatUKF(1); thetaHatUKF(1); zeros
   (2,1)];
Pa = zeros(ennone);
Pa(1:3,1:3) = Pcov;
Pa(4:5,4:5) = Q;
CSI = zeros(ennone,2*ennone+1);
CSImeno = zeros(3,2*ennone+1);
W = zeros(2*ennone+1,1);
% Vettori letture odometriche traslazionali e di
   orientamento
deltaRhoe = zeros(1,Npassi);
deltaThetae = zeros(1,Npassi);

%% Apertura seriale e inizializzazione
% Apertura della porta seriale
fopen(s);
pause(second);
% Attende un comando da tastiera per cominciare
pause
% Lettura dei sensori e degli encoder iniziali
[centimetriprec,encoderprec,status]=leggiSeriale(s,'LQ');
% Se la lettura è andata a buon fine
if status==1
    % Assegnazione prima posizione
    x=deltaX(1);
    y=deltaY(1);
    theta=deltaTheta(1);
    % Generazione delle stringhe di movimentazione del
       robot
    [esistediagonale, esistetheta, stringaD, stringaTheta,
       tempoD,tempoTheta]=MuoviRobot(x,y,theta);
    % Se è stato generato un comando
```

```
    if(esistediagonale||esistetheta)
        % Se è stato generato un comando di rotazione
        if esistetheta
            % Scrive l'angolo sulla seriale e attende un
               tempo sufficiente per la rotazione
            fwrite(s,stringaTheta);
            pause(tempoTheta)
            % Ferma i motori
            fwrite(s,'AOAOQ');
            pause(second)
        end
        % Se è stato generato un comando di traslazione
        if esistediagonale
            % Scrive l'angolo sulla seriale e attende un
               tempo sufficiente per la traslazione
            fwrite(s,stringaD);
            pause(tempoD)
            % Ferma i motori
            fwrite(s,'AOAOQ');
            pause(second)
        end
    end
end

%% Continuo della movimentazione per tutti gli Npassi
   rimanenti
for k = 2:Npassi-1,
    % Lettura dei sensori e degli encoder attuali
    [centimetri,encoder,status]=leggiSeriale(s,'LQ');
    % Se la lettura è andata a buon fine
    if status==1
        % Calcolo i passi encoder da fare (per ogni motore
           )
        passi1 = encoder(1)-encoderprec(1);
        passi2 = encoder(2)-encoderprec(2);
        % Converto i passi encoder in centimetri (per ogni
            motore)
        spazioeffettuato1=passi1*28/3072;
        spazioeffettuato2=passi2*28/3072;
        % Calcolo della stima del percorso fatto leggendo
           i soli encoder
        [posfinale,thetafinale]=CorreggiPosizione(deltaRho
           (k), deltaTheta(k), spazioeffettuato1,
           spazioeffettuato2, deltaRho(k-1), deltaTheta(k
           -1));
```

```
% Fase di predizione
deltaRhoe(k) = deltaRho(k) + sqrt(min(
   spazioeffettuato1,spazioeffettuato2))*NiRho(k);
    %encoders reading - letture odometriche
   traslazionali
for ind = 1:2*ennone+1,
    CSImeno(1,ind) = CSI(1,ind) + deltaRhoe(k)*cos
       (CSI(3,ind))+sqrt(abs(deltaRhoe(k)))*CSI(4,
       ind)*cos(CSI(3,ind)); % xHat(k+1|k)
    CSImeno(2,ind) = CSI(2,ind) + deltaRhoe(k)*sin
       (CSI(3,ind))+sqrt(abs(deltaRhoe(k)))*CSI(4,
       ind)*sin(CSI(3,ind)); % yHat(k+1|k)
    CSImeno(3,ind) = CSI(3,ind) + deltaThetae(k)+
       sqrt(abs(deltaRhoe(k)))*CSI(5,ind); %
       thetaHat(k+1|k)
end
% Stima valor medio
xHatMeno = CSImeno*W;
% Stima a priori
xHatMenoUKF(k+1) = xHatMeno(1);
yHatMenoUKF(k+1) = xHatMeno(2);
thetaHatMenoUKF(k+1) = xHatMeno(3);
% Aggiornamento matrice di covarianza
MatCOV = 0*Pcov;
for ind=1:2*ennone+1,
    MatCOV = MatCOV + W(ind)*(CSImeno(:,ind)-
       xHatMeno)*(CSImeno(:,ind)-xHatMeno)';
end
Pmeno = MatCOV;
% Fase di stima
numeroStime = numeroStime + 1;
ammessi = ones(numSensoriIR,1); % indica quali IR
   saranno considerati nella fase di stima
rhoiReali=centimetri;
ZETA = zeros(numSensoriIR,2*ennone+1);
% Distanze IR attese per ogni sigma point
for ind=1:2*ennone+1,
   [latiAttesi,rhoiAttese] = distanzeIR(CSImeno(1,
      ind),CSImeno(2,ind),CSImeno(3,ind));
   if min(latiAttesi)==0, % un sigma point fuori
      mappa
       [latiAttesi,rhoiAttese,latiAttesi2,
          rhoiAttese2] = distanzeIRRob(CSImeno(1,
          ind),CSImeno(2,ind),CSImeno(3,ind));
       for IR = 1:numSensoriIR,
```

```
                if latiAttesi(IR) == 0,
                    ammessi(IR)=0;
                end
            end
            ZETA(:,ind) = rhoiAttese2;
        else
            ZETA(:,ind) = rhoiAttese;
        end
    end
    for sensore=1:numSensoriIR
        rhoiReali(sensore,1)=centimetri(sensore);
        if centimetri(sensore)<0
            rhoiReali(sensore,1)=rhoiAttese(sensore)+
                IRNoise(sensore,k);
        end
    end
    rhoiReali
    % Distanza media attesa
    zHatMeno = ZETA*W;
    % Innovazione
    innovation = rhoiReali-zHatMeno;
    for IR = 1:numSensoriIR,
        if abs(innovation(IR))/max([rhoiReali(IR)
            zHatMeno(IR)])>0.4,
            ammessi(IR)=0;
        end
    end
    numAmmessi = sum(ammessi); % il numero di misure
       ammesse
    Pzeta = zeros(numAmmessi);
    Pxz = zeros(3,numAmmessi);
    ZETA = riduciM(ZETA,ammessi);
    zHatMeno = riduciM(zHatMeno,ammessi);
    R = ZitaIR*eye(numAmmessi);
    innovation = riduciM(innovation,ammessi);
    % Matrice covarianza Pzeta
    MatCOV = 0*Pzeta;
    for ind=1:2*ennone+1,
        MatCOV = MatCOV + W(ind)*(ZETA(:,ind)-zHatMeno
            )*(ZETA(:,ind)-zHatMeno)';
    end
    Pzeta = MatCOV + R;
    % Matrice covarianza mista Pxz
    MatCOV = 0*Pxz;
    for ind=1:2*ennone+1,
```

```
            MatCOV = MatCOV + W(ind)*(CSImeno(:,ind)-
                xHatMeno)*(ZETA(:,ind)-zHatMeno)';
        end
        Pxz = MatCOV;
        % Calcolo guadagno Kalman K
        K = Pxz*pinv(Pzeta);
        xHat = xHatMeno + K*innovation;
        % Aggiornamento matrice di covarianza
        Pcov = Pmeno - K*Pzeta*K';
        % Calcolo della stima a posteriori:
        xHatUKF(k+1) = xHat(1);
        yHatUKF(k+1) = xHat(2);
        thetaHatUKF(k+1) = xHat(3);
        xaHat = [xHat; 0; 0];
        Pa = 0*Pa;
        Pa(1:3,1:3) = Pcov;
        Pa(4:5,4:5) = Q;
        deltax=xHatUKF(k+1)-xHatUKF(k);
        deltay=yHatUKF(k+1)-yHatUKF(k);
        deltatheta=thetaHatUKF(k+1)-thetaHatUKF(k);
        % Calcolo stringa e comunicazione con Arduino
        [esistediagonale, esistetheta, stringaD,
            stringaTheta,tempoD, tempoTheta]=MuoviRobot(
            deltax,deltay,deltatheta);
        if(esistediagonale||esistetheta)
            if esistetheta
                fwrite(s,stringaTheta);
                pause(tempoTheta)
                fwrite(s,'A0A0Q');
                pause(second)
            end
            if esistediagonale
                fwrite(s,stringaD);
                pause(tempoD)
                fwrite(s,'A0A0Q');
                pause(second)
            end
        end
        encoderprec(1)=encoder(1);
        encoderprec(2)=encoder(2);
    end

end
deltaRhoe(Npassi) = deltaRho(Npassi) + sqrt(deltaRho(
    Npassi))*NiRho(Npassi); %encoders reading - letture
```

```
   odometriche traslazionali
deltaThetae(Npassi) = deltaTheta(Npassi) + sqrt(deltaRho(
   Npassi))*NiTheta(Npassi); %encoders reading - letture
   odometriche di orientamento
% Indice di prestazione normalizzata
J_UKFsgl = sqrt(sum((x-xHatUKF).^2+(y-yHatUKF).^2)/Npassi)
   ;
% Chiusura della seriale
fclose(s);
```

Funzione *SigmaPoints*

```
function [W,CSI] = sigmaPoints(x_m,P_x)
% Definizione e inizializzazione variabili
n = numel(x_m);
cappa = 0;
CSI=zeros(n,2*n+1);
W = zeros(2*n+1,1);
deltaCSI=zeros(n,2*n+1);
% Generazione SigmaPoints
CSI(:,2*n+1) = x_m;
W(2*n+1) = cappa/(cappa+n);
A = chol(P_x);
for ind=1:n,
    CSI(:,ind) = x_m + sqrt(n+cappa)*A(ind,:)';
    CSI(:,ind+n) = x_m - sqrt(n+cappa)*A(ind,:)';
    W(ind) = 1/(2*(n+cappa));
    W(ind+n) = 1/(2*(n+cappa));
end
% Controlli di correttezza
if abs(sum(W)-1)>0.001,
    disp('errore pesi');
    pause;
end
if sum(abs(CSI*W - x_m))>0.001,
    disp('errore media');
    pause;
end
% Generazione SigmaPoints
for ind=1:n,
    deltaCSI(:,ind) = CSI(:,ind) - x_m;
    deltaCSI(:,ind+n) = CSI(:,ind+n) - x_m;
end
COV = zeros(n);
for ind=1:2*n,
```

```
    COV = COV + W(ind)*deltaCSI(:,ind)*deltaCSI(:,ind)';
end
% Controllo correttezza
if sum(abs(COV-P_x))>0.001,
    disp('errore covarianza');
    pause;
end
```

Funzione *distanzeIR*

```
function [lati, distanze] = distanzeIR(Xt,Yt,Thetat)
% Variabili globali
global numSegmenti n q RaggioCirconfRobot numSensoriIR M
   thetaIRHome tol infinito
% Definizione array
letturaIR = zeros(1,numSensoriIR);
latoIR = zeros(1,numSensoriIR);
thetaIR = thetaIRHome + Thetat;
% Per ogni sensore
for IR = 1:numSensoriIR,
    % La lettura minima dal IR al perimetro
    letturaMin = Inf;
    latoMin = 0;
    % Coordinate del IR
    Xi = Xt + RaggioCirconfRobot*cos(thetaIR(IR));
    Yi = Yt + RaggioCirconfRobot*sin(thetaIR(IR));
    % Intersezione con i lati del raggio del IR
       considerato
    XpIR = zeros(1,numSegmenti);
    YpIR = zeros(1,numSegmenti);
    % Calcolo coordinate intersezione del raggio del IR
       con i vari lati
    for lato = 1:numSegmenti,
        if n(lato) < Inf,
            if abs(tan(thetaIR(IR)) - n(lato)) < 0.0001,
                XpIR(lato) = Inf;
                YpIR(lato) = Inf;
            else
                XpIR(lato) = (Xt*tan(thetaIR(IR))-Yt+q(
                   lato))/(tan(thetaIR(IR))-n(lato));
                YpIR(lato) = ((n(lato)*Xt+q(lato))*tan(
                   thetaIR(IR))-n(lato)*Yt)/(tan(thetaIR
                   (IR))-n(lato));
            end
        else
```

```
            XpIR(lato) = q(lato);
            if abs(cos(thetaIR(IR))) < 0.0001,
                YpIR(lato) = Inf;
            else
                YpIR(lato) = Yt + (q(lato)-Xt)*tan(thetaIR
                    (IR));
            end
        end
        % Verifica che l'intersezione appartiene al lato e
            giace nella parte giusta
        if (XpIR(lato) >= min(M(lato,1),M(lato,3))-tol) &&
            (XpIR(lato) <= max(M(lato,1),M(lato,3))+tol)
           && (YpIR(lato) >= min(M(lato,2),M(lato,4))-tol)
            && (YpIR(lato) <= max(M(lato,2),M(lato,4))+tol
           ),
            if (Xi >= min(XpIR(lato),Xt)-tol) && (Xi <=
               max(XpIR(lato),Xt)+tol) && (Yi >= min(YpIR(
               lato),Yt)-tol) && (Yi <= max(YpIR(lato),Yt)
               +tol),
                distIRLato =  sqrt((Xi-XpIR(lato))^2+(Yi-
                    YpIR(lato))^2);
                if distIRLato < letturaMin,
                    letturaMin = distIRLato;
                    latoMin = lato;
                end
            end
        end
    end
    % Si aggiorna con la distanza minima trovata sia la
       distanza che il lato corrispondente
    letturaIR(IR) = letturaMin;
    latoIR(IR) = latoMin;
end
lati = latoIR';
distanze = letturaIR' + RaggioCirconfRobot;
```

Funzione *MuoviRobot*

```
function [esistediagonale, esistetheta, stringaD,
   stringaTheta,tempoD,tempoTheta]=MuoviRobot(deltax,
   deltay,deltatheta)
% Inizializzazione variabili
temposetup=1;
tempoungiro=2;
diagonale=sqrt(deltax^2+deltay^2);
```

```
esistediagonale=0;
esistetheta=0;
stringaD='';
stringaTheta='';
tempoD=0;
tempoTheta=0;
% Se la diagonale da effettuare è abbastanza grande
 if abs(diagonale)>3
    % Calcolo del numero di angoli giro da effettuare
    esistediagonale=1;
    numerogirid=floor(diagonale/28);
    stringagirid=sprintf('%d',numerogirid);
    restod=mod(diagonale,28);
    % Calcolo dell'angolo da effettuare dopo l'ultimo
       angolo giro
    if restod<7
        numeroangolid=round(restod/26*360);
        stringaangolid=sprintf('%d',numeroangolid);
    elseif restod<15
        numeroangolid=round(restod/24*360);
        stringaangolid=sprintf('%d',numeroangolid);
    else
        numeroangolid=round(restod/28*360);
        stringaangolid=sprintf('%d',numeroangolid);
    end
    % Calcolo del tempo per effettuare la diagonale
    tempoD=temposetup+numerogirid*tempoungiro+(
       numeroangolid/360)*tempoungiro;
    % Creazione della stringa
    stringaD=strcat('G',stringagirid,'G',stringagirid,'A',
       stringaangolid,'A',stringaangolid,'Q');
 end
 % Se l'angolo da effettuare è abbastanza grande
 if abs(deltatheta)>10
     % Calcolo dell'angolo da effettuare
     esistetheta=1;
     numerogiritheta=floor((deltatheta*4+15)/360);
     % Se l'angolo è positivo
     if numerogiritheta>0
        % Calcolo del numero di angoli giri da effettuare
        stringagiritheta=sprintf('%d',numerogiritheta);
        restotheta=mod(deltatheta*4+15,360);
        % Calcolo dell'angolo da effettuare dopo l'ultimo
           angolo giro
        numeroangolitheta=round(restotheta);
```

```
            stringaangolitheta=sprintf('%d',numeroangolitheta)
                ;
            % Creazione della stringa
            stringaTheta=strcat('G',stringagiritheta,'GI',
                stringagiritheta,'A',stringaangolitheta,'AI',
                stringaangolitheta,'Q');
        % Se l'angolo è negativo
        else
            % Calcolo del numero di angoli giri da effettuare
            stringagiritheta=sprintf('%d',abs(numerogiritheta)
                );
            restotheta=mod(deltatheta*4+15,360);
            % Calcolo dell'angolo da effettuare dopo l'ultimo
                angolo giro
            numeroangolitheta=abs(round(restotheta));
            stringaangolitheta=sprintf('%d',numeroangolitheta)
                ;
            % Creazione della stringa
            stringaTheta=strcat('GI',stringagiritheta,'G',
                stringagiritheta,'AI',stringaangolitheta,'A',
                stringaangolitheta,'Q');
        end
        % Calcolo del tempo per effettuare l'angolo
        tempoTheta=temposetup+abs(numerogiritheta)*
            tempoungiro+abs(numeroangolitheta/360)*tempoungiro
            ;
    end
end
```

Elenco delle figure

Bibliografia

[1] Loris Cerroni. *Realizzazione di una cella automatizzata per il tracciamento di robot mobili.* Tesi di Laurea in Ingegneria dell'Automazione, Università degli Studi di Roma "Tor Vergata", 2009.

[2] Atmel Corporation. *ATmega640/1280/1281/2560/2561.* 2010.

[3] SHARP Corporation. *GP2Y0A21YK0F.* 2006.

[4] Ernest O. Doebelin, Cigada Alfredo, and Gasparetto Michele. *Strumenti e metodi di misura.* McGraw-Hill, 2008.

[5] Dimension Engineering. *Sabertooth 2x10 User's Guide.* 2007.

[6] Fulvio Forni. *Filtro di Kalman Esteso per la localizzazione di robot mobili.* Tesi di Laurea in Ingegneria Informatica, Università degli Studi di Roma "Tor Vergata", 2004.

[7] Marco Gerosi. *Navigazione e localizzazione del robot Nomad 150 mediante filtro di Kalman.* Tesi di Laurea in Ingegneria Informatica, Università degli Studi di Roma "Tor Vergata", 2005.

[8] Fabrizio Grandoni, Agostino Martinelli, Francesco Martinelli, Salvatore Nicosia, and Paolo Valigi. *Sensor fusion for robot localization.* Articu-

lated and Mobile Robotics for Services and Technologies (RAMSETE). Springer-Verlag, 2001.

[9] Simon J. Julier and Jeffrey K. Uhlmann. *Unscented Filtering and Nonlinear Estimation*, volume 92. Proceedings of the IEEE, 2004.

[10] Costanzo Manes, Agostino Martinelli, Francesco Martinelli, and Pasquale Palumbo. *Mobile robot localization based on a polynomial approach.* IEEE International Conference on Robotics and Automation (ICRA), Roma, 2007.

[11] Agostino Martinelli. *The odometry error of a mobile robot with synchronous drive system*, volume 18. IEEE Transactions on Robotic and Automation, 2002.

[12] Francesco Martinelli. *Robot localization: comparable performance of EKF and UKF in some interesting indoor settings.* MED, 2008.

[13] Marco Martines. *Attuazione di un sensore infrarossi per la mappatura di ambienti e inseguimento di landmark in movimento.* Tesi di Laurea in Ingegneria dell'Automazione, Università degli Studi di Roma "Tor Vergata", 2010.

[14] Giordano Scarciotti. *Visual Servoing mediante un sistema di visione stereoscopica.* Tesi di Laurea in Ingegneria dell'Automazione, Università degli Studi di Roma "Tor Vergata", 2010.

[15] Greg Welch and Gary Bishop. *An introduction to the Kalman filter.* SIGGRAPH, 2001.