Dipartimento di Ingegneria Elettrica e Elettronica
Corso di Laurea Specialistica in Ingegneria Elettronica
Università degli studi di Cagliari

PROGETTO E IMPLEMENTAZIONE DI LEGGI DI CONTROLLO DISTRIBUITE PER UN PLOTONE DI VEICOLI AUTONOMI

Marianna Stara

Relatore
Prof. Alessandro Giua

Anno Accademico

2008/2009

Ringraziamenti

Vorrei ringraziare i miei genitori, per tutto. Un Grande grazie a Giampaolo. Un grazie a tutti i miei amici e colleghi che mi hanno fatto compagnia in questi anni. A Silvia e Carla grazie dell'incoraggiamento durante questi mesi di lavoro e non solo.

Vorrei ringraziare il professore Alessandro Giua che mi ha permesso di fare questa tesi, per il tempo dedicatomi e i consigli ricevuti.

Un grazie anche alle persone che prima di me hanno lavorato con i robot Khepera III, che mi hanno aiutato nella fase iniziale e con la loro esperienza in questo tipo di progetti, Andrea e Mauro.

Grazie all'AutoLab per le ore passate in compagnia durante la realizzazione di questo progetto.

Grazie a tutti!!!!!!

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Obiettivo del progetto

Scopo di questo lavoro è la realizzazione di un sistema distribuito multi agente implementato su dei robot. Il laboratorio di automatica è in possesso di 5 *KheperaIII* prodotti dalla K-Team.[1] Il sistema di controllo distribuito verrà implementato su 4 di questi robot[2]. Il progetto è suddivisibile nei seguenti sotto-progetti:

- movimento in plotone degli agenti in formazione circolare (*Cerchio*);
- inseguimento lineare degli agenti mantenendo una distanza fissata (*Treno*);
- aggiramento degli ostacoli lungo il percorso.

I tre sotto progetti sono stati trattati singolarmente e in seguito integrati in un unico progetto che racchiude tutte e tre le funzionalità.

1.2 Motivazione del problema

Il campo di ricerca in cui questo progetto si colloca è il Networked Control System (NCS). Le funzionalità di un tipico NCS sono implementate grazie alluso di:

- sensori, per acquisire informazioni;

[1] *www.k-team.com*. Il sito della *K-Team* è in manutenzione dal mese di gennaio 2010.

[2] Per un malfunzionamento dovuto al mancato aggiornamento driver per la scheda WiFi del 5^o robot, si è lavorato solo su quattro di quelli a disposizione ma il tutto è estendibile a 5 e più robot.

- controllori, per prendere decisioni e impartire comandi;
- attuatori, per attuare il controllo;
- sistema di comunicazione, per abilitare lo scambio di informazioni.

L'ultima caratteristica non è sempre presente, in quanto è possibile implementare un algoritmo NCS senza comunicazione tra i singoli agenti del sistema.

Questa tesi, in particolar modo, pone l'attenzione su un particolare aspetto dei sistemi di controllo distribuiti, il comportamento emergente (*emerging behavior*). Tanti sistemi che implementano lo stesso comportamento, che unendosi formano un entità unica con funzionalità nuove derivanti dai singoli sistemi. In pratica è la situazione nella quale un sistema esibisce proprietà sulla base delle leggi che governano le sue componenti. Ciò permette di creare controllori capaci di fare comportamenti complessi, implementando funzionalità molto più elementari.

1.3 Descrizione del sistema fisico

Il sistema fisico può essere descritto da due punti di vista:

- punto di vista del robot;
- punto di vista dell'osservatore esterno;

che si differenziano per il tipo di approccio che si vuole usare per la soluzione del problema. Il primo ci permette di costruire un sistema di controllo distribuito senza controllo centrale, il secondo permette un controllo centralizzato che in questo progetto non è stato trattato. Per realizzare il progetto è stato studiato in dettaglio il sistema fisico dal punto di vista del robot.

Sensori IR Ogni robot può vedere entro un raggio di 10 centimetri massimo da se stesso (si veda figura 1.1) tramite i sensori IR. Questo va considerato come una specifica di progetto. Il robot è dotato anche di sensori ad ultrasuoni i quali hanno una portata da 20 cm a 4 metri. Resta quindi una zona d'ombra entro la quale non è possibile individuare la presenza di oggetti.

In questo lavoro si è scelto di usare i soli sensori IR. La scelta è stata dettata in primo luogo dall'interesse ad esplorare le potenzialità di un sistema di controllo regolato solo su dei sensori di prossimità, in secondo luogo dalla specifica di progetto. Infatti, tra le specifiche di progetto c'è il mantenimento di una distanza fissata tra i robot dell'ordine di qualche centimetro. Per un

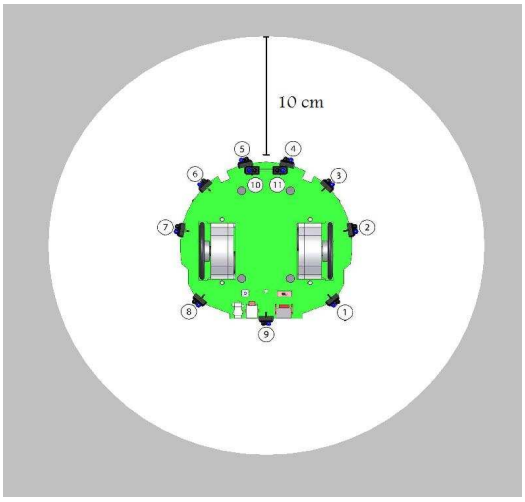

Figura 1.1: Raggio di vista del robot

plotone che si muove in formazione rettilinea si deve scegliere il valore della distanza tra i singoli agenti. Dato che il diametro di un robot è di circa 12 cm, la scelta più opportuna è quella di un valore inferiore ai 10 cm. Ciò permette un plotone in formazione più compatta e una realizzazione in spazi limitati. Se si volesse ottenere lo stesso con gli ultrasuoni, si potrebbe scegliere una distanza di 30 cm tra un robot e l'altro e quindi sarebbe necessario uno spazio rettilineo di circa 1.30 m per il solo posizionamento senza considerare il movimento degli agenti.

Consideriamo, quindi, una scena di 10 cm di raggio da ogni robot. Questo non ci permetterà di sapere dove sono gli altri agenti, ma solo di capire se c'è qualcosa nelle vicinanze, senza distinguere che oggetto abbiamo vicino. È sconsigliabile cercare di distinguere i vari oggetti in base al colore perché seppur stessi materiali di colori diversi riflettono diversamente l'infrarosso, la stessa riflessione si può avere anche da materiali completamente differenti posti a distanza diversa; non sarebbe possibile usare il valore fornitoci dal IR sia come misura della distanza dall'oggetto sia come identificazione dell'oggetto stesso. Inoltre le misurazioni sui sensori sono affette da rumore che non permette una precisione adeguata per la distinzione dei colori.

Si sono dotati sia i robot che gli ostacoli di un rivestimento in carta bianca che permette di rendere il più possibile costanti le proprietà riflessive, per poter così attuare una buona misura della distanza tra robot o tra robot e oggetti. Il bianco è stato scelto proprio per avere massima riflessione. Inoltre tale rivestimento permette di rendere il retro del robot uniforme. Il robot nel retro ha una superficie non omogenea a causa delle varie interfacce e della scheda wireless, tutte ivi collocate.

Il nostro algoritmo dovrà tenere conto del limite fisico datoci dall'uso di questi dispositivi, il quale influirà sui tempi di elaborazione. Per esempio, se il sistema campiona i sensori IR con un tempo inferiore al tempo in cui il robot esegue le misurazioni, non avremo un valore aggiornato ma solo il valore dato dalla precedente misurazione ripetuto per ogni periodo di interrogazione dei sensori. Quindi si potrà avere un aggiornamento della scena in un intervallo pari o maggiore a quello in cui il robot esegue le misurazioni dai sensori. Questo impone un vincolo sui tempi del sistema, come il tempo di campionamento.

Distanza di sicurezza Il mantenimento della distanza di sicurezza ha richiesto, prima di tutto, una funzione che leghi il valore misurato dai sensori IR e la distanza tra i veicoli. È stato studiato tale funzione basandosi su misure sperimentali in determinate condizioni di luce esterna, per evitar le interferenze il più possibile.

Il controllo vero e proprio per questa funzionalità è stato implementato con un regolatore PID proporzionale. Grazie a tale controllo il plotone si muove in formazione compatta in maniera uniforme senza collisioni tra i vari veicoli.

Movimento circolare Anche per la realizzazione del movimento circolare si sono utilizzati solamente i sensori IR, ciò ha richiesto un algoritmo di codifica del percorso da seguire in base alle particolarità del piano su cui si esegue la prova. Sono stati usati i due sensori IR posti nella parte inferiore del robot i quali hanno permesso il movimento lungo un particolare percorso deciso dalla pista. Quindi si è creato un algoritmo in grado di seguire una pista di forma circolare, ma il tutto è estendibile anche ad altri tipi di percorsi. La vicinanza dei sensori al pavimento rende necessario un forte contrasto per riuscire a distinguere il percorso da seguire da quello da non seguire. Per cui, la pista è stata realizzata sfruttando il contrasto presente tra i colori bianco e nero. Si è usato il colore nero per indicare il percorso e il bianco per il resto.

Ostacoli L'aggiramento degli ostacoli ha richiesto, prima di tutto, la scelta del tipo di ostacolo da aggirare. Per prima è sorta la necessità di un algoritmo che consentisse al robot di aggirare un altro robot fermo a causa di un malfunzionamento. Per cui l'ostacolo è stato creato di forma cilindrica delle stesse dimensioni del robot. Si è scelta una direzione preferenziale di sorpasso, nel nostro caso il robot sorpasserà da sinistra. Durante la fase di sorpasso è richiesto il controllo della scena da tutti i sensori posti intorno al robot per evitare collisioni. Successivamente si sono estese le funziona-

lità dell'algoritmo affinché fosse possibile aggirare più di un robot fermo e si evitassero oggetti inattesi presenti fuori dalla pista.

1.4 Contributi tesi

Il presente lavoro di tesi vuole essere un contributo alla ricerca nel campo dei NCS. In particolar modo si sono implementate funzionalità ampiamente discusse in letteratura, quali il *path following* e l'*obstacle avoidance*.

Per l'inseguimento del percorso desiderato possiamo citare il lavoro di Lan et al. [15] e quello di Lee e Park [17]. Quest'ultimo propone l'implementazione di comportamenti differenti a seconda che il robot sia o meno all'interno del percorso desiderato. Si è preso spunto da questa tecnica per implementare un algoritmo di ricerca del percorso nel caso il veicolo sia fuori da percorso desiderato; e un algoritmo per restare nel percorso indefinitamente una volta che il veicolo è all'interno di esso.

L'articolo [15] implementa un algoritmo per la ricerca del percorso basandosi sulla conoscenza da parte del robot della sua posizione assoluta e della posizione del percorso rispetto ad esso. Questo implica che il veicolo sappia dove si trova il percorso, e di conseguenza conosca la direzione verso cui dirigersi. Nel nostro caso non è presente una mappatura dell'intera scena, quindi il veicolo non sa la sua posizione assoluta né tanto meno quella del percorso. Nel nostro caso tale tecnica non risulta implementabile, si è quindi realizzato un comportamento random per la ricerca del percorso, ovviamente dipendente dalla morfologia dell'ambiente.

Il campo di studi per l'aggiramento degli ostacoli fornisce molti spunti implementativi. Cito qui alcuni articoli di riferimento: Lapierre et al.[16]e Ribeiro et al.[21].

L'articolo [16] propone di utilizzare la tecnica dei Deformable Virtual Zone (DVZ) per implementare l'aggiramento dell'ostacolo. Questo approccio permette al robot di tracciare una mappa dell'ambiente circostante entro il suo raggio di vista, questa mappa viene aggiornata man mano che il robot si muove, cambiando così la scena. Questo algoritmo è attuato nella sola direzione di movimento del veicolo.

Nell'articolo [21] viene proposto il metodo dei campi potenziali per evitare gli ostacoli. Ogni oggetto viene paragonato ad una particella di carica positiva o negativa. Assegnando carica negativa al veicolo, stessa carica agli ostacoli, e carica opposta alla zona in cui vogliamo che esso si muova, il movimento verrà attuato in funzione di questo campo di cariche. L'applicazione di questo algoritmo richiede la presenza di un solo veicolo che evita più osta-

coli. Nel nostro caso sono presenti più di un veicolo, i quali devono formare un plotone ed è quindi sconveniente un allontanamento tra essi.

Nel nostro caso si è quindi attuato

- un algoritmo per la ricerca degli ostacoli che rilevi la loro posizione rispetto a quella del veicolo;
- un comportamento che permetta di aggirare l'ostacolo e non semplicemente evitarlo.

Per cui gli studi fatti cono serviti da spunto per la fase implementativa, e sono stati adeguati alle esigenze del progetto in questione.

1.5 Risultati ottenuti

Gli algoritmi progettati sono stati implementati su ogni veicolo e per ogni simulazione sono stati salvati in opportuni file *(log-file)* i valori dei sensori IR misurati e i valori delle velocità lineare e angolari attuate. Ciò ha permesso di graficare i risultati ottenuti sperimentalmente e confrontarli con quelli della fase di progettazione. Questo tecnica di verifica è stata usata per l'algoritmo per mantenere la distanza di sicurezza.

Mentre per la verifica dell'algoritmo relativo al sorpasso non sono stati realizzati grafici del modello e la verifica è stata eseguita solo sulla base del comportamento atteso in termini di movimento del veicolo e il comportamento realmente attuato del robot. Per l'algoritmo di inseguimento del percorso, per verificare che il comportamento atteso corrisponda a quello attuato, si è tracciato il grafico della velocità angolare al variare del tempo.

Dai dati sperimentali raccolti possiamo dire che:

- l'algoritmo di inseguimento del percorso funziona correttamente per la pista scelta, ma è attuabile anche per piste di altre forme o diversi raggi di curvatura;
- l'algoritmo per mantenere la distanza di sicurezza funziona correttamente nel caso del veicolo capofila fermo, in tal caso la distanza di sicurezza è quella desiderata;
- l'algoritmo per mantenere la distanza di sicurezza funziona come le aspettative nel caso del veicolo capofila in movimento, in tal caso la distanza di sicurezza varia in un intervallo di ampiezza 1 cm con valori maggiori della distanza desiderata di 3 cm, questo a causa dell'impossibilità di avere dei punti di equilibrio del sistema;

- l'algoritmo per il sorpasso funziona correttamente nel caso di un veicolo che vuole sorpassare uno o più ostacoli;

- l'algoritmo per il sorpasso nel caso del plotone in movimento compatto può incombere nella simultaneità dei singoli sistemi di controllo dei robot, e quindi più di un veicolo inizia la fase di sorpasso contemporaneamente. È stato implementato un algoritmo di precedenza, per risolvere il problema ma in assenza di comunicazione tra i veicoli è impossibile discriminare tra i veicoli centrali i quali non possono sapere la loro posizione esatta. Questo algoritmo permette però di creare un ordinamento del sorpasso in cui il capofila sarà il primo a sorpassare, poi è il turno di quelli centrali e successivamente dell'ultimo.

1.6 Struttura tesi

Nel capitolo 2 sono illustrati gli articoli letti per realizzare il progetto. Il capitolo è suddiviso nei due principali argomenti trattati: l'inseguimento del percorso *(path following)* e l'aggiramento dell'ostacolo *(obstacle avoidance)*. Tale suddivisione è dettata dai due principali algoritmi realizzati. Questo capitolo vuole essere un riassunto dello studio preliminare svolto per la realizzazione del progetto e fonte di ispirazione per eventuali sviluppi futuri.

Nel capitolo 3 è presentata la classificazione per i due principali tipi di robot su ruote, gli *Hilare-type* e i *car-like*, e il modello cinematico del tipo di robot che meglio descrive i veicoli KheperaIII, su cui il progetto in questione è implementato.

Nel capitolo 4 viene studiato il modello complessivo per il plotone dei veicoli affinché mantengano una distanza fissata tra loro. L'analisi è svolta prima sul modello di un solo veicolo, si è analizzato il controllo a ciclo aperto relativo ad esso e la realizzazione del controllo a ciclo chiuso per il mantenimento della distanza desiderata. Successivamente si è studiato il sistema complessivo relativo all'intero plotone.

Nel capitolo 5 è descritto il tipo di controllo implementato sui veicoli separatamente per ogni funzionalità. Il sistema di controllo implementato è un sistema di controllo distribuito. Questo implica che tutti i robot hanno lo stesso sistema di controllo che rende il loro comportamento autonomo.

Nel capitolo 6 si descrive come è possibile utilizzare i sensori infrarossi per individuare un ostacolo e misurare la distanza tra il veicolo e l'eventuale ostacolo. Questo studio pone le basi per lo sviluppo di qualsiasi algoritmo di inseguimento tra i veicoli e aggiramento degli ostacoli. La parte sviluppata

nella sezione 6.2 è propedeutica per l' implementazione degli algoritmi sul robot.

Nel capitolo 7 è trattata l'implementazione sui robot KheperaIII degli algoritmi presentati nella sezione 5. Questo capitolo è strutturato in due parti:

- funzionalità generali implementate, dove sono spiegate le funzioni base necessarie per l'implementazione degli algoritmi veri e propri;
- implementazione degli algoritmi, dove è spiegata la tecnica utilizzata per implementare gli algoritmi in un robot Khepera III.

Nel capitolo 8 è spiegato come sono stati svolti gli esperimenti, la realizzazione fisica e l'algoritmo implementato per eseguire il test. Si sono discussi i risultati ottenuti in fase di verifica sul robot KheperaIII.

Nel capitolo 9 sono presentate le conclusioni relative al progetto svolto.

Nell'appendice A è riportata la definizione formale di sistema ibrido.

Nell'appendice B sono presentati i sistemi interconnessi, nelle possibili connessioni.

Nell'appendice C vengono definiti i controllori PID.

Nell'appendice D è riportato il codice C del programma principale,e l'interfaccia delle funzioni già presentate nel capitolo 7.

Nell'appendice E è riportato il codice matlab per l'interpolazione della curva per il calcolo della distanza.

Nell'appendice F è riportato un breve manuale sull'uso del robot Khepera III.

Nell'appendice G sono riportati alcuni richiami sul sistema operativo Linux in generale, e i comandi principali da conoscere per poter usare il robot.

Nell'appendice H sono riportate le definizione degli acronimi utilizzati in questa tesi.

Capitolo 2

Rassegna della letteratura

In questo capitolo sono illustrati gli articoli letti per realizzare il progetto. Il capitolo è suddiviso nei due principali argomenti trattati: l'inseguimento del percorso *(path following)* e l'aggiramento dell'ostacolo *(obstacle avoidance)*. Tale suddivisione è dettata dai due principali algoritmi realizzati. Questo capitolo vuole essere un riassunto dello studio preliminare svolto per la realizzazione del progetto e fonte di ispirazione per eventuali sviluppi futuri.

2.1 Inseguimento del percorso desiderato

L'inseguimento del percorso desiderato è ampiamente trattato in letteratura. Sono citati qui gli articoli letti per la realizzazione dell'algoritmo dell'inseguimento del percorso, nella particolarità, nel nostro caso, di percorso di forma circolare.

- Lan et al. [15] studiano il problema del *path following* dove un gruppo di unicicli sono guidati verso un dato percorso fino ad ottenere una data formazione tra i veicoli. L'approccio utilizzato è un controllo ibrido, per considerare il fatto che i vicini di ciascun robot cambiano al passare del tempo, a causa delle limitazioni delle capacità dei sensori. Lo spazio di stato di ciascun veicolo è stato partizionato in diverse regioni relative al percorso. In seguito è stata sintetizzata una legge di controllo che permettesse di seguire tutti i veicoli dentro una regione dove un altra legge di controllo opera per ottenere contemporaneamente l'inseguimento del percorso e la coordinazione del movimento tra gli agenti. I risultati ottenuti dalla legge di controllo per il cambio di stato ha permesso di ottenere che i veicoli una volta giunti nel percorso si dispongano in una formazione regolare, mantenendo una data distanza da i loro vicini e che il plotone si muova in maniera compatta. Si

presume che ogni singolo agente sappia, istante per istante, dove si trova il percorso da seguire rispetto alla sua posizione attuale. Ciò richiede che il percorso da seguire sia in un qualche modo memorizzato nel robot.

- Lee e Park [17] discutono del problema di non poter garantire la stabilità asintotica di un punto di equilibrio in un sistema dinamico di inseguimento del percorso, dovuto dall'avere dei vincoli anolonomi[1] nel sistema dinamico. Un sistema con vincoli anolònomi è altamente influenzato dalo stato iniziale, per cui le performance dipenderanno da quest'ultimo. Viene presentato un sistema dinamico di inseguimento del percorso in cui sono generati due tipi di percorsi. Il primo è il percorso originale desiderato, il inseguimento de percorso a cui tutti i robot devono convergere. Il secondo è un percorso temporaneo, generato dal sistema, differente dal percorso originale che permette di eliminare i problemi dovuti allo stato iniziale. Il lavoro si conclude con i risultati delle simulazioni al computer, che dimostrano che il sistema garantisce la stabilità locale. Tutto lo studio considera come robot un Hilare-type.

2.2 Aggiramento ostacolo

Riguardo a questo argomento si sono trovati molti articoli che sono serviti da spunto per realizzare l'algoritmo di controllo. Citerò qui i più importanti.

2.2.1 Metodo dei campi potenziali

Nel saggio di Ribeiro [21] viene fatto uno studio per l'aggiramento degli ostacoli basato sul *metodo dei campi potenziali*. In questo approccio il robot è considerato come una particella che si muove immersa in dei campi potenziali generati dagli obiettivi (zone vuote in cui il robot può passare) e dagli ostacoli presenti nell'ambiente (la morfologia dell'ambiente è nota a priori). Ogni campo potenziale è visto come un campo di energia e il suo gradiente, in ciascun punto, è una forza. Il robot immerso in questi campi potenziali è soggetto all'azione di una forza che lo guida verso l'obiettivo (questa è l'azione della forza attrattiva che deriva dal gradiente del potenziale attrattivo generato dall'obiettivo) mantenendolo lontano dagli ostacoli (questa è l'azione ottenuta dalla forza di repulsione che il gradiente del potenziale di repulsione genera dagli ostacoli). Tutto il sistema viene assimilato a delle

[1] Nonholonomic constraints, restrizione sugli atti di moto.

particelle cariche positivamente e negativamente che si attraggono e si respingono. Il robot, ad esempio, è una particella a carica positiva, l'obiettivo è una particella carica negativa e gli ostacoli sono particelle a carica positiva. L'ostacolo genera una forza repulsiva che fa in modo che il robot si allontani dall'ostacolo stesso. La combinazione delle forze attrattive e repulsive guida il robot verso il percorso lontano dagli ostacoli. Questo approccio richiede che l'ambiente di esecuzione sia noto a priori, in modo da dare ad ogni zona il suo giusto potenziale attrattivo o repulsivo. Tratta l'argomento dei campi potenziali anche l'articolo[14].

Per far si che l'algoritmo funzioni anche in ambienti non noti a priori è stata aggiunta la parte relativa al metodo *Vector Field Histogram (VFH)*. Questo metodo[6] permette di individuare ostacoli non conosciuti a priori e di evitare collisioni con altri veicoli in movimento. Questo metodo crea un modello dell'ambiente circostante come un istogramma bidimensionale. La scena viene aggiornata continuamente con un tempo di campionamento dato dalla lettura dei sensori.

2.2.2 Curvature-velocity method (CVM)

Quasny et al. [20] trattano la soluzione al problema di *obstacle avoidance* tramite il metodo CVM per robot Hilare-type. Questo metodo permette di decidere la velocità lineare e angolare del robot basandosi sulla massimizzazione di una funzione obiettivo che considera la velocità del robot, la direzione dell'obiettivo, e la distanza dall'ostacolo, verificando che rispetti anche i limiti fisici e dinamici del robot. Il metodo si basa sulla seguente funzione:

$$f(v, w) = \alpha_1 speed(v) + \alpha_2 dist(v, w) + \alpha_3 head(w)$$

dove v è la velocità lineare, w la velocità angolare, speed(v) la velocità del robot, head(w) la direzione del robot e dist(v,w) la distanza da un set di ostacoli entro un dato limite. Tutte queste grandezze sono vincolate dai limiti fisici e dinamici del robot. Il CVM aggiusta i coefficienti α per massimizzare la funzione precedente per trovare la coppia di valori di v e w ottimali da applicare in tempo reale[2] al veicolo.

[2]Dato in realtà dal tempo di convergenza ad un valore che soddisfi la massimizzazione della funzione *f*, quindi l'algoritmo può richiedere diversi passi prolungando così il tempo di campionamento.

2.2.3 Metodo radiale

Benayas et al. [4] propongono il metodo radiale[3] come miglioria del metodo *CVM*. Questo metodo utilizza la distanza radiale per calcolare la miglior direzione di movimento in un solo passo. Successivamente il metodo CVM usa questa informazione per calcolare la velocità lineare e angolare ottimali. Questo algoritmo viene applicato per ambienti interni. Ogni ostacolo viene approssimato con un cerchio, in modo da rendere più omogeneo il calcolo della distanza, tale calcolo sarebbe molto complesso se si considerano ostacoli di geometria arbitraria. Per ciascun ostacolo viene calcolato il valore del fascio radiale (beam) associato. Questo valore dipenderà dagli angoli limite delle tangenti all'ostacolo e dalla distanza minima tra il veicolo e l'ostacolo. Si ha praticamente una visione radiale più accurata della scena intorno al veicolo, la quale verrà poi elaborata dal CVM per impostar la migliore velocità lineare e angolare.

2.2.4 Deformable Virtual Zone principle (DVZ)

Lapierre et al. in [16] per affrontare il problema dell'aggiramento degli ostacoli usano il principio della *zona virtuale deformabile*. Viene definita una zona libera intorno al veicolo, nella quale se si presenta un ostacolo il veicolo sarà costretto a muoversi di conseguenza. La cinematica del robot dipenderà dalla collocazione della zona di rischio che esso individua intorno a sé. Questa zona può essere deformata in base alle informazioni che il robot acquisisce dalla scena, in modo da allontanarsi dagli ostacoli che si presentano di fronte a lui. In questo studio viene considerata solo la zona davanti al robot, quindi, supponendo un movimento in avanti del robot, ci sarà un aggiornamento della scena solo nella direzione di movimento del robot.

[3] Beam method.

Capitolo 3

Classificazione robot e modello cinematico

Nella prima sezione si darà una distinzione tra i due principali tipi di robot dotati di ruote: gli *Hilare-type* e i *car-like*. Nella seconda sezione è descritto il modello cinematico del tipo di robot che meglio descrive i veicoli KheperaIII, su cui il progetto in questione è implementato.

3.1 Classificazione robot

Vi sono diversi tipi di robot su ruote, in questa sezione sono descritti due importanti modelli in quanto i più simili al robot KheperaIII. Per le definizioni si è fatto riferimento al libro [11] "Autonomous Robots: Modeling, Path Planning, and Control". I tipi di robot sono:

- Car-like. I robot di tipo car-like sono dotati di 4 ruote, e il motore che governa il movimento è di tipo differenziale, simile a quello di una macchina. Il motore differenziale permette di impartire velocità diverse alle due ruote motrici, in genere quelle posteriori. Un meccanismo di sterzo permette al veicolo di girare tramite una rotazione delle ruote anteriori.

- Hilare type. I robot Hilare-type hanno due ruote motrici indipendenti, e sono, in genere, bilanciati da una terza ruota passiva. La cinematica di questo tipo di veicolo viene in genere paragonata a quella di un carrello della spesa, per le similitudini quali il moto governato dal moto delle due ruote posteriori, e la posizione del baricentro che grava sulla ruota anteriore.

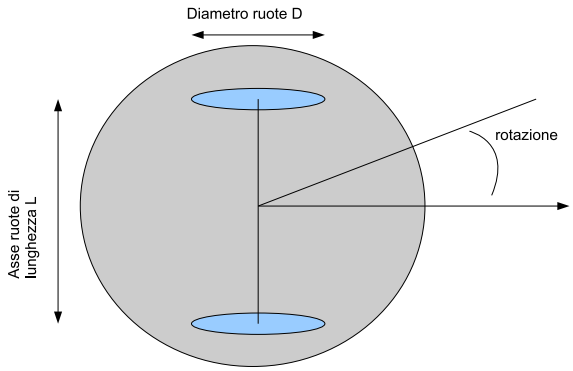

Figura 3.1: Schema dell'asse delle ruote

Non è possibile identificare il robot KheperaIII col modello car-like in quanto non sono presenti le due ruote anteriori, che nel car-like permetto al robot di girare. Il robot KheperaIII può, invece, essere identificato col modello di robot Hilare-type in quanto:

- ha due ruote motrici, governate da un motore differenziale che permette di impartire velocità diverse alle singole ruote;
- il baricentro è spostato in avanti e grava sulla batteria che, si può dire, funga dalla terza ruota passiva del modello Hilare-type.

Si noti che seppur le ruote del KheperaIII non sono governate da due motori indipendenti, come richiesto dalla definizione del Hilare-type, a ciascuna di esse può essere imposta una velocità indipendentemente dall'altra.

3.2 Modello cinematico Hilare-type

In questa sezione verrà presentato il modello cinematico di un robot Hilare-type. In figura 3.1 è raffigurato uno schema dell'asse delle ruote, dove la freccia indica la direzione di movimento dettato dalla velocità lineare, e la rotazione indica un eventuale virata dovuta alla velocità angolare. La lunghezza dell'asse tra le ruote è chiamata L e il diametro della ruota con D. Considerare il robot come un oggetto dotato di una velocità lineare v e di una velocità angolare w che regolano il moto rispetto al suo punto di equilibrio ci permette di considerare la cinematica del robot come quella di un *uniciclo*[1].

[1]Dispositivo fisico dotato di una sola ruota, la cui cinematica, equivalente a quella di un Hilare-type robot, è rappresentata da una velocità lineare e una velocità angolare.

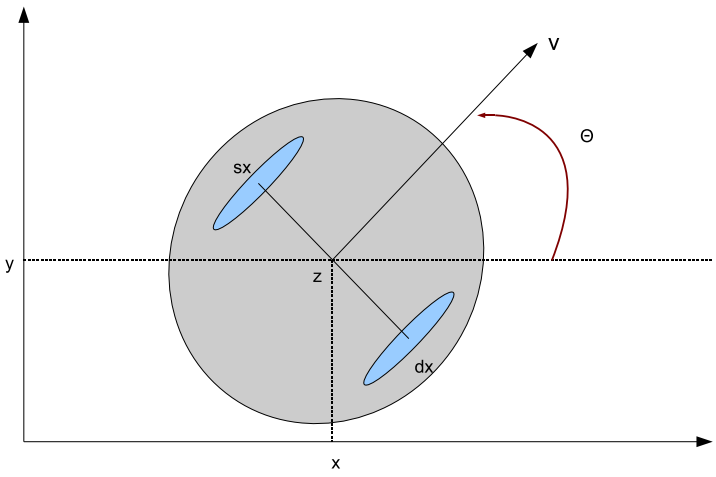

Figura 3.2: Sistema di riferimento per la rappresentazione cinematica

Si indica come centro del robot il punto $z = (x, y)$ in figura 3.2. Nella stessa figura con l'etichetta *sx* si indica la ruota sinistra, e con *dx* la ruota destra, l'angolo θ indica la rotazione del veicolo rispetto alla direzione principale del moto. La lunghezza dell'asse è stata misurata,$L = 88mm$ mentre il diametro delle ruote ci viene fornito dalla casa produttrice $D = 41mm$.

La velocità di rotazione della ruota destra è indicata con V_d, e quella della ruota sinistra è V_s, entrambe sono espresse in $[giri/s]$. È possibile convertire la velocità di rotazione in una velocità lineare espressa in $[mm/s]$ (indichiamo con v_d la velocità lineare della ruota destra e con v_s quella della ruota sinistra), con la seguente equazione:

$$\begin{cases} v_d = 2\pi R \cdot V_d \\ v_s = 2\pi R \cdot V_s \end{cases} \tag{3.1}$$

per cui la velocità lineare v del robot sarà data dalla media delle velocità delle singole ruote:

$$v = \frac{v_d + v_s}{2}.$$

La velocità angolare w è funzione della differenza delle velocità lineari delle ruote, v_d e v_s. Per convenzione si indica la velocità angolare di segno positivo quando la rotazione è in senso antiorario, di segno negativo in caso contrario. Per il calcolo della velocità angolare w, si sfrutta la definizione di

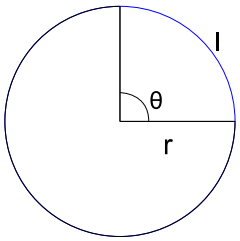

Figura 3.3: Definizione di radiante.

radiante[2] illustrata in figura 3.3, dove r è il raggio, l è l'arco e θ l'angolo al centro della circonferenza. Possiamo quindi definire l'angolo θ come:

$$\theta = \frac{l}{r}$$

Si considera come raggio di rotazione la distanza tra le due ruote pari all'asse L, e come variazione dell'arco la differenza delle due velocità lineari; per cui la w sarà la variazione dell'arco rispetto al tempo, il tutto fratto il raggio.

$$w = \dot{\theta} = \frac{(v_d - v_s)}{L} \tag{3.2}$$

In equazione (3.2) si vede che se $v_d > v_s$ la rotazione sarà in senso antiorario ($w > 0$), se $v_d < v_s$ la rotazione sarà in senso orario ($w < 0$) e non vi sarà rotazione nel caso le due velocità siano identiche. Questa definizione impone un approssimazione in quanto non si sta calcolando l'arco ma la retta che unisce i due estremi dell'arco.

Si esegua un ulteriore conversione di unità di misura delle velocità di rotazione delle ruote (V_d e V_s) da $[giri/s]$ a $[rad/s]$. Si indica con V_{dr} la velocità di rotazione in $[rad/s]$ della ruota destra, e con V_{sr} quella relativa alla ruota sinistra.

$$\begin{cases} V_{dr} = 2\pi \cdot V_d \\ V_{sr} = 2\pi \cdot V_s \end{cases} \tag{3.3}$$

[2]Il radiante è l'angolo al centro di una circonferenza, di raggio arbitrario, che sottende un arco di lunghezza pari al raggio.

Tale conversione permette di scrivere in maniera più compatta l'equazione del modello cinematico del sistema nel caso in due dimensioni, come illustrato in figura 3.2 e definito nell'equazione (3.4).

$$\begin{cases} \dot{x} = & \frac{R}{2}\,(V_{dr} + V_{sr})cos(\theta) \\ \dot{y} = & \frac{R}{2}\,(V_{dr} + V_{sr})sin(\theta) \\ \dot{\theta} = & \frac{R}{L}\,(V_{dr} - V_{sr}) \end{cases} \tag{3.4}$$

Si dovrà invertire l'espressione per poter impartire i comandi al robot, il quale governa le ruote imponendo delle velocità in giri al secondo.

Capitolo 4

Modello controllore

In questa sezione viene studiato il modello complessivo per il plotone dei veicoli affinché mantengano una distanza fissata tra loro. L'analisi è svolta prima sul modello di un solo veicolo, si è analizzato il controllo a ciclo aperto relativo ad esso e la realizzazione del controllo a ciclo chiuso per il mantenimento della distanza desiderata. Successivamente si è studiato il sistema complessivo relativo all'intero plotone.

4.1 Modello singolo veicolo

Il veicolo può essere modellato tramite il modello Ingresso-Uscita (IU) di controllo a ciclo aperto rappresentato in figura 4.1, dove:

- v è l'ingresso, con $v : \mathbb{R} \to \mathbb{R}$;
- z è l'uscita, con $z : \mathbb{R} \to \mathbb{R}$.

L'ingresso del sistema è la velocità v, l'uscita è la posizione z, per cui se ne deduce che il processo P che modella il sistema è un semplice integratore.

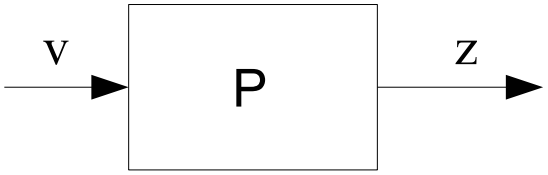

Figura 4.1: Modello per un veicolo con controllo a ciclo aperto.

Per cui l'intero sistema è modellato dalla equazione (4.1), dove la variabile indipendente è il tempo $t \in \Re$. Il sistema è di ordine 1 in quanto l'uscita ha ordine di derivazione unitario.

$$\frac{dx(t)}{dt} = v(t)$$

Che espresso in altra forma diventa:

$$z(t) + z(0) = \int_0^t v(t)dt. \tag{4.1}$$

L'equazione (4.2) è la trasformata di Laplace, tramite teorema dell'integrale, del sistema (4.1).

$$Z(s) = \frac{1}{s}V(s) \tag{4.2}$$

La funzione di trasferimento del processo è indicata nell'equazione (4.3), ed indica appunto l'integrazione insita nel modello.

$$P(s) = \frac{V(s)}{Z(s)} = \frac{1}{s} \tag{4.3}$$

Si è scelto di rappresentare il modello complessivo, quello con tutti i veicoli, in un modello in Variabili di Stato (VS), in quanto in generale l'uscita di un sistema in un certo istante di tempo t non dipende dal solo ingresso al tempo t, ma anche dall'evoluzione precedente del sistema. Ed è possibile tenere conto di questo fatto introducendo una grandezza intermedia tra ingressi e uscite, chiamata *stato* del sistema. Lo stato gode della proprietà di concentrare in sè l'informazione sul passato e sul presente sistema.

Illustro il modello in VS per un sistema Multiple-Input Multiple-Output (MIMO)[1].

Definizione 1 ***Modello in VS*** *Un modello in variabili di stato di ordine n, con r ingressi e p uscite è descritto come segue. Si prenda :*

- *il vettore di stato* $x(t) = [x_1(t), x_2(t), \ldots, x_n(t)]^T \in \mathbb{R}^n$;
- *il vettore degli ingressi* $u(t) = [u_1(t), u_2(t), \ldots, u_n(t)]^T \in \mathbb{R}^r$;
- *il vettore delle uscite* $y(t) = [y_1(t), y_2(t), \ldots, y_n(t)]^T \in \mathbb{R}^p$.

[1]Il modello per il sistema Single-Input Single-Output (SISO) si può vedere come un caso particolare del modello MIMO

Il modello in VS è:

$$\begin{cases} \dot{\mathbf{x}}(t) &= \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t) \qquad \textit{con: } \mathbf{A} \in \mathbb{R}^{n\times n} \textit{ e } \mathbf{B} \in \mathbb{R}^{n\times r} \\ \mathbf{y}(t) &= \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t) \qquad \textit{con: } \mathbf{C} \in \mathbb{R}^{p\times n} \textit{ e } \mathbf{D} \in \mathbb{R}^{p\times r} \end{cases} \tag{4.4}$$

Da ora in avanti si userà la rappresentazione del modello in VS. Lo stato del sistema al tempo t, che indicheremo con $x(t)$, è la posizione del veicolo, indicata con z nella figura 4.1, e coincide con l'uscita del sistema $y(t)$. L'ingesso al tempo t è la velocità $v(t)$. Il sistema è di tipo SISO quindi con 1 ingresso e 1 uscita, e di ordine 1 quindi con un solo stato. Il modello generale in VS (4.4) si ridurrà ad due equazioni lineari, una per la trasformazione dello stato e l'altra per la trasformazione dell'uscita. La (4.5) rappresenta il modello in VS, dove, in accordo con la (4.4) si è indicata la variabile di uscita con y(t), e la variabile di stato con x(t).

$$\begin{cases} \dot{x}(t) = v(t) \\ y(t) = x(t) \end{cases} \tag{4.5}$$

dove::

- $\mathbf{A} \in \mathbb{R}^{1\times 1}$ e $\mathbf{A} = 0$;
- $\mathbf{B} \in \mathbb{R}^{1\times 1}$ e $\mathbf{B} = 1$;
- $\mathbf{C} \in \mathbb{R}^{1\times 1}$ e $\mathbf{C} = 1$;
- $\mathbf{D} \in \mathbb{R}^{1\times 1}$ e $\mathbf{D} = 0$.

Riassumendo :

- il singolo veicolo è rappresentato da un modello in VS;
- il modello è dinamico, l'ordine del sistema è diverso da zero o, in altri termini, l'uscita dipende dallo stato e non solo dall'ingresso;
- il modello è lineare, la trasformazione del'uscita e l'equazione di stato sono lineari;
- il modello è stazionario, in (4.5) non vi è dipendenza esplicita dal tempo;
- il modello è strettamente proprio, l'ordine di derivazione dell'uscita ($n = 1$) è maggiore di quello dell'ingresso ($m = 0$).

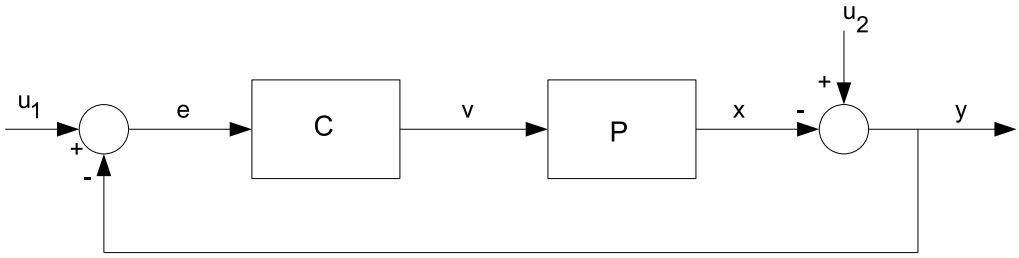

Figura 4.2: Modello del singolo veicolo controllato in retroazione

4.2 Controllo sul singolo veicolo

In questa sezione è discussa la tecnica di controllo a ciclo chiuso utilizzata per governare il moto del veicolo in funzione del moto degli altri veicoli.

Il moto di ciascun veicolo è regolato dal veicolo che lo precede. Si considererà il movimento solo lungo una direzione. Ogni veicolo deve impostare la propria velocità lineare, quindi il controllo agirà sul solo moto rettilineo. Il sistema di controllo sarà modellato su una dimensione, semplificazione che permette di rappresentare a pieno il sistema. Per meglio capire il sistema di controllo si veda figura 4.2, dove:

- u_1 e u_2 sono i due ingressi al sistema. In particolare u_1 è il set-point, il valore di distanza che i veicoli devono mantenere tra loro; u_2 è la posizione del veicolo che precede quello rappresentato nel modello;
- y è l'uscita del sistema, corrispondente alla distanza misurata tra il veicolo nel modello e quello che lo precede, $y = u_2 - x$;
- C è il controllore che permetterà al sistema a ciclo chiuso di far convergere l'uscita al valore desiderato;
- P è il processo del sistema, rappresenta il veicolo e corrisponde ad una funzione integratrice;
- e è l'errore, corrispondente a $e = u_1 - y$;
- x è la posizione del veicolo, ottenuta dall integrale di v;
- v è la velocità del veicolo imposta tramite il controllore C.

La figura 4.2 mostra come il controllore C agisca sul modello del veicolo proposto nella figura 4.1 in funzione dell'errore e al tempo t. Se l'errore $e(t)$ è nullo, e quindi l'uscita y è uguale al set-point $u_1(t)$ il controllore non agisce

sulla velocità $v(t)$. Al contrario, se $e(t) \neq 0$ allora sarà necessaria un'azione correttrice sulla variabile $v(t)$.

Nella rappresentazione in VS del sistema di controllo:

- x è lo stato del sistema[2], con $x \in \mathbb{R}$;
- y l'uscita del sistema, con $y \in \mathbb{R}$;
- il vettore $\mathbf{u(t)} = \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix}$ è l'insieme degli ingressi del sistema (L'ingresso $u_1(t)$ è costante mentre $u_2(t)$ è un ingresso variabile col tempo), $u(t) \in \mathbb{R}^2$.

È necessario decidere il tipo di regolazione da applicare. In appendice C sono riportati alcuni richiami teorici sui sistemi di regolazione. Procedendo per gradi si è realizzato prima di tutto il regolatore più semplice, un controllore ad azione proporzionale. Posto il guadagno pari a K possiamo scrivere il modello del sistema di controllo. La velocità v, nel modello in VS è rappresentata dalla derivata dello stato e quindi $v(t) = \dot{x}(t)$. La dinamica del sistema è:

$$\dot{x}(t) = -K \cdot e(t),$$

è presente il segno meno in quanto la velocità da impostare deve avere segno opposto all'errore e:

se $y(t) > u_1(t)$ allora $e(t) < 0$ e $v(t) > 0$;

se $y(t) > u_1(t)$ allora $e(t) > 0$ e $v(t) < 0$.

Il modello in VS è il seguente:

$$\begin{cases} \dot{x}(t) = -Kx(t) - K \begin{bmatrix} 1 & -1 \end{bmatrix} u(t) \\ y(t) = -x(t) + \begin{bmatrix} 0 & 1 \end{bmatrix} u(t) \end{cases} \tag{4.6}$$

Si considera come stato iniziale lo stato $x(t) = 0$.

Dall'analisi della stabilità col criterio degli autovalori si vede che il sistema è sempre stabile in quanto l'unico autovalore[3] ha valore pari a -K.

L'ingresso u_2 è un segnale a rampa in quanto la posizione del veicolo davanti aumenta con il passare del tempo: $u_2(t) = t$. Dato che è stato utilizzato un regolatore con azione proporzionale, ci dobbiamo aspettare un errore a regime non nullo, causato anche dal fatto che la catena diretta a

[2]Tutte le variabili sono espresse nel tempo.

[3]In questo caso la matrice della dinamica è $A_{[1\times1]}$, per cui il calcolo degli autovalori tramite il $det(\lambda I - A) = 0$ si riduce all'equazione $\lambda + K = 0$.

K	e [cm]
1	3
3	1
5	0.5
10	0.3

Tabella 4.1: Tabelle valori guadagno e relativo errore a regime.

monte dell'ingresso u_2 ha un solo polo nell'origine e non è sufficiente per ottenere un errore a regime nullo con un ingresso a rampa. La regolazione applicata è quindi di tipo *statico*, con errore a regime finito, ma non nullo.

Per decidere i vari valori dei parametri si è costruito il modello con Simulink e si è calcolato l'errore a regime.

È stata scelta come distanza desiderata tra i singoli veicoli u_1 pari a 4 cm

Dato che il nostro sistema sarà sicuramente stabile il valore minimo del guadagno K è 1, e partendo da esso è possibile analizzare i valori dell'errore a regime e il tempo del transitorio.

In tabella 4.1 è indicato:

- nella prima colonna il valore del guadagno del controllore proporzionale impostato;
- nella seconda colonna il valore dell'errore a regime corrispondente al particolare valore di K.

Si è scelto infine un guadagno K pari a 10 con un errore a regime di 0.3 cm. Per un guadagno più elevato l'uscita del sistema oscilla. L'andamento dell'uscita per un guadagno K = 10 è illustrato in figura 4.3. Si noti che l'errore a regime dato da tale regolatore statico è di circa 3 mm, quindi di un valore basso rispetto all'errore di misura.

L'ingresso $u_1(t)$ (si veda figura 4.4) è stato realizzato come una funzione a gradino[4] con valore iniziale nullo e valore finale pari a 4, l'istante del fronte di salita del gradino è t=1 s. Il tempo del transitorio, indicato in figura 4.3, è di circa 85 s che possiamo considerare accettabile.

Per il nostro sistema[5] i valori di errore a regime e tempo di transitorio ottenuti sono più che ottimi, è inutile progettare un regolatore più complicato per avere un errore a regime nullo, in quanto complicherebbe il sistema senza aggiungere migliorie apprezzabili. Inoltre aumentare il guadagno provoca delle oscillazioni dell'uscita.

[4]Blocco *Step* di *Simulink*

[5]Considerando l'errore della misura della distanza, anche con un controllore astatico non si otterrebbe comunque un miglioramento apprezzabile.

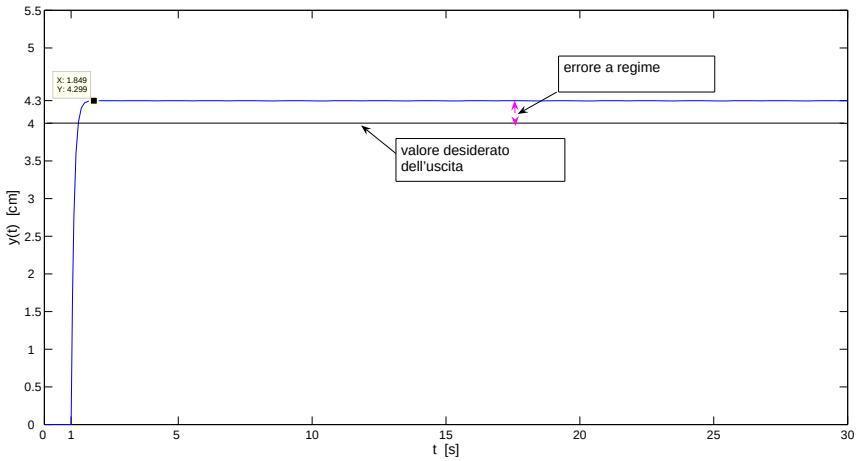

Figura 4.3: Transitorio del controllore proporzionale con guadagno pari a 10

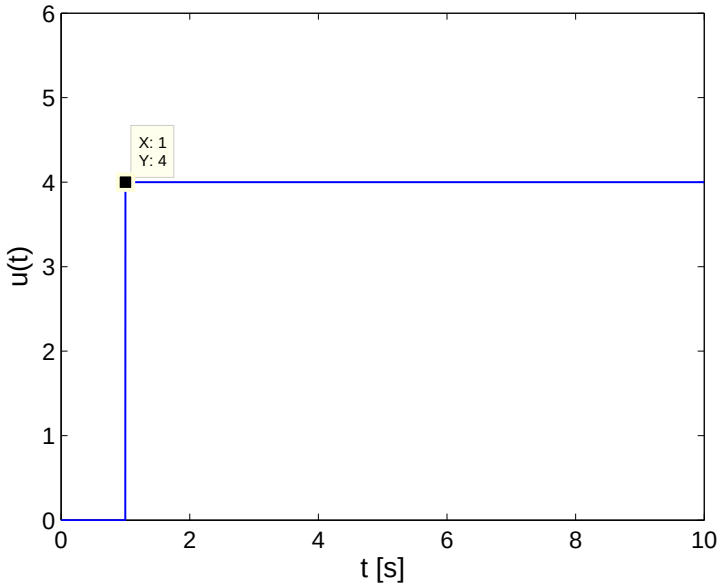

Figura 4.4: Set-point

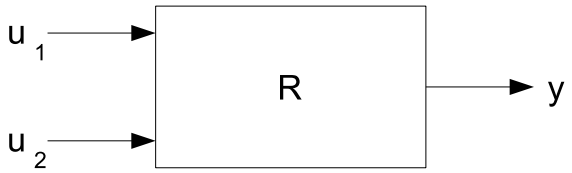

Figura 4.5: Blocco rappresentante un veicolo

4.3 Modello complessivo

Prima di realizzare il modello complessivo si è racchiuso in un unico blocco il sistema rappresentante ciascun veicolo, per il primo veicolo si veda figura 4.1 per gli altri veicoli figura 4.2. Il modello del sistema di controllo relativo ad un veicolo (figura 4.5) è indicato con la lettera R, come Robot[6].

Il modello complessivo dei veicoli deve tener conto del tipo di connessioni tra essi. I singoli sistemi, uno per ciascun robot, saranno connessi con il sistema del veicolo che lo precede e con quello che lo segue. Il tipo di connessione tra i singoli sottosistemi è a cascata[7].

Il sistema complessivo è chiamato *Sp*, indicando con la lettera S il sistema e con la p il fatto che racchiuda il funzionamento dell'intero il plotone (si veda figura 4.7).

Si suppone che il moto di tutti i veicoli sia orientato solo lungo la direzione x e che il moto sia rettilineo, e solo successivamente, con l'aggiunta del moto nel percorso di forma circolare si supporrà che il cerchio sia abbastanza grande da consentire al primo robot di non vedere l'ultimo della coda.

Nel sistema *Sp* il primo veicolo R_1 non ha nessun veicolo davanti e quindi non potrà regolare la sua velocità con la legge di controllo che governa gli altri. Per cui il suo modello è privo di controllo a ciclo chiuso, come indicato in figura 4.1 e l'equazione (4.5) che lo descrive è modificata come segue:

$$\begin{cases} \dot{x}(t) = v(t) \\ y(t) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} d_1(t). \end{cases} \tag{4.7}$$

Il sistema R_1 ha un modello senza regolazione, con due uscite:

- $y_1(t)$ corrispondente alla distanza da lui stimata $d_1(t)$, e in questo caso è il valore di fondo scala della misura, $y_1(t)$ è anche uscita del sistema *Sp*;

[6]La lettera V di veicolo che si sarebbe voluto usare per uniformità di linguaggio può creare confusione tra la variabile velocità e la parola veicolo.

[7]Si veda appendice B.1 per i richiami teorici.

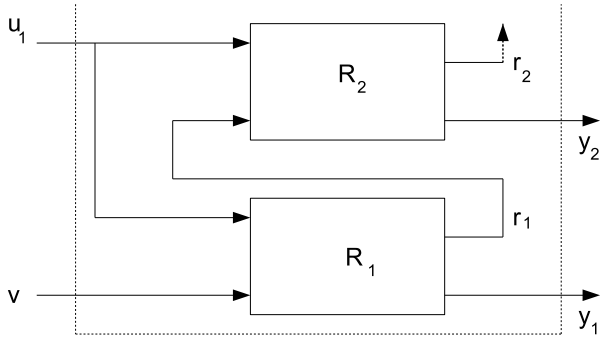

Figura 4.6: Esempio di connessione tra due sistemi per l'algoritmo di inseguimento

- $y_2(t)$ corrispondente alla posizione di R_1, è una variabile interna al sistema *Sp* e corrisponde ad un ingresso di R_2.

I due ingressi del sistema R_1 sono: $u(t) = \begin{bmatrix} v(t) \\ d_1(t) \end{bmatrix}$.

Ogni altro sistema avrà nome R_i dove i indica il numero del veicolo corrispondente, in questo caso $i \in [2 \div 4]$. Il modello matematico è simile a quello descritto dall'equazione (4.6), con l'aggiunta di un uscita. Per cui il modello matematico in questo caso è:

$$\begin{cases} \dot{x}(t) = -Kx(t) - K \begin{bmatrix} 1 & -1 \end{bmatrix} u(t) \\ y(t) = \begin{bmatrix} -1 \\ 1 \end{bmatrix} x(t) + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} u(t) \end{cases} \tag{4.8}$$

L'uscita del sistema i-esimo y_i è la distanza tra il veicolo R_i e R_{i-1}, queste saranno anche le uscite del sistema complessivo. Mentre da ogni sistema R_i si ha un altra uscita (indicata con r_i in figura 4.6 e come secondo elemento del vettore y(s) nell'equazione (4.8)), che indica la posizione x_i, questa variabile è interna al sistema *Sp* e funge da ingresso u_2 del sistema $(i+1)-$esimo. La connessione tra due sistemi è rappresentata in figura 4.6, dove l'ingresso u_2 del primo veicolo è esterno al sistema complessivo.

Il modello complessivo (figura 4.7) ha:

- tre ingressi, che raggrupperemo nel vettore $u(t) = \begin{bmatrix} v(t) \\ u_1(t) \\ d_1(t) \end{bmatrix}$;
- quattro uscite, raggruppate nel vettore $y(t)$.

Per il modello matematico del sistema *Sp* si rimanda alla sezione successiva.

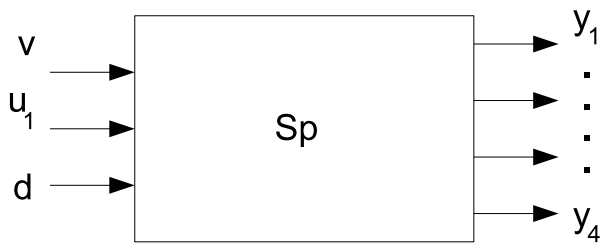

Figura 4.7: Sistema che modella l'intero plotone

4.3.1 Modello in variabili di stato

Definiamo il modello in sistema *Sp* in maniera formale:

- il sistema è di ordine n=4, con 3 ingressi e 4 uscite;
- $\mathbf{x}$ è il vettore di stato, con $\mathbf{x} : \mathbb{R} \to \mathbb{R}^4$;
- $\mathbf{y}$ è il vettore delle uscite, con $\mathbf{y} : \mathbb{R} \to \mathbb{R}^4$;
- $\mathbf{u}$ è il vettore di stato, con $\mathbf{u} : \mathbb{R} \to \mathbb{R}^3$.

Gli ingressi del sistema complessivo sono 3, in quanto:

- $u_1(t)$ è la velocità del primo veicolo $v(t)$, impostata come una costante;
- $u_2(t)$ è il set-point del singolo sistema R_i indicato precedentemente con $u_1(t)$ e ora rinominato u_2, corrispondente al valore della distanza desiderata tra i veicoli misurata in cm;
- $u_3(t)$ è d, valore di fondo scala della misura della distanza.

I tre vettori sono dunque:

$$\mathbf{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ u_3(t) \end{bmatrix} , \mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} . \text{ e } \mathbf{y}(t) = \begin{bmatrix} y_1 \\ y_2 \\ y_3(t) \\ y_4(t) \end{bmatrix} \tag{4.9}$$

La dinamica del sistema è rappresentata dal vettore:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \end{bmatrix}$$

L'equazione (4.10) descrive il modello del sistema totale.

$$\begin{cases} \dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ K & -K & 0 & 0 \\ 0 & K & -K & 0 \\ 0 & 0 & K & -K \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & -K & 0 \\ 0 & -K & 0 \\ 0 & -K & 0 \end{bmatrix} \mathbf{u} \\ \mathbf{y} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{u} \end{cases} \tag{4.10}$$

4.3.2 Stabilità e stati di equilibrio

Analizzeremo prima la stabilità col criterio degli autovalori, poi verrà eseguita la ricerca degli stati di equilibrio del sistema.

Criterio autovalori

Si consideri il sistema (4.10), e la sua matrice A, che per la definizione data del modello in VS è:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ K & -K & 0 & 0 \\ 0 & K & -K & 0 \\ 0 & 0 & K & -K \end{bmatrix}. \tag{4.11}$$

- Tale sistema è asintoticamente stabile se e solo se tutti gli autovalori della matrice **A** hanno parte reale negativa.
- Tale sistema è stabile se e solo se la matrice **A** non ha autovalori a parte reale positiva e gli eventuali autovalori a parte reale nulla hanno indice unitario (si veda *Analisi dei sistemi dinamici*[13] pag.112).

Il calcolo degli autovalori si ottiene risolvendo l'equazione:

$$det(\lambda I - A) = 0$$

La matrice è triangolare superiore, per cui gli autovalori della matrice sono gli elementi della diagonale. Gli autovalori sono $\lambda_1 = 0$ di molteplicità unitaria e $\lambda_2 = -K$ di molteplicità pari a 3, quindi il sistema è stabile ma non asintoticamente. Ciò implica:

- la stabilità dei punti di equilibrio;

- la stabilità del movimento, definito nella equazione (4.13).

Definizione 2 ***Movimento (o moto).*** *Si consideri il sistema dinamico*

$$\dot{x}(t) = f(x(t)), \tag{4.12}$$

e sia $x(t)$ la sua evoluzione a partire la sua evoluzione a partire da uno stato iniziale $x(0) = x_0$. Definiamo ***movimento*** *(o moto) del sistema associato allo stato iniziale x_0, il sottoinsieme di $\mathbb{R}^n \times \mathbb{R}$ dato da*

$$m(x_0) = (\bar{x}, t) | t \geq 0, x(t) = \bar{x}. \tag{4.13}$$

Stati di equilibrio

Per il calcolo dei punti di equilibrio forniamo prima una breve definizione.

Definizione 3 ***Stati di equilibrio*** *Uno stato x_e è uno stato di equilibrio o, equivalentemente un punto di equilibrio per un sistema in VS, se vale la seguente condizione:*

$$x(\tau) = x_e \;\Rightarrow\; (\forall t \geq \tau)\; x(t) = x_e,$$

ovvero se ogni traiettoria che parte da x_e in un generico istante τ resta in x_e in ogni istante successivo. Da un punto di vista strettamente matematico il calcolo degli stati di equilibrio si riduce a trovare i valori di x che rendono nulla la $\dot{x}$.

Per il sistema (4.10) il vettore x_e è soluzione dell'equazione:

$$\mathbf{A}x_e + \mathbf{B}u = 0. \tag{4.14}$$

Dove B è la matrice:

$$\begin{bmatrix} 1 & -K & 0 \\ 0 & -K & 0 \\ 0 & -K & 0 \\ 0 & -K & 0 \end{bmatrix}.$$

Quindi il calcolo dei punti di equilibrio si riduce alla soluzione dell'equazione (4.14). Possiamo svolgere i calcoli in due casi diversi:

- veicolo capofila fermo, gli altri in movimento;
- veicolo capofila in movimento, gli altri veicoli in movimento.

Veicolo capofila fermo Se il primo veicolo è fermo non è necessario modellare anche il suo funzionamento ma, dato che la posizione è nota e costante nel tempo verrà impostata come un parametro in ingresso. Si indica con x_l la posizione del primo veicolo. Posto $v_l = 0$ e $x_1 = x_l$ i valori della velocità e della posizione del leader, posso riscrivere la (4.10) nel seguente modo:

$$\begin{cases} \dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -K & 0 & 0 \\ 0 & K & -K & 0 \\ 0 & 0 & K & -K \end{bmatrix} \mathbf{x} + \begin{bmatrix} K \\ 0 \\ 0 \end{bmatrix} x_l + \begin{bmatrix} 0 & 0 & 0 \\ 0 & -K & 0 \\ 0 & -K & 0 \\ 0 & -K & 0 \end{bmatrix} \mathbf{u} \\ \mathbf{y} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{u} \end{cases} \tag{4.15}$$

È possibile eliminare la prima riga delle matrici A, B e C del sistema (4.10) e anche la prima colonna delle matrici A e C dello stesso sistema , in modo da eliminare la modellizzazione del primo veicolo e includere la sua posizione nella modellizzazione del secondo veicolo. La matrice D si può eliminare totalmente. Nell'equazione (4.15) si vede che lo stato di x_1 è utile solo nel calcolo della dinamica del secondo veicolo, ed essendo lo stato x_1 costante, e uguale a x_l è possibile passarlo come parametro in ingresso senza che sia richiesta la modellizzazione del primo veicolo. Per cui il sistema (4.15) si semplifica così:

$$\begin{cases} \dot{\hat{\mathbf{x}}} = \begin{bmatrix} -K & 0 & 0 \\ K & -K & 0 \\ 0 & K & -K \end{bmatrix} \hat{\mathbf{x}} + \begin{bmatrix} K \\ 0 \\ 0 \end{bmatrix} x_l + \begin{bmatrix} -K \\ -K \\ -K \end{bmatrix} \hat{\mathbf{u}} \\ \hat{\mathbf{y}} = \begin{bmatrix} -1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \hat{\mathbf{x}} \end{cases} \tag{4.16}$$

Dove i nuovi vettori del modello (4.16) sono:

$$\hat{\dot{\mathbf{x}}} = \begin{bmatrix} \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} \quad \hat{\mathbf{x}} = \begin{bmatrix} x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad \hat{\mathbf{y}} = \begin{bmatrix} y_2 \\ y_3 \\ y_4 \end{bmatrix} \quad \hat{\mathbf{u}} = u_1$$

Gli autovalori della matrice $\hat{A}$ sono tre coincidenti con $\lambda_1 = -K$, quindi ne possiamo dedurre che il sistema è asintoticamente stabile.

Il calcolo dei punti di equilibrio richiede la soluzione di:

$$\hat{A}\hat{\mathbf{x}} + \begin{bmatrix} K \\ 0 \\ 0 \end{bmatrix} x_l - \begin{bmatrix} K \\ K \\ K \end{bmatrix} \hat{\mathbf{u}} = 0$$

$$\hat{A}\hat{\mathbf{x}} = \begin{bmatrix} -K \\ 0 \\ 0 \end{bmatrix} x_l + \begin{bmatrix} K \\ K \\ K \end{bmatrix} \hat{\mathbf{u}}$$

e di conseguenza l'inversione della matrice $\hat{A}$.

Per il calcolo della matrice inversa in forma parametrica si è utilizzato il metodo di Gauss-Jordan[3]. Prima di tutto si è costruita la matrice

$$G = [A\ I]$$

e si è proceduto alla combinazione lineare delle righe della matrice G fino a quando non si è ottenuto la matrice G in forma $G = [I\ A^{-1}]$. La matrice $\hat{A}^{-1}$ è:

$$\begin{bmatrix} -\frac{1}{K} & 0 & 0 \\ -\frac{1}{K} & -\frac{1}{K} & 0 \\ -\frac{1}{K} & -\frac{1}{K} & -\frac{1}{K} \end{bmatrix} \tag{4.17}$$

Si sostituisce la (4.17) nalla seguente equazione:

$$\hat{\mathbf{x}}_e = \hat{A}^{-1} \left(\begin{bmatrix} -K \\ 0 \\ 0 \end{bmatrix} x_l + \begin{bmatrix} K \\ K \\ K \end{bmatrix} u_1 \right) \tag{4.18}$$

ottenendo risultato:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_l \\ x_l - u_1 \\ x_l - 2u_1 \\ x_l - 3u_1 \end{bmatrix}. \tag{4.19}$$

Il risultato è esattamente quello desiderato, se si considera che nel modello il veicolo ha dimensioni trascurabili.

Veicolo capofila in movimento Nel caso il veicolo capofila sia in movimento, la sua velocità lineare v_l è diversa da zero e non è possibile fare le semplificazioni che si sono fatte in precedenza.

È richiesta l'inversione della matrice A come espressa in equazione (4.11), che essendo singolare non potrà essere invertita. Per capire se il sistema ammette infinite soluzioni o nessuna, si è analizzato il sistema scritto nella forma:

$$Ax_e = -Bu \tag{4.20}$$

e si è posto $B' = -Bu$. Scritta la matrice M, come:

$$M = [A\,|B']$$

si ha che:

- se $rank(A) < rank(M)$ il sistema (4.20) non ammette soluzioni;
- se $rank(A) = rank(M)$ il sistema (4.20) ammette infinite soluzioni.

Nel nostro caso

$$M = \left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & -v_l \\ K & -K & 0 & 0 & Ku_1 \\ 0 & K & -K & 0 & Ku_1 \\ 0 & 0 & K & -K & Ku_1 \end{array}\right]$$

il $rank(A) = 3$ e $rank(M) = 4$, di conseguenza il sistema non ammette soluzioni, non esisteranno punti di equilibrio. Ciò non preclude la stabilità in quanto come stabilito col criterio degli autovalori, il sistema è stabile. Quindi non essendoci stati di equilibrio si parlerà di stabilità del movimento.

Definizione 4 ***Movimento stabile.*** *Si consideri il sistema* (4.12)*. Sia* $x(t)$ *la sua evoluzione a partire dallo stato* x_0 *e* $\tilde{x}(t)$ *la sua evoluzione a partire dallo stato* $\tilde{x}_0$*.*

Il movimento m_0 *è detto stabile se per ogni* $\epsilon > 0$ *esiste un* $\delta(\epsilon) > 0$ *tale che* $||\tilde{x}_0 - x_0|| \leq \delta(\epsilon)$*, allora* $||\tilde{x}(t) - x(t)|| \leq \epsilon$ *per ogni* $t \geq 0$*. In caso contrario* m_0 *è detto instabile.*

Un movimento m_0 *è detto stabile se perturbando lo stato da cui l'evoluzione* $x(t)$ *ha origine, mantenendosi però sufficientemente vicino a tale stato, si ottiene una nuova evoluzione* $\tilde{x}(t)$ *che in ogni istante* $t \geq 0$ *è arbitrariamente prossima all'evoluzione di* $x(t)$*.*

Capitolo 5

Algoritmo di controllo

In questo capitolo è descritto il tipo di controllo implementato sui veicoli separatamente per ogni funzionalità. Il sistema di controllo implementato è un sistema di controllo distribuito. Questo implica che tutti i robot hanno lo stesso sistema di controllo che rende il loro comportamento autonomo.

5.1 Movimento circolare

Questa sezione spiegherà l'algoritmo utilizzato per realizzare la parte del progetto denominata in letteratura come *path following* o inseguimento del percorso desiderato. Nel nostro caso il percorso da seguire è di forma circolare o ellittica, da cui il titolo della sezione, *Movimento circolare*.

Lo stato in cui il veicolo può trovarsi sono principalmente due:

- il veicolo è nel percorso desiderato,
- il veicolo è fuori dal percorso desiderato.

La distinzione tra questi due stati è data dalla scena vista dai sensori IR numero 10 e 11[1], quelli posizionati sotto il robot. La pista è individuabile in base al contrasto dato dal colore nero della pista al colore bianco dello sfondo (si veda figura 5.1 dove la forma del robot è stata modificata per far capire meglio la direzione di marcia).

5.1.1 Dentro il percorso desiderato

Supponiamo, innanzitutto, che il veicolo si trovi già nel percorso da seguire. Quando i due sensori IR numero 10 e 11 vedono la pista allora il

[1] Si veda figura 6.1

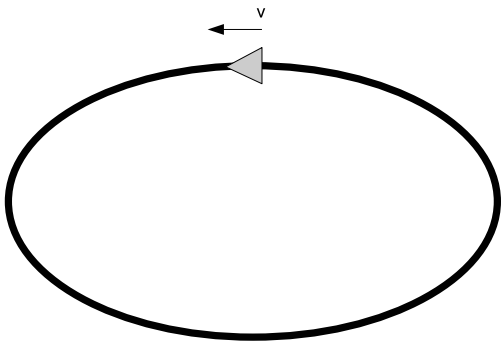

Figura 5.1: Modello della pista

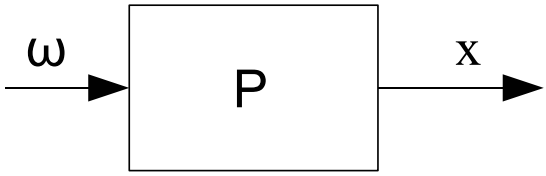

Figura 5.2: Sistema di controllo per l'inseguimento del percorso.

veicolo capisce di essere nel percorso desiderato. Ora il comportamento del robot deve essere tale da garantire la sua permanenza all'interno del percorso per un tempo indefinito. Dato che il movimento del veicolo stesso causa un aggiornamento della scena sarà necessario un controllo per adeguarsi a tali cambiamenti.

L'algoritmo implementato prevede un controllo che agisce sulla sola velocità angolare lasciando costante, e impostata come parametro, la velocità lineare di crociera lungo il percorso.

La velocità angolare sarà controllata mediante un controllo a ciclo aperto, come illustrato in figura 5.2, dove in ingresso si ha la velocità angolare ω e in uscita la posizione del veicolo.

Il sistema completo P è descritto nel seguente algoritmo.

Algoritmo 5 *Il valore di ω cambia a seconda della situazione:*

- *se entrambi i sensori sono dentro la pista, la velocità angolare sarà nulla;*
- *se il sensore destro è fuori dalla pista mentre il sinistro è dentro la pista, la velocità angolare dovrà impartire una rotazione in senso antiorario ($w > 0$);*

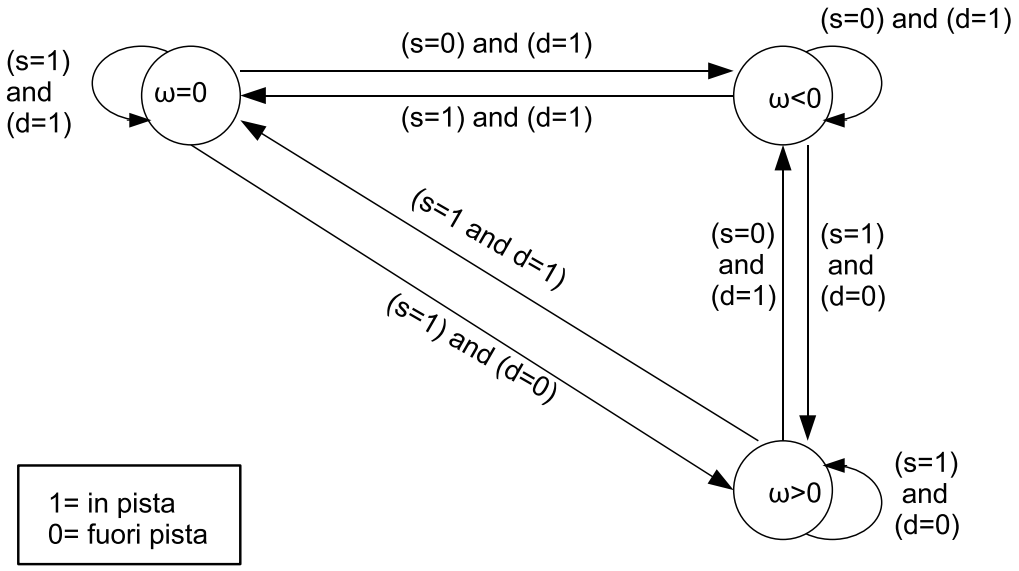

Figura 5.3: Algoritmo per restare nel percorso.

- *se il sensore sinistro è fuori dalla pista mentre il destro è dentro la pista, la velocità angolare dovrà impartire una rotazione in senso orario* ($w < 0$).

In figura 5.3 è rappresentato l'algoritmo precedentemente descritto tramite una macchina a stati finiti. Viene fatta l'ipotesi che l'unico stato iniziale permesso è corrispondente al veicolo dentro il percorso, con almeno uno dei due sensori. E quindi nell'algoritmo illustrato in figura 5.3 tutti gli stati appartengono all'insieme degli stati iniziali. Gli eventi sono:

- s rappresenta il sensore sinistro, se questo è nel percorso avrà valore 1, se è fuori dal percorso avrà valore 0;

- d rappresenta il sensore destro, se questo è nel percorso avrà valore 1, se è fuori dal percorso avrà valore 0.

I passaggi di stato sono dettati dalla combinazione dei valori di s e d. Si suppone che una volta all'interno della pista non sia possibile uscirne, per questo gli eventi non possono assumere contemporaneamente il valore 0. Il passaggio di stato si ha ad ogni periodo di campionamento solo se il valore delle variabili s e d è cambiato rispetto al periodo precedente.

Gli stati sono stati etichettati con la funzione che regola la velocità angolare:

- stato $w = 0$, velocità angolare nulla;
- stato $w < 0$, velocità angolare minore di zero;
- stato $w > 0$, velocità angolare maggiore di zero.

La rotazione da impartire dipenderà dal tempo di campionamento (T) utilizzato, dal raggio di curvatura[2] della pista e dalla velocità lineare (v) a cui il robot sta andando.

Si è deciso il valore della velocità angolare per via sperimentale, e l'algoritmo utilizzato è il seguente.

Algoritmo 6 *Si è scelto il periodo di campionamento in grado di permettere al processore di svolgere tutti i calcoli richiesti e che garantisca un aggiornamento dei valori dei sensori;*

1. *si è impostata una velocità lineare costante, si parte da un valore alto*[3] *che andrà via via ridotto sino a trovare il valore massimo ammissibile;*
2. *si è scelta una velocità angolare;*
3. *è stata eseguita la prova sulla pista:*
   (a) *se il veicolo esce dalla pista il problema è la velocità lineare troppo alta per quel tempo di campionamento. Dato che il tempo di campionamento non può essere ridotto, si modifica il valore della velocità lineare, si ricomincia dal punto 1;*
   (b) *se il veicolo oscilla, la velocità angolare è talmente elevata da far spostare il veicolo tra i due estremi del bordo della pista. Si deve ricominciare dal punto 2;*
   (c) *se il veicolo impiega un tempo troppo elevato per orientarsi correttamente lungo il percorso il valore della velocità angolare è troppo basso, si deve ricominciare dal punto 2;*
   (d) *se il veicolo si muove come desiderato, abbiamo trovato:*
      - *il valore di velocità lineare massimo ammissibile per questo periodo di campionamento;*
      - *il valore della velocità angolare ottimale.*

[2]Il raggio di curvatura è il raggio della circonferenza, detta *cerchio osculatore.*, che meglio approssima la curva

[3]Nei limiti fisici del veicolo.

Alla fine dell'algoritmo il valore di velocità lineare trovato fissa il limite massimo del moto all'interno della pista. Tale valore può essere ridotto a piacimento senza intaccare il funzionamento del sistema. Il valore di velocità angolare trovato invece è quello ottimale per la pista in questione, è dipendente fortemente dallo spessore della striscia che traccia il percorso ma anche dal raggio di curvatura del percorso stesso.

5.1.2 Fuori dal percorso desiderato

Se il veicolo non si trova dentro la pista esso non potrà sapere a che distanza si trova da essa, nè in che direzione deve dirigersi. È necessario implementare un algoritmo di ricerca della pista. Le soluzioni possono essere varie. Si può, ad esempio, implementare un movimento pseudo-random fintanto che il robot non trova la pista. Nel nostro caso si suppone che se il robot non è nel percorso desiderato allora si troverà al centro della pista, questo perché la pista è agli estremi dello spazio disponibile. Per tale situazione un comportamento random non è la soluzione migliore da seguire per la ricerca della pista, in quanto si potrebbe finire in una sorta di loop in cui il robot non trova mai la pista. È stato invece impostato come algoritmo per la ricerca della pista un semplice moto rettilineo, il quale permette sempre il raggiungimento della pista, anche se con tempi variabili a seconda della posizione iniziale del veicolo.

Una volta che il veicolo ha raggiunto la pista è importante che questo si riposizioni lungo il percorso. Per implementare tale funzionalità è stata seguita prima una strada, che successivamente ha richiesto una modifica sostanziale per rendere il sistema più robusto.

Calcolo angolo rientro in pista (I)

L'algoritmo che posiziona il veicolo lungo la pista si basa sulla geometria descritta in figura 5.5 dove:

- L è la larghezza della pista;
- D è lo spazio percorso per attraversare tutta la pista;
- α è l'angolo di ingresso in pista.

Algoritmo 7 *Il calcolo dell'angolo di rientro in pista e successivo posizionamento corretto richiede i seguenti passi:*

1. *il veicolo (che nell'istante di campionamento precedente era fuori dalla pista) rileva la pista con un sensore, salva in una variabile temporanea (t_i) l'istante di tempo ;*

2. *con velocità lineare costante v e velocità angolare nulla il veicolo attraversa trasversalmente la linea nera della pista;*

3. *quando vede lo sfondo bianco con lo stesso sensore con cui è stata individuata la pista, salva in un altra variabile (t_f) il tempo ;*

4. *calcolo il tempo di attraversamento della pista t_{att}=t_f - t_i;*

5. *conoscendo la velocità di attraversamento v posso ricavare la distanza percorsa per attraversare la pista: $D = t_{att} \cdot v$;*

6. *nota la larghezza della pista L, l'angolo di ingresso in pista è pari ad $\alpha = \arcsin(\frac{D}{L})$;*

7. *si imposta una velocità angolare pari ad α per un tempo pari a un secondo, e velocità lineare nulla, per consentire il corretto posizionamento in pista.*

L'algoritmo è illustrato in figura 5.4, dove:

- stato 0, veicolo fuori della pista, attua un moto rettilineo uniforme sino al rientro in pista;
- stato 1, vista della pista col sensore sinistro, attua un moto rettilineo uniforme sino a quando non uscirà dalla pista con lo stesso sensore;
- stato 2, vista della pista col sensore destro, attua un moto rettilineo uniforme sino a quando non uscirà dalla pista con lo stesso sensore;
- stato 3, calcolo dell'angolo di ingresso in pista;
- stato 4, il controllore attua una velocità angolare pari all'angolo calcolato nello stato 3 per un intervallo di tempo di un secondo.

Calcolo angolo rientro in pista (II)

Il precedente algoritmo, se pur funzionante, ha il difetto di essere dipendente dalle dimensioni della pista, per cui si è studiato un approccio più robusto per svolgere gli stessi calcoli. Si è sostituito la larghezza della pista con la distanza tra i sensori. L'algoritmo è illustrato in figura 5.7. Il tempo di inizio e il tempo di fine sono stati presi nel seguente modo:

- t_i è l'istante di tempo in cui uno dei sensori vede la pista;
- t_f è l'istante di tempo in cui entrambi i sensori sono sopra la pista.

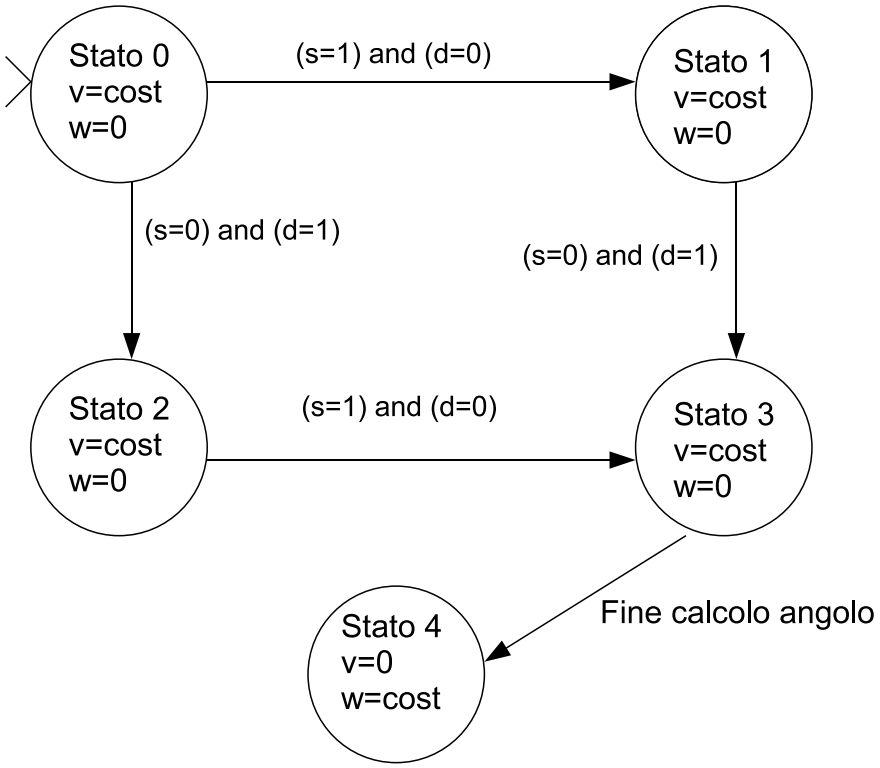

Figura 5.4: Primo algoritmo per il rientro in pista.

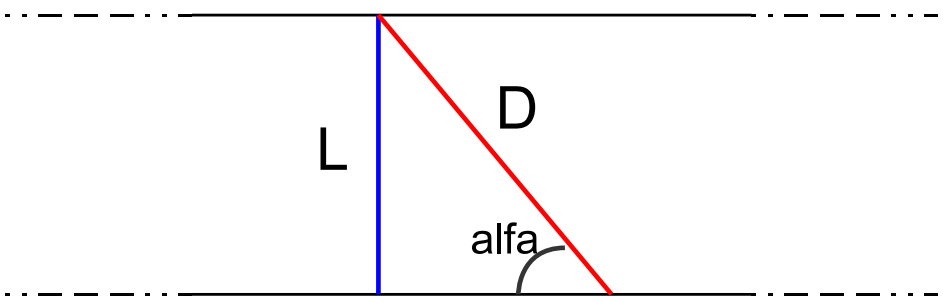

Figura 5.5: Geometria per il calcolo dell'angolo di rientro in pista.

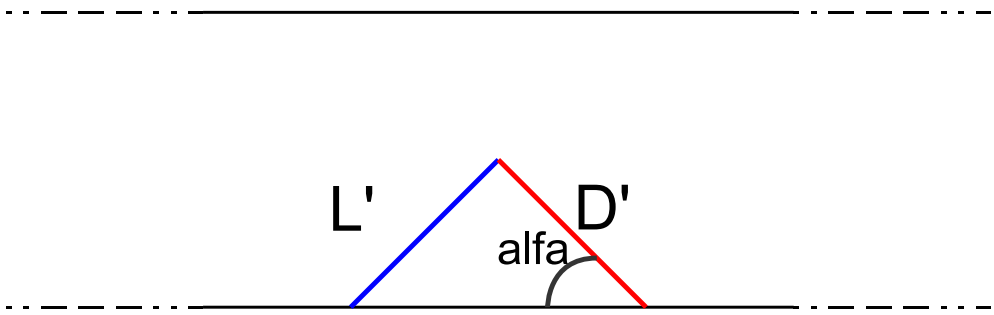

Figura 5.6: Calcolo angolo ingresso in pista in modo corretto.

Il calcolo dell'angolo è svolto esattamente come nel caso precedente, solo che la distanza L non è più la larghezza della pista ma la distanza tra i due sensori IR 10 e 11 (di circa 16 mm) che indicheremo con L', e la distanza percorsa all'interno della pista la chiameremo D'. Per attuare una rotazione pari all'angolo α, calcolato come nel paragrafo precedente, si impone una velocità angolare pari in valore ad α per un intervallo di tempo di un secondo.

In figura 5.6 sono illustrati i nuovi parametri su cui è impostato l'algoritmo corretto:

- L' è la distanza tra i due sensori, $L' = 16mm$;
- D' è la distanza percorsa all'interno della pista calcolata dalla formula:

$$D = t_{att} \cdot v;$$

- l'angolo α è pari $\alpha = \arcsin(\frac{D'}{L'})$;
- la velocità angolare attuata sarà: $w = \alpha rad/s$ e verrà mantenuta per un intervallo di tempo di un secondo in modo da compiere l'intera rotazione.

In questa seconda soluzione il sistema non è più dipendente dalle dimensioni della pista in quanto i calcoli vengono fatti in base alle caratteristiche geometriche del robot.

5.1.3 Algoritmo complessivo

I due algoritmi, quello che riguarda il comportamento fuori dalla pista e quello che riguarda il comportamento in pista, sono stati integrati in unico

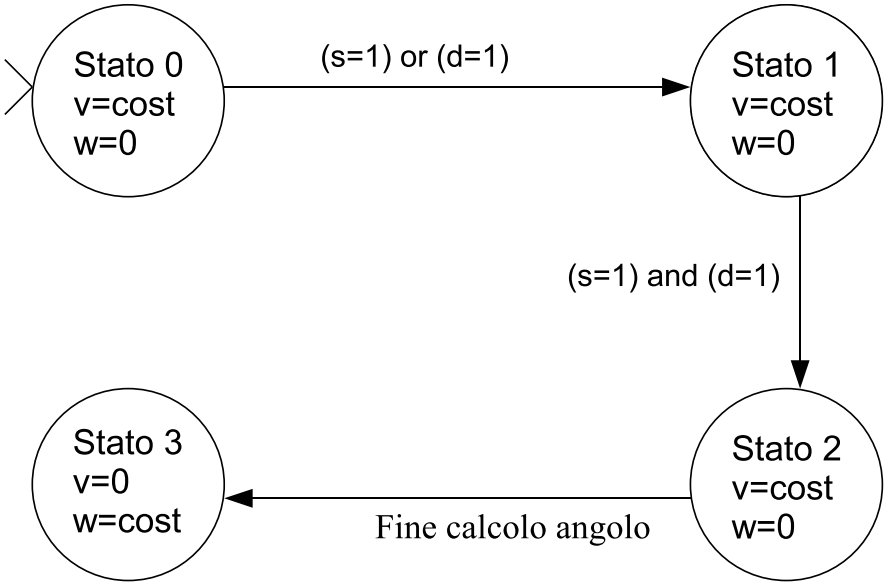

Figura 5.7: Secondo algoritmo per il rientro in pista

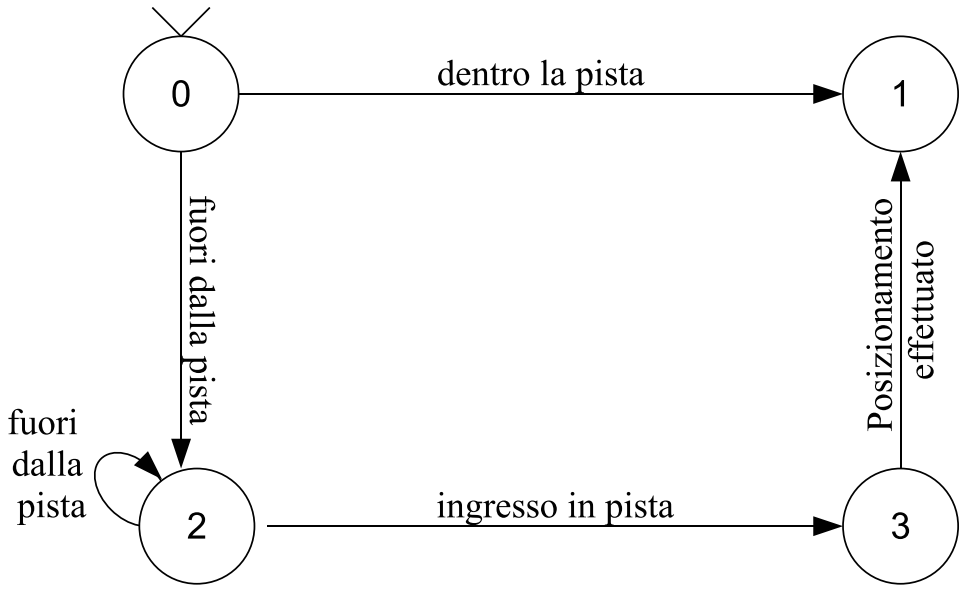

Figura 5.8: Grafo algoritmo per l'inseguimento del percorso

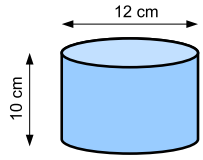

Figura 5.9: Ostacolo

algoritmo. In figura 5.8 è illustrato un grafo che riassume i due algoritmi per l'inseguimento del percorso.

- Lo stato 0 indica lo stato iniziale di decisione, in cui il veicolo sonda la scena per capire se si trova o meno all'interno della pista designata. Se si trova fuori dalla pista va nello stato 2, altrimenti nello stato 1;
- Lo stato 1 corrisponde al corretto posizionamento in pista, per cui è anche lo stato finale dell'algoritmo;
- Lo stato 2 corrisponde al comportamento che il veicolo adotta quando sta cercando la pista. Se non trova la pista resta nello stato 2, altrimenti passa allo stato 3;
- Lo stato 3 è uno stato di appoggio che serve per il corretto posizionamento del veicolo nella pista, finito il posizionamento il processo torna nello stato 1.

5.2 Algoritmo di controllo per l'ostacolo

Il fine ultimo dell'aggiramento degli ostacoli è quello di aggirare un eventuale veicolo che si è fermato per un malfunzionamento. Si è pensato di costruire un ostacolo di forma cilindrica delle stesse dimensioni del robot, che funga da veicolo fermo. Il cilindro ha un diametro di 12 cm circa e un altezza di 10 cm, si veda figura 5.9. Tale ostacolo è individuabile da tutti i sensori del veicolo, per cui sarà possibile capire l'angolo di vista dell'ostacolo. Il veicolo non potrà capire se ciò che ha davanti è un ostacolo o un altro veicolo in movimento, per cui in una prima fase ci occuperemo solo dell'aggiramento degli ostacoli escludendo la possibilità di incontrare altri veicoli in movimento.

5.2.1 Come evitare gli ostacoli

Il primo algoritmo realizzato ha lo scopo di evitare, senza aggirare, gli ostacoli che si presentano lungo il cammino del veicolo.

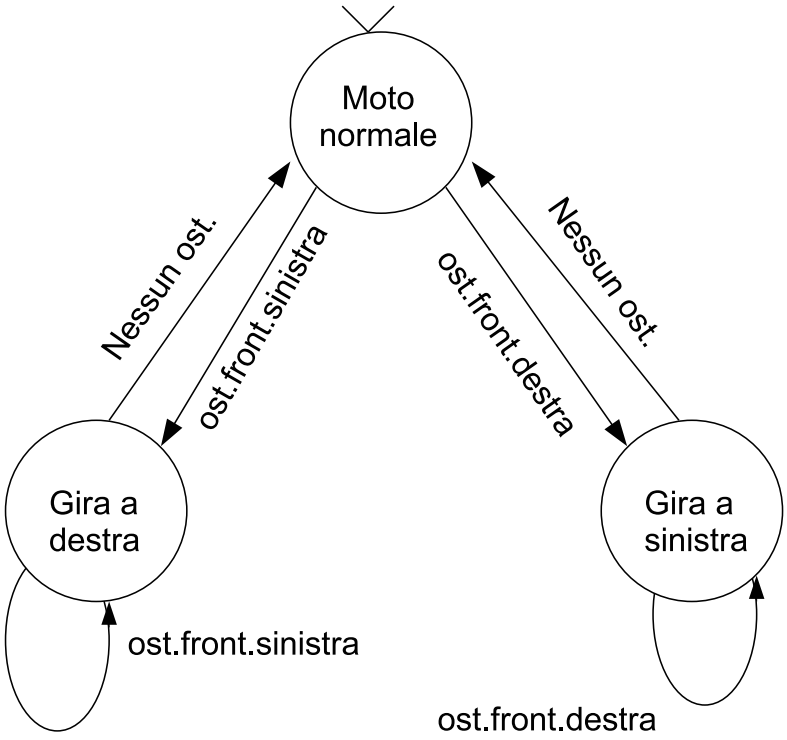

Figura 5.10: Rappresentazione dell'algoritmo per evitare un ostacolo

Si è creato un algoritmo per evitare l'ostacolo nel caso di moto rettilineo uniforme del robot, ma il moto del veicolo in assenza di ostacoli non è rilevante ai fini dell'algoritmo.

Algoritmo 8 *Se il veicolo vede un ostacolo davanti a se:*

1. *calcola da che lato l'ostacolo risulta più vicino, se:*
   - *l'ostacolo è più vicino da sinistra passo al passa alla fase 2;*
   - *l'ostacolo è più vicino da destra passo al passa alla fase 3;*
2. *il veicolo gira a destra fino a quando davanti a sè non ci sarà nessun ostacolo, l'algoritmo termina qui;*
3. *il veicolo gira a sinistra fino a quando davanti a sè non ci sarà nessun ostacolo, l'algoritmo termina qui.*

L'algoritmo 8 è illustrato in figura 5.10.

Una volta incontrato un ostacolo nel suo percorso deciderà se continuare il suo cammino da sinistra o da destra, a seconda della vicinanza dell'oggetto. Se l'ostacolo è più vicino da sinistra, allora il robot girerà a destra; se l'ostacolo è più vicino da destra allora il veicolo girerà a sinistra. Questo algoritmo ha lo scopo di evitare un solo ostacolo isolato in uno spazio vasto.

Si è ampliato l'algoritmo precedente per far si che se il veicolo si trovi tra due ostacoli, uno alla sua destra e l'altro alla sua sinistra, allora sia in grado di valutare se ci sia spazio sufficiente per passare in mezzo agli ostacoli o sia necessario cercare una via migliore. Questo algoritmo, come il precedente, si basa sulla sola scena che si presenta davanti al robot e non su quello che il robot si lascia alle spalle. La dimensione dell'ostacolo è nota a priori, mentre non ne è nota la posizione. Se i due ostacoli, quello a destra e quello a sinistra della direzione di marcia del veicolo, sono posti ad un distanza tra loro che è superiore alle dimensioni del robot più una zona di sicurezza, allora il veicolo potrà passare in mezzo agli ostacoli.

In termini più formali l'algoritmo è il seguente:

Algoritmo 9 *1. Se il veicolo vede un solo ostacolo davanti a sè esegue l'algoritmo 8, se vede più di un ostacolo si passa al punto 2;*

2. calcola se tra i due ostacoli c'è spazio sufficiente per passare, se la risposta è affermativa passa alla fase 3 in caso contrario passa alla fase 4;

3. il veicolo passa in mezzo agli ostacoli facendo in modo che la sua distanza dall'ostacolo a sinistra sia uguale alla sua distanza dall'ostacolo a destra, quando non avrà più ostacoli vicino a sè l'algoritmo termina qui;

4. il veicolo gira a sinistra fino a quando davanti a sè non ci sarà nessun altro ostacolo, l'algoritmo termina qui.

5.2.2 Come aggirare gli ostacoli

In questa sezione verrà descritto il processo per aggirare l'ostacolo. L'algoritmo per aggirare l'ostacolo comprende un sistema più complesso di quello per l'inseguimento della pista. La realizzazione di questo algoritmo è stato diviso in diverse fasi, dove ognuna migliora e aggiunge funzionalità alle precedenti.

Fase I

Nella prima fase di realizzazione dell'algoritmo si è realizzato:

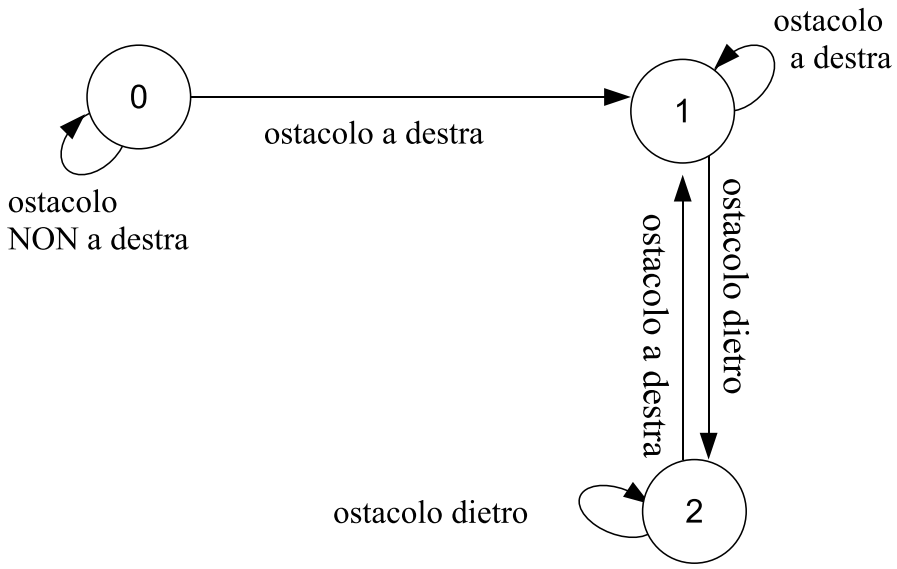

Figura 5.11: Grafo algoritmo per aggirare l'ostacolo

- il comportamento base del veicolo che deve sorpassare;
- le funzionalità generali che consentono ad un veicolo, che si trova da solo nel percorso desiderato, di evitare un ostacolo dentro la pista.

Supponiamo in questa fase che il veicolo abbia spazio libero intorno a se e che non si aspetti di trovare altri veicoli in movimento ma solo un possibile ostacolo.

L'algoritmo per aggirare l'ostacolo è diviso in tre parti (figura 5.11):

- Stato 0, il veicolo si gira a sinistra per lasciare l'ostacolo alla sua destra, quando l'ostacolo sarà alla sua destra allora passerà allo stato 1;
- Stato 1, il veicolo andrà dritto (velocità lineare costante e velocità angolare nulla) fino a quando avrà l'ostacolo alla sua destra, quando l'ostacolo sarà dietro di esso allora passerà allo stato 2;
- Stato 2, il veicolo girerà a destra (velocità lineare costante e velocità angolare costante) fino a quando non vedrà nuovamente l'ostacolo alla sua destra, quando l'ostacolo sarà alla sua destra allora passerà allo stato 1;

È chiaro che l'algoritmo finisce in un loop e il veicolo si trova ad aggirare indefinitamente l'ostacolo se non vi è la possibilità di uscire dal loop. Questo

non è un problema perché questo algoritmo è stato studiato appositamente per aggirare un ostacolo che si trova lungo il percorso stabilito, per cui durante la fase di sorpasso il veicolo uscirà dalla pista e aggirerà l'ostacolo. Una volta fuori dalla pista, l'algoritmo dovrà avere un doppio controllo: da un lato verificare che l'ostacolo resti sempre alla destra del robot, dall'altro cercare la pista per posizionarsi nel percorso prestabilito.

In un ottica più generale dell'intero sistema si può vedere l'algoritmo del sorpasso come una sub-routine del programma principale del movimento circolare. In tale sub-routine vi sarà una ricerca continua del percorso principale, indicato come rientro in pista la quale comporta l'uscita dalla subroutine di sorpasso e ripresa del funzionamento nominale.

La formulazione dell'algoritmo in termini più formali è la seguente:

Algoritmo 10 ***Aggiramento ostacolo.*** *Quando viene individuato l'ostacolo frontalmente al veicolo:*

1. *il veicolo ruota verso sinistra sino a quando l'ostacolo non sarà alla sua destra, se l'ostacolo è alla sua destra passa alla fase 2;*

2. *il veicolo si muoverà in avanti sino a quando non riuscirà più a vedere l'ostacolo alla sua destra,se non è rientrato nel percorso si passa alla fase 3 altrimenti alla fase 4;*

3. *il veicolo ruoterà verso destra sino a quando non ci sarà nuovamente l'ostacolo in vista nel suo lato destro, se non è rientrato nel percorso si passa alla fase 2 altrimenti alla fase 4;*

4. *il veicolo ha individuato il percorso da seguire, prima di riprendere il suo moto nel percorso dovrà calcolare l'angolo di rientro in pista e orientarsi correttamente nel percorso, come spiegato nel paragrafo 5.1.2, si esce dall'algoritmo.*

Fase II

In questa fase della realizzazione dell'algoritmo per il sorpasso, l'ostacolo presente non è fisso. Si suppone che l'ostacolo sia un veicolo fermo per un malfunzionamento e che quindi possa riprendere a muoversi in un qualsiasi istante. Ciò richiede un controllo continuo della scena intorno al veicolo in fase di sorpasso.

L'algoritmo si basa su quello implementato nella **Fase I**. Lo studio è avvenuto in due fasi:

1. si è studiato in che modo il robot capisce che l'ostacolo si è mosso e non è più di intralcio al suo moto;

2. si è studiato il comportamento che il robot deve seguire nel caso di sparizione dell'ostacolo.

Sparizione dell'ostacolo Con la dicitura "scomparsa dell'ostacolo" si intende la ripresa del moto da parte del veicolo (che qui chiameremo *veicolo O*) che precedentemente era fermo per un malfunzionamento. Il malfunzionamento più banale è la scarica della batteria, ma può capitare anche un blocco della routine del programma dovuto alla richiesta delle risorse da parte del Sistema Operativo del Korebot.

Il *veicolo O*, dopo essere rimasto fermo per un intervallo di tempo tale da far credere agli altri veicoli che esso è un ostacolo da aggirare, riprende a muoversi con un comportamento dettato dalla sua posizione. Si possono verificare diversi casi:

- il *veicolo O* è fermo lungo la pista;
- il *veicolo O* si è fermato nella prima fase del sorpasso (stato 0);
- il *veicolo O* si è fermato in una delle ultime fasi del sorpasso (stato 1 e 2).

Nei primi due casi il robot è fermo sulla pista, nell'ultimo è fermo fuori dalla pista ma comunque potrebbe essere abbastanza vicino da ostacolare il moto del veicolo. A seconda dello stato in cui si è fermato la ripresa del moto cambia:

- nel primo caso, il *veicolo O* riprende il moto lungo il percorso;
- nel secondo caso il *veicolo O* prima ruota per riposizionarsi lungo il percorso poi riprende il movimento circolare;
- nell'ultimo caso il *veicolo O* cerca la pista in base all'algoritmo 11.

In tutti questi casi si ha che il *veicolo O* si muove, e cambia il suo stato da ostacolo a veicolo in movimento. Il veicolo generico che ha iniziato il sorpasso del *veicolo O* si accorge che questo si è mosso perché:

- la distanza tra esso e l'ostacolo è cambiata;
- non c'è nessun ostacolo dentro il raggio di vista.

Basare la cognizione della sparizione dell'ostacolo solo sulla variazione della distanza non è un approccio corretto in quanto il movimento stesso del veicolo causa un allontanamento dal *veicolo O*. Per quanto riguarda il fatto che il veicolo non veda nulla all'interno del suo raggio di vista non basta per assumere con certezza che l'ostacolo è sparito.

Si è scelto di identificare la sparizione dell'ostacolo con la seguente dicitura:

- per un certo intervallo di tempo non vi sono ostacoli all'interno del raggio di vista del veicolo.

Comportamento per la sparizione dell'ostacolo Nel caso di sparizione dell'ostacolo è necessario che nell'algoritmo ci sia una memoria degli avvenimenti precedenti in modo da poter ripercorrere i passi fatti. Sono state aggiunte delle modifiche all'algoritmo 10.

Algoritmo 11 ***Sparizione ostacolo.*** *Dopo che è stato individuato l'ostacolo il veicolo dovrà eseguire le seguenti operazioni:*

1. *il veicolo ruota verso sinistra sino a quando l'ostacolo non sarà alla sua destra, nella variabile temp_sx è memorizzato l'angolo di rotazione, se l'ostacolo è alla sua destra passa alla fase 2, se l'ostacolo è sparito passa alla fase 5;*

2. *il veicolo si muoverà in avanti sino a quando non riuscirà più a vedere l'ostacolo alla sua destra, se è sparito l'ostacolo si passa alla fase 6, se non è rientrato nel percorso si passa alla fase 3 altrimenti alla fase 4;*

3. *il veicolo ruoterà verso destra sino a quando non ci sarà nuovamente l'ostacolo in vista nel suo lato destro, nella variabile temp_dx verrà memorizzato l'angolo di rotazione verso destra, se è sparito l'ostacolo si passa alla fase 6, se non è rientrato nel percorso si passa alla fase 2 altrimenti alla fase 4;*

4. *il veicolo ha individuato il percorso da seguire, prima di riprendere il suo moto nel percorso dovrà calcolare l'angolo di rientro in pista e orientarsi correttamente nel percorso, come spiegato nel paragrafo 5.1.2, si esce dall'algoritmo.*

5. *il veicolo ruoterà verso destra di una quantità pari al valore memorizzato in temp_sx, se è tornato in pista passa alla fase 4, se è ricomparso l'ostacolo si passa alla fase 0;*

6. *verrà calcolato l'angolo complessivo di rotazione dato dai valori memorizzati dalle variabili temp_sx e temp_dx, se l'angolo è diverso da zero si passa alla fase 7 altrimenti alla fase 8;*

7. *verrà attuato una rotazione uguale ma di verso opposto rispetto alla rotazione complessiva calcolata al passo 6, se è tornato in pista passa alla fase 4 altrimenti passa alla fase 8 .*

8. *il veicolo attua un moto rettilineo sino al ritrovamento della pista, si passa alla fase 4.*

La memorizzazione del percorso fatto per sorpassare è sotto forma di rotazione del veicolo. Non è stata implementata nessuna funzione di odometria ma solo una memorizzazione delle rotazioni eseguite. Ciò permette di decidere quale comportamento attuare nel caso di sparizione dell'ostacolo. Alla fine della fase 7 dell'algoritmo il veicolo si può trovare in due situazioni:

- in pista, orientato con un certo angolo che richiede il riposizionamento del veicolo (fase 4);

- parallelo alla pista ma lontano da essa, quindi passerà alla fase 9 dove procederà con moto rettilineo sino all'arrivo nel percorso desiderato.

Quest'ultima situazione causa un comportamento anomalo nel caso che il veicolo si trovi lontano da una curva del percorso, per cui percorre un lungo tragitto prima di rientrare in pista. Nella maggior parte dei casi il funzionamento è quello sperato in quanto la pista è di forma ellittica e quindi il veicolo trova la pista in un lasso di tempo che possiamo considerare accettabile, rispetto al tempo necessario per eseguire il sorpasso.

Fase III

In questa fase si considera l'ostacolo come un veicolo fermo per un malfunzionamento. Da ciò deriva che i veicoli fermi possono essere più di uno, e che tutti devono essere aggirati dal veicolo che sta sorpassando. Risulta quindi utile un estensione dell'algoritmo proposto nella **Fase I** (5.2.2) affinché sia in grado di sorpassare più ostacoli vicini lungo il percorso.

Nella **Fase I** non ci siamo preoccupati della scena intorno al robot, dando per scontato che ci possono essere solo un veicolo e un ostacolo. Ora il numero di soggetti della scena è vario e non noto a priori. Per cui l'algoritmo dovrà far si che il robot sia in grado di capire se è l'ostacolo è uno solo o più di uno.

Con questa parte si vuole attuare un comportamento adeguato nel veicolo in fase di sorpasso quando questo si trova davanti un ulteriore ostacolo.

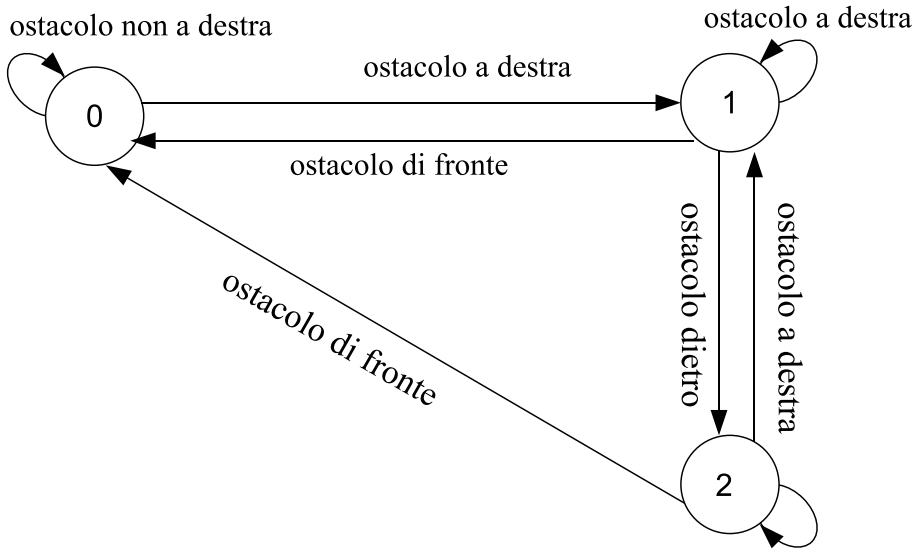

Figura 5.12: Modifica errata per evitare più ostacoli

Soluzione non corretta La prima soluzione è la più semplice e anche la più immediata. Se un veicolo che sta sorpassando trova davanti a sé un ostacolo allora dovrà girare nuovamente a sinistra per aggirare anche questo.

La modifica imposta all'algoritmo 10 è illustrata in figura 5.12. Nel caso il veicolo veda l'ostacolo di fronte a sé allora girerà nuovamente a sinistra sino a quando non si sarà lasciato l'ostacolo alla sua destra. In linea teorica questa soluzione funziona correttamente, sopratutto se si considera questo algoritmo come comportamento a se stante. Se non si considerano le altre funzionalità che devono essere implementate, l'algoritmo avrebbe l'inconveniente di girare indefinitamente intorno agli ostacoli senza continuare il suo percorso.

Come spiegato in 5.2.2 il loop non si verifica perché il veicolo cerca sempre il percorso da seguire. Va considerato che lo stato 0 (figura 5.12) corrisponde allo stato in cui il veicolo è ancora sulla pista e non ha ancora cominciato il sorpasso ma si sta solo posizionando in una posizione che gli permetta di svolgere la manovra.

Consideriamo la Fase II 5.2.2, dove si è progettato un algoritmo per far si che se l'ostacolo dovesse sparire[4] il veicolo sia in grado di attuare un comportamento che lo riporti in pista.

[4]Con sparizione dell'ostacolo si intende il fatto che il veicolo fermo per un malfunzionamento (ostacolo) riprenda a muoversi e non sia più dentro il raggio di vista del robot

L'algoritmo 11 prevede un comportamento diverso da attuare nei due casi di sparizione dell'ostacolo:

- punto 1 dell' algoritmo 11, il veicolo è ancora dentro la pista e in seguito alla sparizione dell'ostacolo il comportamento attuato è quello descritto al punto 5;
- punto 3 dell'algoritmo 11, in seguito alla sparizione dell'ostacolo il comportamento attuato è dettato dal punto 6, in quanto non è possibile sapere di quanto si è allonato dalla pista.

Lo stato 0 in figura 5.12 prevede che questa sia la prima operazione da attuare quando si esce dalla routine dell'inseguimento del percorso, e quindi che il veicolo sia ancora vicino alla pista. Mentre, se si incontra un ostacolo di fronte, nelle fasi del sorpasso etichettate col numero 1 o 2 il veicolo potrebbe essere lontano dalla pista. Per cui il comportamento da attuare nei due casi deve essere diverso, ciò implica una correzione all'algoritmo di figura 5.12.

Soluzione corretta L'algoritmo 11 implica che in ogni fase del sorpasso il robot sia in grado di capire di quanto si è allontanato dal percorso e che sia in grado di attuare un comportamento adeguato per il rientro in pista. Lo stato 0 della figura 5.12 verrà sdoppiato come in figura 5.13. Lo stato etichettato col numero 10 ha la stessa evoluzione discreta[5] dello stato 0 ma cambia la sua evoluzione continua e cioè il comportamento attuato durante la permanenza in questo stato. Le funzionalità incorporate in questo stato sono:

- la ricerca continua della pista, che nello stato 0 non viene eseguita perché si presume che il veicolo sia ancora nella pista;
- la velocità lineare non è nulla, come nello stato 0, ma è pari alla velocità utilizzata nelle altre fasi del sorpasso, per rendere il movimento più fluido.

L'aggiunta di queste due funzionalità è necessaria per la corretta implementazione della **Fase II**.

Fase IV

In questa fase si è studiato un protocollo di precedenza per il sorpasso. Non potendo comunicare tra loro i robot sono chiamati a capire in che posizione si trovano, per esempio se sono i primi o gli ultimi della fila, e in base a questa acquisiscono un ordine di precedenza in fase di sorpasso.

[5]Passaggio da uno stato ad un altro caratterizzato dal verificarsi di un evento.

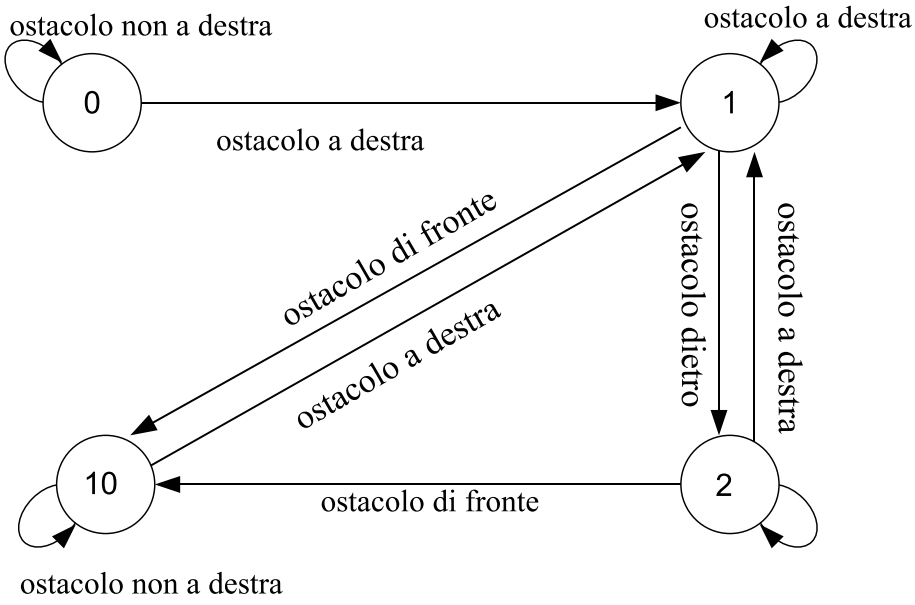

Figura 5.13: Modifica corretta per evitare più ostacoli

La necessità di implementare questa funzionalità è nata quando il plotone, in formazione compatta, si deve fermare di fronte ad un ostacolo. Può capitare che se il plotone sta compiendo il movimento circolare a regime, cioè quando tutti i veicoli mantengono la distanza fissata rispetto al veicolo che lo precede. In questo caso tutti i veicoli si fermeranno davanti ad un ostacolo nello stesso istante, ma solo il primo veicolo ha effettivamente davanti l'ostacolo, mentre gli altri hanno davanti un altro veicolo momentaneamente fermo che non dovrebbero considerare come un ostacolo.

Bisogna considerare che quando il primo ha aggirato l'ostacolo, sarà il secondo a svolgere la stessa operazione, poi al terzo e così via... Non è quindi possibile evitare a priori il sorpasso per i veicoli che non sono i primi della fila.

Prima di tutto si è implementato una funzione che analizzando la scena all'avvio, e a cadenza regolare, rende il veicolo in grado di capire la sua posizione relativa rispetto al plotone.

Algoritmo 12 ***Posizione relativa.*** *Fissato un periodo che possiamo chiamare di campionamento della posizione, si analizza la scena a cadenza regolare dettata da questo periodo.*

1. *se il veicolo non vede nulla intorno a sé, allora sa di essere solo;*

2. *se il veicolo vede un oggetto davanti e dietro, allora sa di essere al centro della formazione;*

3. *se il veicolo vede un oggetto dietro e nulla davanti, allora sa di essere il primo;*

4. *se il veicolo vede un oggetto davanti e nulla dietro, allora sa di essere l'ultimo;*

Il veicolo, consapevole della sua posizione relativa, quando vede un ostacolo aspetterà un tempo diverso a seconda a che sia il primo veicolo, quello centrale o l'ultimo della fila. Ovviamente il tempo minore è assegnato al primo della fila e quello più lungo all'ultimo.

L'avvio del sorpasso in base al tempo trascorso ha permesso anche la distinzione tra un ostacolo vero e proprio e un veicolo momentaneamente fermo. Si suppone che il tempo in cui il veicolo aspetti prima di sorpassare sia sufficiente per far si che l'ostacolo venga identificato come tale. In fase sperimentale si sceglierà opportunamente tale valore.

Un inconveniente che nasce da questo algoritmo è che il 2^o e 3^o veicolo sanno entrambi di essere centrali ma non sanno quanti veicoli li precedono e li seguono e quindi potrebbero iniziare il sorpasso contemporaneamente. Si evita questo inconveniente impostando dei tempi appropriati, decisi in fase sperimentale. Tali tempi devono consentire al veicolo capofila di aggirare l'ostacolo in modo che il secondo veicolo della fila non decida di sorpassare prima che il primo abbia concluso le prime fasi del sorpasso. Questo consente al secondo veicolo di avanzare appena il primo si è spostato, e quindi al terzo di seguire il secondo. Entrambi i veicoli, il 2^o e 3^o, si fermano davanti all'ostacolo e aspettano lo stesso lasso di tempo, a meno che non vi sia stato un aggiornamento della posizione nel mentre.

Per cui si è scelto di forzare l'aggiornamento della posizione qualora il veicolo si fermi e successivamente riprende il moto senza iniziare il sorpasso.

Nell'istante in cui il veicolo (ad esempio il 2^o) riprende il moto, il veicolo stesso non vede l'ostacolo perché deve prima percorrere lo spazio che c'è tra lui e l'ostacolo, il quale è maggiore del suo raggio di vista.[6] Quando aggiorna la sua posizione il 2^o veicolo capisce di essere il capofila e aggirerà l'ostacolo di conseguenza. La posizione del veicolo numero 3, invece, resta quella di veicolo centrale.

[6]Tale spazio è pari al diametro del robot più la distanza di sicurezza mantenuta dal primo veicolo e quella mantenuta dal secondo veicolo. Da ricordare che il diametro del robot è di 12 cm a cui va aggiunta la distanza di sicurezza di circa 4 cm, e il raggio di vista è di circa 10 cm.

Utilizzo algoritmo *Posizione relativa* per la formazione del plotone

L'*algoritmo 12* ha anche un altra funzionalità che può essere utilizzata nell'algoritmo di inseguimento presentato in sezione 4. Nella sezione 4 ci siamo preoccupati di proporre solo il modello per l'inseguimento tra i veicoli affinché mantengano la stessa distanza, senza preoccuparci della possibilità che i veicoli si possano trovare lontani[7] tra loro.

Sotto l'ipotesi che i veicoli possano essere lontani tra loro è necessario implementare un ulteriore funzionalità che permetta ad ogni veicolo di attuare un comportamento adeguato alla situazione per ricompattare il plotone.

La nuova funzionalità può essere descritta con l'algoritmo 13

Algoritmo 13 ***Formazione plotone.***

1. *Si segue l'algoritmo 12 per individuare la posizione relativa del veicolo nel plotone.*
2. *Se il veicolo è solo allora attuerà una velocità lineare random, si passa alla fase 1;*
3. *Se il veicolo è il primo della fila attuerà una velocità lineare costante, si passa alla fase 1;*
4. *Se il veicolo è al centro della fila attuerà una velocità lineare controllata dal controllore presentato nella sezione 4, si esce dall'algoritmo;*
5. *Se il veicolo è l'ultimo dalla fila attuerà una velocità lineare controllata dal controllore presentato nella sezione 4, si esce dall'algoritmo;*

L'algoritmo termina in un numero finito di passi per tutti i veicoli tranne che per il veicolo capofila. Non è possibile modificare l'algoritmo affinché anche per il primo veicolo termini in un numero finito di passi perché non avendo visione totale della scena è necessario che il primo veicolo controlli costantemente la sua posizione.

Questo algoritmo permette ai veicoli che si trovano in posizioni iniziali lontane tra loro di incontrarsi in un tempo finito e di posizionare il plotone in formazione simil treno.

5.3 Algoritmo complessivo

Il sistema complessivo è rappresentato dall'automa ibrido in figura 5.14 La figura 5.14 illustra l'integrazione, nell'intero programma, delle tre funzio-

[7] Fuori dal loro raggio di vista

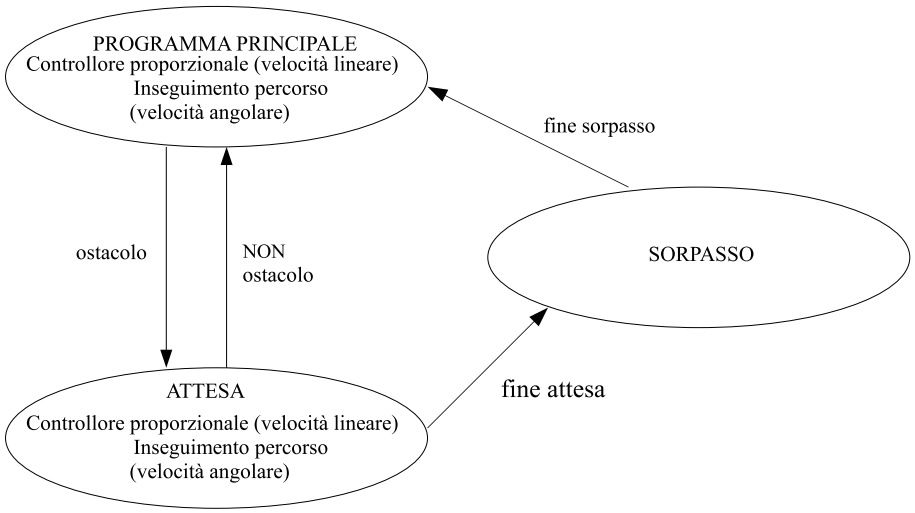

Figura 5.14: Automa ibrido dell'intero algoritmo

nalità:

- algoritmo principale: seguire il percorso desiderato e il programma per mantenere una distanza fissata dagli altri veicoli;
- stato di attesa: che rappresenta il tempo di decisione in cui il veicolo deve valutare se sorpassare o meno;
- sorpasso: fase di sorpasso dell'ostacolo, alla conclusione della quale si torna nell'algoritmo principale.

Nel programma principale sono implementati:

- l'algoritmo per l'inseguimento del percorso, controllo sulla velocità angolare;
- il controllo sulla velocità lineare per garantire il mantenimento della distanza di sicurezza.

Lo stato di attesa rappresenta il tempo di attesa prima dell'avvio del sorpasso vero e proprio, in cui è contemplata anche la possibilità che non vi sia un vero ostacolo da sorpassare e quindi il rientro al programma principale.

Lo stato di sorpasso implementa tutte le funzionalità descritte nella sezione 5.2 in un unico algoritmo. Quando il sorpasso è stato eseguito e il veicolo è tornato dentro il percorso viene eseguito nuovamente il programma principale.

Capitolo 6

Utilizzo dei sensori IR

In questo capitolo si descrive come è possibile utilizzare i sensori infrarossi per individuare un ostacolo e misurare la distanza tra il veicolo e l'eventuale ostacolo. Questo studio pone le basi per lo sviluppo di qualsiasi algoritmo di inseguimento tra i veicoli e aggiramento degli ostacoli. La parte sviluppata nella sezione 6.2 è propedeutica per l' implementazione degli algoritmi sul robot.

6.1 Generalità sui sensori a infrarossi

I robot *Khepera III* hanno 9 sensori infrarossi laterali (numerati dall'uno al nove, come in figura 6.1) che coprono l'inero perimetro del veicolo e due posizionati nel lato inferiore del robot (indicati col numero 10 e 11). Questi ultimi sono utili per realizzare l'inseguimento di una linea (*line following*). In figura 6.1 sono indicate le posizioni dei sensori infrarossi visti da sotto il robot. Ogni sensore è composto da un emettitore di luce infrarossa e un ricevitore.[1] Il dispositivo infrarossi include un emettitore a infrarossi e un foto-transistor. La casa produttrice suggerisce, in uno spazio limitato dal raggio di vista dei sensori, delle possibili applicazioni quali:

- codifica del raggio di vista come sensore di posizione
- rilevamento di materiale riflettivo come carta, nastro magnetico,...

Il robot esegue su questi sensori due diverse misure:

- *misura della luce ambientale.* Questa misura è ottenuta usando solo il ricevitore, senza emettere luce con l'emettitore. Ogni misura viene

[1] Il modello del sensore è **TCRT5000**, prodotti da *Vishay Telefunken.*

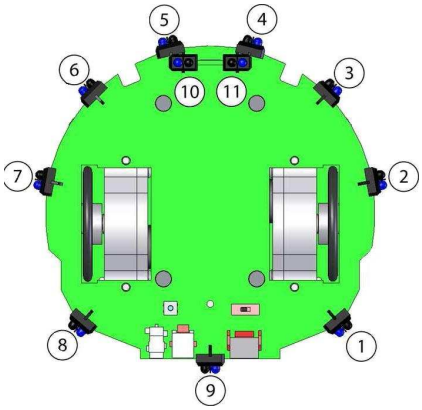

Figura 6.1: Vista sensori IR del Khepera III

eseguita ogni 33 ms. Durante questi 33 ms, gli undici sensori vengono letti uno di seguito all'altro ogni 3 ms. Viene restituito, ogni 3 ms, il valore dell'ultima misurazione fatta.

- *misura della luce riflessa dagli ostacoli (chiamata anche misura di prossimità).*Questa misura è ottenuta usando l'emettitore del dispositivo. Il valore restituito è la differenza tra la misura eseguita emettendo la luce e la misura effettuata senza l'emissione di luce. Ogni misura viene eseguita ogni 33 ms. Durante questi 33 ms, gli undici sensori vengono letti uno di seguito all'altro ogni 3 ms. Viene restituito, ogni 3 ms, il valore dell'ultima misurazione fatta.

L'output di ciascuna misura è un valore analogico che viene convertito in un valore digitale a 12 bit.

Le misure sono fortemente influenzate dalla luce ambientale. È sconsigliato usare i sensori infrarossi in prossimità di una luce con spettro che comprende anche la frequenza di emissione degli infrarossi. Le misure dipendono dalla riflettività dei singoli oggetti e sono influenzate dal rumore. Il manuale fornisce il grafico di andamento della corrente di collettore del transistor[2] in base alla distanza dell'oggetto visto dal sensore infrarossi. Consigliano, comunque, di considerarlo solo a titolo informativo e non come referenza.

Il mio lavoro è stato svolto più ad alto livello, studiando le funzioni di libreria fornite dalla *K-Team*. Per cui i valori dei sensori IR, relativi alla riflessione degli oggetti, sono dati dalla funzione:

```
int kh3_proximity_ir(char *outbuf, knet_dev_t *hDev)
```

I parametri della funzione sono:

[2]Si veda figura 3.9 del manuale [7].

- `outbuf` è il buffer dove saranno memorizzati i valori dei sensori alla chiamata della funzione;
- `hDev` è il socket che si occupa della gestione del dispositivo relativo ai sensori, chiamato *dsPic*;

e restituisce un intero che avrà valore pari all'indirizzo di un puntatore dove è memorizzato il valore della misura oppure valore NULL se l'operazione non è andata a buon fine.

Questa funzione, per ogni sensore, memorizza nel vettore `outbuf` un valore in byte con formato *little-endian*.

Definizione 14 ***Little-endian.*** *Il formato* ***little-endian*** *è un metodo usato dai calcolatori per memorizzare i dati, tipico dei processori della famiglia Intel. La memorizzazione, di dati in questo formato, inizia dal byte meno significativo per finire col più significativo.*

Definizione 15 ***Big-endian.*** *Il formato* ***big-endian*** *è un metodo usato dai calcolatori per memorizzare i dati, tipico dei processori della famiglia Motorola. La memorizzazione, di dati in questo formato, inizia dal byte più significativo per finire col meno significativo.*

in accordo con la sua definizione, il formato *little-endian* prevede che il sensore i avrà un valore associato su due byte `outbuf[2i-1]` e `outbuf[2i]`; di cui il primo rappresenta le cifre meno significative della misura e il secondo le cifre più significative. Per cui per avere la lettura del sensore è necessaria l'operazione

$$sensore[i] = outbuf[2i-1] | (outbuf[2i] << 8)$$

shift di 8 bit delle cifre più significative e il successivo *or bit a bit* con le cifre meno significative. Con questa operazione si converte il formato della misura da *little-endian* a *big-endian*, scrittura comune dei numeri.

Il valore misurato sui sensori è dipendente dalla luce riflessa dagli oggetti, più questi riflettono la luce maggiore sarà il valore misurato. Ovviamente, tale valore dipende anche dalla distanza, a parità di potere riflessivo, minore sarà la distanza maggiore sarà il valore misurato.

6.2 Misura di una distanza

In questa sezione si studia una relazione tra i valori dei sensori ottenuti dalla funzione di libreria `kh3_proximity_ir(...)` e le distanze misurate. La trattazione è stata divisa in tre parti:

- raccolta dati, misurazioni sui sensori a varie distanze;
- interpolazione dei valori misurati in una funzione;
- verifica della precisione e accuratezza della funzione proposta.

6.2.1 Raccolta dati

Sono stati fatti degli esperimenti per trovare una relazione tra il valore restituito dalla funzione `kh3_proximity_ir(...)` e la distanza esistente tra robot e oggetto. Preso il sensore i-esimo, sono state eseguite misure successive nelle stesse identiche condizioni, cioè robot e ostacolo non sono stati mossi. Si è visto che l'influenza della luce esterna ha un ruolo fondamentale nel rendere le cifre meno significative della variabile `outbuf` associata (`outbuf[2i-1]`) totalmente aleatorie, mentre le cifre più significative (`outbuf[2i]`) assumono quasi sempre lo stesso valore con una leggera variazione. Per questo motivo si è scelto di utilizzare le cifre più significative per studiare una funzione sperimentale di misurazione della distanza. Da questo punto in poi verrà sottointeso che si sta considerando la cifra più significativa della variabile `outbuf`.

Sono state eseguite 10 misurazioni del valore del sensore per ciascuna distanza. Il numero delle misurazioni è composto da:

- 5 misurazioni per il sensore frontale numero 4;
- 5 misurazioni per il sensore frontale numero 5.

In tabella 6.1 è elencata la statistica dei valori misurati per ciascuna distanza reale. Tra parentesi è indicato il numero di occorrenze di ciascun valore. Risulta evidente dalla tabella 6.1 che il raggio di vista del robot, utilizzando i soli sensori IR, è di non più di 10 cm. Nelle misurazioni eseguite vi sono due valori dei sensori che non compaiono mai, il valore 9 e il valore 10. Nella figura 6.2 sono stati inseriti i valori della tabella 6.1, dove l'ascissa è il valore memorizzato nella variabile `outbuf[2i]` e l'ordinata è la corrispondente distanza reale.

6.2.2 Funzione interpolatrice

La funzione f che si vuole realizzare deve avere come variabile indipendente (x) la cifra più significativa del valore misurato dai sensori, che qui chiamiamo anche IRm[3], e come variabile dipendente (y) la distanza misurata.

[3]La lettera m sta per la parola "misura".

distanza reale [cm]	*valore sensore*			
1				15(10)
2	11(3)	12(3)	13(3)	14(1)
3		6(3)	7(5)	8(2)
4		3(2)	4(7)	5(1)
5			2(4)	3(6)
6			1(1)	2(9)
7			1(8)	2(2)
8				1(10)
9			0(1)	1(9)
10			0(4)	1(6)
11 o più				0(10)

Tabella 6.1: Raccolta dati misure dei sensori

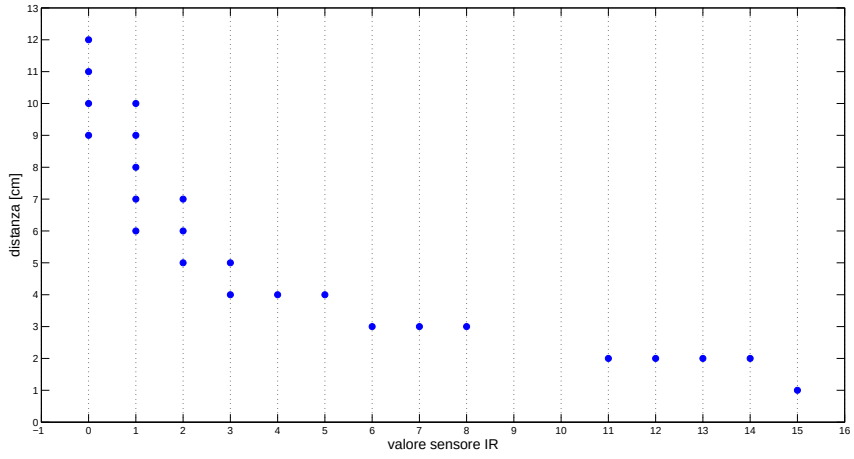

Figura 6.2: Misure dai sensori IR

IRm	*distanza reale*[*cm*]					*media pesata distanza*
0			11(10)	10(4)	9(1)	10.6
1	10(6)	9(9)	8(10)	7(8)	6(1)	8.32
2			7(2)	6(9)	5(4)	5.87
3				5(6)	4(2)	4.75
4					4(7)	4
5					4(1)	4
6					3(3)	3
7					3(5)	3
8					3(2)	3
9						
10						
11					2(3)	2
12					2(2)	2
13					2(3)	2
14					2(1)	2
15					1(10)	1

Tabella 6.2: Inversione relazione dei dati raccolti

$$y = f(x) \text{ con: } x \in \mathbb{N} \text{ e } y \in \mathbb{R};$$

entrambe le variabili, x e y, possono assumere solo valori all'interno di un certo intervallo:

$x \in [0:15]$ valori discreti (0,1,2,...);
$y \in [0:12]$ valori discreti (anche valori decimali).

In fase di raccolta dati la variabile indipendente è la distanza e quella dipendente è IRm, ora invece si vuole invertire la relazione. Nella tabella 6.2 sono indicate le distanze effettive (dalla tabella 6.1) in funzione di IRm, e nell'ultima colonna è indicata la media pesata della distanza in base al numero di occorrenze di IRm per quella data distanza. Esempio di calcolo della media della distanza per IRm pari a 1:

$$media\_distanza = \frac{(11 \cdot 10 + 10 \cdot 4 + 9 \cdot 1)}{(10 + 4 + 1)} = 10.6.$$

La figura 6.3 indica chiaramente la posizione del valore medio della distanza per ciascun valore misurato del sensore. I valori reali di distanza sono indicati con un pallino, mentre il valore medio è indicato con una stella.

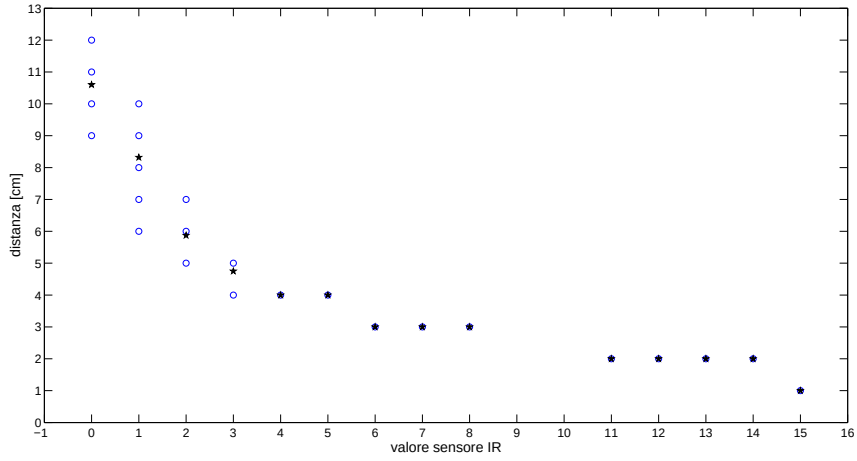

Figura 6.3: Media dei valori di distanza corrispondenti a ciascun valore del sensore

Si vuole trovare una curva interpolatrice dei punti corrispondenti ai valori medi, tenendo conto che per IRm pari a 5 abbiamo una sola misura, e per valori del sensore pari a 9 e 10 non vi sono misure corrispondenti. Non si può considerare reale la percentuale di occorrenze di $IRm = 5$, ne si può dire nulla sulla percentuale di occorrenze degli altri due valori. In più per $IRm = 0$ si impone che la distanza media sia pari a 11 cm, in quanto aumentando per tale distanza si è certi che IRm assuma valore nullo.

Si è eseguito l'interpolazione di valori medi. Per svolgere questi calcoli ci si è serviti del software MATLAB, e in particolare delle funzioni:

- `polyfit(x,d,n)` che calcola i coefficienti della funzione di grado `n` che meglio interpola i punti del vettore `y` in funzione del vettore `x`, tali coefficienti verranno memorizzati nel vettore `p`, con approssimazione tramite il metodo dei minimi quadrati;

- `polyval(p,x)` valuta la funzione espressa dal vettore `p` nei punti corrispondenti ai valori della variabile indipendente, indicati dal vettore `x`.

Nel nostro caso:

$$\begin{array}{ll} x \in \mathbb{N} & x = [0, 1, 2, \ldots, 8, 11, \ldots, 15]; \\ d \in \mathbb{R}. & d = [11,\ 8.32,\ 5.87,\ 4.75,\ 4,\ 4,\ 3,\ 3,\ 3,\ 2,\ 2,\ 2,\ 1]. \end{array}$$

Si è calcolato varie curve approssimatrici, graficate nella figura 6.4:

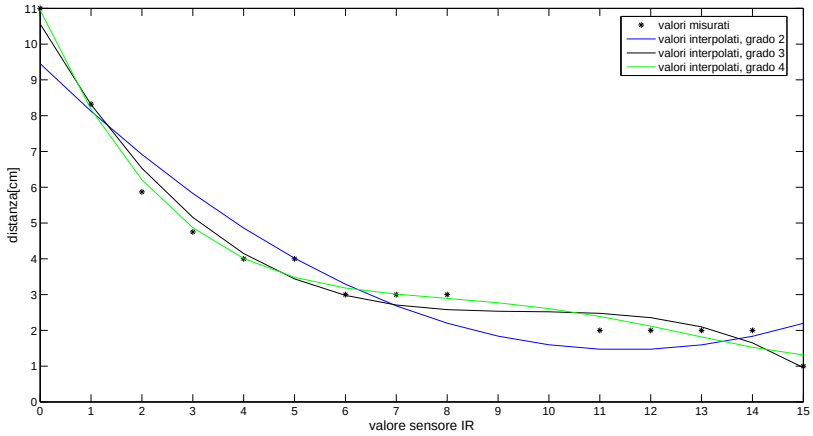

Figura 6.4: Curve interpolatrici

- curva di grado 2 (di colore blu), come si vede dalla figura 6.4, non permette di avere una giusta corrispondenza tra i valori IRm e la curva, sopratutto per valori di IRm superiori a 10;

- curva di grado 3 (di colore nero) ha una buona corrispondenza per valori bassi di IRm ma l'andamento è quasi costante per i valori di IRm compresi tra 8 e 11;

- curva di grado 4 (di colore verde) ha una corrispondenza accettabile tra i valori IRm e le distanze relative.

In prima approssimazione si era scelta una funzione di secondo grado, la curva in blu nella figura 6.4, la quale ha permesso di sviluppare gli altri aspetti del progetto seppure non fornisse la giusta accuratezza della misura. Una approssimazione di questo tipo non permette di stimare con adeguata accuratezza le misure su distanze vicino agli estremi della curva. Per esempio le distanze inferiori ai 2 cm non vengono discriminate, e lo stesso si può dire per distanze maggiori di 8 cm.

La curva di terzo grado è stata scartata a causa dell'andamento costante tra $IRm \in [8 \div 11]$ che misurano quasi tutti una distanza di circa 2.5 cm, valore accettabile per IRm=8 ma non per IRm=12.

Si è passati successivamente ad una approssimazione più accurata di tale funzione, arrivando ad una espressione di 4^o grado, rappresentata dalla curva verde del grafico 6.4. La scelta è caduta su questa curva per la buona corrispondenza tra le distanze reali e quelle calcolate dalla formula. Per esempio per IRm=12 la distanza calcolata è di 2.12 cm, e guardando la tabella 6.1 si

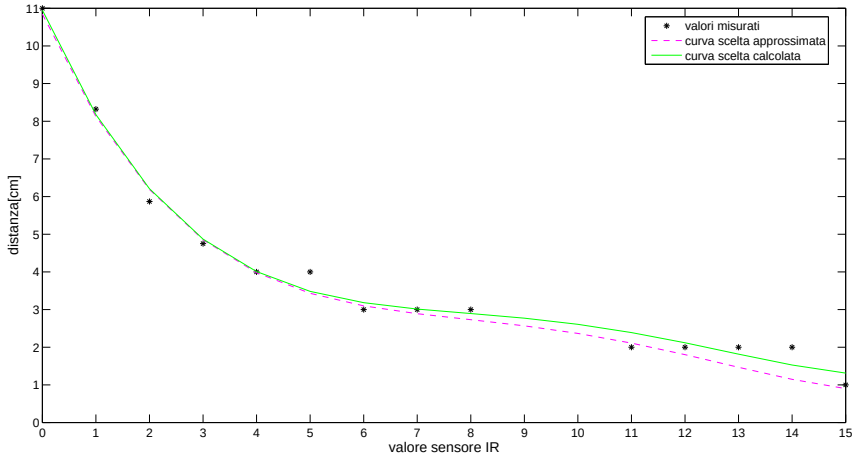

Figura 6.5: Funzione scelta per il calcolo della distanza

vede che il valore più probabile per una distanza di 2 cm è proprio IRm=12. Un altro esempio si ha per il valore di IRm=7 per il quale la distanza della curva corrisponde esattamente col valore di distanza associato ad IRm=7, considerando che il valore di IRm più probabile per una distanza di 3 cm è proprio IRm=7. Si è dunque scelto come curva interpolatrice una curva di grado 4^o.

Si è approssimato ulteriormente la curva , troncando le cifre alla 4 cifra decimale perché una precisione ulteriore sarebbe stata inutile data l'imprecisione della misura. L'espressione matematica di calcolo della distanza è data da

$$y = 0.0008 \cdot x^4 - 0.033 \cdot x^3 + 0.482 \cdot x^2 - 3.168 \cdot x + 10.846. \qquad (6.1)$$

dove la variabile indipendente x è il valore misurato dai sensori (cifra più significativa) con la funzione `kh3_proximity_ir` e la variabile dipendente y è la distanza misurata.

In figura 6.5, è stata graficata in rosa la curva corrispondente alla approssimazione scelta, rappresentata dall'equazione (6.1).

I valori di distanza che possiamo ottenere dalla funzione (6.1) non hanno un dominio continuo ma discreto in quanto i valori dei sensori IR sono numeri discreti che vanno da 0 a 15. Per cui i valori di distanza che potremo misurare sono dati dalla tabella 6.3. Qualora la misura che si vuole fare è sulla distanza vista dai due sensori frontali allora i possibili risultati ottenibili sono dati dalla tabella 6.4.

valore sensore IR	*distanza misurata [cm]*
0	10.845
1	8.128
2	6.187
3	4.854
4	3.979
5	3.431
6	3.099
7	2.890
8	2.731
9	2.568
10	2.366
11	2.110
12	1.803
13	1.468
14	1.147
15	0.900

Tabella 6.3: Valori ottenibili delle distanze misurate da un solo sensore

media sensori IR	*distanza misurata [cm]*
0.5	9.378
1.5	7.071
2.5	5.454
3.5	4.368
4.5	3.671
5.5	3.244
6.5	2.984
7.5	2.808
8.5	2.652
9.5	2.473
10.5	2.245
11.5	1.962
12.5	1.637
13.5	1.302
14.5	1.010

Tabella 6.4: Ulteriori valori ottenibili delle distanze misurate sulla media dei due sensori frontali

Si può notare come per per valori di IRm = 6, 7 e 8 in tabella 6.2 la distanza reale è sempre 3 cm mentre per la nostra funzione (6.1) assume dei valori leggermente diversi (si veda tabella 6.3), in modo da associare all'intensità della luce una maggiore vicinanza all'oggetto. Da notare anche la saturazione nel valore massimo e minimo di distanza misurata: ogni distanza superiore a 10 cm è riconosciuta come pari a 10.845 cm anche se non vi sono ostacoli; ogni valore di distanza inferiore a 1 cm sarà misurata in 0.9 cm anche se l'oggetto è più vicino. Questo impone una verifica che il valore misurato non sia in zona di saturazione.

6.2.3 Validazione

È stata eseguita la prova di verifica dell'approssimazione direttamente sul robot. Per fare ciò si è costruita la funzione in linguaggio C:

```
float calc_dist (float sensIR_val).
```

che implementa l'equazione (6.1) in linguaggio C. La variabile `sensIR_val` è `outbuf[2i]` relativamente al sensore i-esimo, `sensIR_val` è di tipo `float` perché è necessario poter passare in ingresso anche un valore decimale ottenuto ad esempio dalla media delle misure effettuate dai due sensori frontali. Nel caso invece si voglia la distanza calcolata su un solo sensore allora il programma eseguirà il cast in automatico da `int` a `float`.

Si è scritto un programma C, chiamato `test_distanza`, che permette al robot di leggere il valore dei sensori e calcolare la distanza con la funzione `calc_dist(...`.

Posto un veicolo a distanza nota da un altro veicolo, si è calcolata tale distanza. In tabella 6.5 sono riportati:

- nella prima colonna, i valori di distanza reali;
- nelle altre colonne, i valori di distanza calcolati dal programma

  `test_distanza`

  in base al valore restituitoci dal sensore i-esimo (con i=1,2,3.), approssimati alla prima cifra decimale;
- nell'ultima colonna il valore di distanza ottenuto dal programma

  `test_distanza`

  sulla media dei valori misurati da sensori 4 e 5, la distanza è stata approssimata alla seconda cifra decimale.

distanza reale[cm]	*1*	*2*	*3*	$(4+5)/2$
1	0.9	1.2	0.9	1.01
2	1.2	1.5	1.5	2.56
3	2.7	2.9	3.1	3.25
4	3.4	4.0	4.9	4.85
5	4.9	4.9	6.2	6.2
6	6.2	6.2	8.1	7.07
7	8.1	6.2	8.1	8.13
8	8.1	8.1	8.1	9.37
9	10.9	10.9	10.9	9.37
10	10.9	10.9	10.9	10.84

Tabella 6.5: Verifica della distanza misurata

Avendo a disposizione un intervallo più ampio di valori ottenibili compresi tra $1cm$ e $4cm$ la precisione maggiore nella misura si ha per valori di distanza compresi in quell'intervallo. Per distanze maggiori non si ha un ventaglio di valori altrettanto ampio. È possibile considerare due tipi di misura:

- eseguita su un solo sensore;
- eseguita sulla media dei due sensori frontali;

Quest'ultima misurazione permette di avere un insieme di valori di distanza ancora più ampio, oltre quelli ottenuti dalla tabella 6.3 si aggiungono quelli della tabella 6.4. Dalla tabella 6.5 si vede che la accuratezza nella misura ottenuta dalla media dei valori dei due sensori frontali (si veda l'ultima colonna) non è molto diversa da quella relativa alla misura su un solo sensore. Quindi è possibile considerare come equivalenti i due tipi di misure.

È stata verificata la precisione delle misure effettuate dal robot tramite la media dei valori restituiti dai due sensori frontali. In tabella 6.6 sono riportati i valori misurati per ogni distanza e la loro percentuale di occorrenza. I valori misurati sono tutti sufficientemente vicini tra loro da poter considerare la misura con la giusta precisione per il suo scopo.

Tabella 6.6: Verifica precisione misura

distanza reale[cm]	*valore misurato [cm]*	*percentuale [%]*
1	0.9	80
	1.01	20
2	2.56	20
	2.11	20
	1.96	30
	1.64	20
	1.47	10
3	3.25	20
	2.98	40
	2.89	40
4	4.85	20
	4.36	40
	3.97	40
5	6.19	20
	5.45	60
	4.85	20
6	8.12	25
	7.07	25
	6.19	50
7	8.13	72
	7.07	28
8	9.37	40
	8.12	60
9	8.12	30
	9.37	35
	10.85	35
10	8.12	33
	9.37	50
	10.85	17

Capitolo 7

Implementazione su KheperaIII

In questo capitolo è trattata l'implementazione sui robot KheperaIII degli algoritmi presentati nella sezione 5. Questo capitolo è strutturato in due parti:

- funzionalità generali implementate, dove sono spiegate le funzioni base necessarie per l'implementazione degli algoritmi veri e propri;
- implementazione degli algoritmi, dove è spiegata la tecnica utilizzata per implementare gli algoritmi in un robot Khepera III.

7.1 Funzionalità generali

Le funzioni di libreria forniteci dalla K-Team non sono sufficienti per l'implementazione degli algoritmi, per cui partendo da esse si sono realizzate delle nuove funzioni più adatte ai nostri scopi. In questa sezione sono presentate quelle base e di conseguenza utilizzate da tutti, o quasi, gli algoritmi. Nella sezione successiva sono presentate le funzioni create per gli scopi richiesti dallo specifico algoritmo.

7.1.1 Soglie dei sensori IR

Per poter implementare gli algoritmi precedentemente descritti è importante avere una visione chiara dal punto di vista del robot. La prima funzionalità implementata è

```
sensoriIR(void).
```

Questa funzione utilizza la funzione di libreria `kh3_proximity_ir(...)`, presentata nella sezione 6.2; per semplicità di lettura riproponiamo qui l'interfaccia della funzione.

```
int kh3_proximity_ir(char *outbuf, knet_dev_t *hDev)
```

I valori memorizzati nella variabile `outbuf` sono i valori misurati dai vari sensori. È stato necessario quindi costruire un programma che permettesse di visualizzare questi valori durante il movimento del robot. Per questa ragione è stata creata la funzione `senroriIR()`, la quale, dopo aver convertito i valori di `outbuf` dal formato *little-endian*[1] al formato *big-endian*, restituisce tali valori. L'output è ottenuto con una stampa a schermo nella shell del KheperaIII.[2]

Questa funzione è la base e d'aiuto per ricavare diversi dati sperimentali, come ad esempio:

- `sensoriIR()` è d'aiuto in fase di progetto per individuare delle soglie per i passaggio di stato nelle operazioni più elementari, oppure per una lettura dei valori misurati senza nessuna rielaborazione;
- `calc_dist(...)` è basata su dati sperimentali, misure dei sensori IR ottenuti con la precedente funzione, e calcola la distanza degli oggetti nel raggio di vista del robot.

7.1.2 Calcolo di una distanza

Per il calcolo della distanza si sono sfruttati i dati sperimentali ottenuti dalla funzione `senroriIR()` e si è implementata la funzione

```
float calc_dist (float sensIR_val),
```

descritta nella sezione 6.2. Per cui in questo capitolo verrà solo elencata per completezza.

7.1.3 Attuare una velocità

Per far muovere il veicolo dobbiamo, in un qualche modo comunicargli con che velocità deve eseguire il suo moto. Nella sezione 3.2 è spiegato il modello cinematico del robot Khepera III e come le velocità delle singole ruote sono in relazione con la velocità angolare e lineare dell'intero veicolo.

È necessario studiare prima le funzioni di libreria del robot, per capire in che modo implementare le leggi del modello cinematico già descritto. Le funzioni di libreria forniteci dalla *K-Team* permettono di settare le regolazioni dei controller dei motori delle ruote, il quale impone delle velocità singolarmente ad ogni ruota. Il manuale del robot [7] ci fornisce i dati tecnici dei controller.

[1] Si veda sezione GeneralitaIR per la definizione del formato.

[2] La shell di questa distribuzione di linux, come per molte distribuzioni embedded, è *sh*.

La prima funzione di libreria utile è:

```
int initMot(knet_dev_t *hDev)
```

la quale inizializza e configura il controller del motore. La variabile di tipo `int` restituita ha valore 1 se l'operazione è andata a buon fine e valore 0 in caso contrario. La variabile in ingresso `hDev` è il socket[3] che si occupa della gestione del singolo motore, che possiamo chiamare `mot1` per il primo motore e `mot2` per il secondo.

Questa funzione è richiamata all'interno della funzione di inizializzazione di tutti i dispositivi del veicolo:

```
int initKH3( void )
```

la quale non ha parametri in ingresso e restituisce un intero di valore 1 se le operazioni sono andate a buon fine e 0 in caso contrario. Questa funzione inizializza il motore e il bus dati del veicolo, il quale stabilisce le comunicazioni tra il processore e le periferiche di input (sensori IR e ultrasuoni).

La K-Team fornisce anche un programma di test (`khepera3_test`) da provare sul robot con lo scopo di spiegare il funzionamento dei principali comandi. Da questo programma si è utilizzato la funzione

```
int motSpeed( int argc, char *argv[], void *data)
```

nel modo seguente

```
int motSpeed( int m_sx, int m_dx).
```

La funzione `motSpeed` prende in ingresso due variabili di tipo `int`, `m_sx` e `m_dx`, che impostano il numero di giri del motore sinistro e destro rispettivamente. Viene impostato il modo di funzionamento del controllore del motore, la scelta è tra *modalità velocità* e *modalità posizione*. Nel nostro caso ci interessa il funzionamento in modalità velocità, il quale permette di agire sulla velocità del motore.

La velocità del motore è misurata in giri al secondo $[giri/s]$, e va trasformata nella velocità della ruota. Questa trasformazione dipende dai rapporti dell'encoder del motore e delle ruote e dal numero di misure eseguite per rivoluzione delle ruote. Tale rapporto per la configurazione di default da noi usato ci è fornito dalla K-Team e la conversione è la seguente:

$$RealSpeed = \frac{MotorSpeed}{Kspeed} \left[\frac{mm}{s}\right]. \tag{7.1}$$

- RealSpeed è la velocità della ruota espressa in $[\frac{mm}{s}]$;

[3]Definito come la variabile `knet_dev_t`.

- MotorSpeed è la velocità del motore espressa in $[\frac{giri}{s}]$;
- Kspeed è una costante pari a 144.01 nella configurazione di default; dipende dalla circonferenza delle ruote (di $128.8mm$), dal numero di misurazioni che l'encoder compie per ogni rivoluzione della ruota (2764 misurazioni) e dalla risoluzione dell'encoder (compie quattro volte una misurazione per ogni pulsazione, x4).

Nel modello cinematico proposto nella sezione 3.2 si imposta la velocità delle singole ruote impostando la velocità lineare v e la velocità angolare w. Riporto qui le equazioni di conversione:

$$\begin{cases} v = \frac{v_d+v_s}{2} \\ \\ w = \frac{(v_d-v_s)}{L} \end{cases} \tag{7.2}$$

dove:

- v è la velocità lineare del robot in [mm/s];
- w è la velocità angolare del robot in [rad/s];
- v_d è la velocità della ruota destra in [mm/s];
- v_s è la velocità della ruota sinistra in [mm/s];
- L è la lunghezza dell'asse delle ruote.

Indichiamo con V_{md} la velocità del motore destro e con V_{ms} la velocità del motore sinistro. La conversione in velocità lineare delle singole ruote è data dalla 7.1, che qui riscriviamo:

$$\begin{cases} v_d = \frac{V_{md}}{Kspeed} \\ \\ v_s = \frac{V_{ms}}{Kspeed}. \end{cases} \tag{7.3}$$

Riscriviamo ora la (7.2) nel nostro caso specifico:

$$\begin{cases} v = \frac{1}{2}\frac{V_{md}+V_{ms}}{Kspeed} \\ \\ w = \frac{1}{L}\frac{V_{md}-V_{ms}}{Kspeed} \end{cases} \tag{7.4}$$

dove L è la lunghezza dell'asse tra le ruote. Sfruttando la (7.4) possiamo assegnare le velocità ai motori delle singole ruote in [giri/s] come indicato nell'equazione (7.5).

$$\begin{cases} V_{dx} = \frac{(Kspeed)}{2}\,(2\,v + w\,L) \\ V_{sx} = \frac{(Kspeed)}{2}\,(2\,v - w\,L) \end{cases} \tag{7.5}$$

È stata quindi creata una funzione in C, che implementi la (7.5), con la seguente interfaccia:

```
int set_velocita(float v, float w)
```

dove v è la velocità lineare in mm/s e w è la velocità angolare in rad/s, e restituisce una intero pari a 1 se l'operazione è andata a buon fine, pari a 0 in caso contrario. All'interno della funzione `set_velocita(...)` viene richiamata la funzione `motSpeed(...)` che assegna la velocità ai motori delle singole ruote.

7.2 Il programma

7.2.1 Sistema ibrido

Per poter implementare un qualsiasi comportamento sul robot è necessario prima di tutto realizzare un sistema ibrido, che modelli il comportamento robot.

Il sistema ibrido[4] da implementare si basa su:

- dinamiche ad avanzamento temporale;
- dinamiche ad avanzamento discreto.

L'avanzamento temporale è dettato dal periodo di campionamento (T), il sistema tempo reale progettato aggiorna lo stato ad una cadenza temporale fissata dal tempo di campionamento. Ogni T secondi il programma legge il valore dei sensori, e se necessario cambia lo stato discreto del sistema.

[4] In appendice A sono riportati alcuni richiami teorici.

Sistema ibrido in C

Per realizzare il sistema ibrido in un linguaggio comprensibile dal robot è necessario tradurre gli algoritmi in linguaggio C.

Il sistema implementato è un sistema in tempo reale con un dato periodo di campionamento, cadenza temporale con cui viene aggiornata la scena e tempo necessario per svolgere le operazioni richieste dal programma.

Per impostare il periodo di campionamento si struttura il programma principale come illustrato in figura 7.1.

La funzione `kb_getTime()` è una funzione di libreria forniteci dalla K-Team che non richiede parametri in ingresso e restituisce il tempo trascorso dall'avvio del programma, misurato in *ms*. Tutte le variabili di tempo sono definite di tipo `kb_time_t` che ridefinisce un tipo `unsigned int` di 32 bit (la risoluzione della misura è del millisecondo).

Si vede nel pseudo codice descritto in figura 7.1 come è stata realizzata la cadenza temporale in linguaggio C. Il periodo di campionamento è ottenuto grazie alle operazioni:

- memorizzazione istante di tempo di inizio operazione,

  ```
  start = kb_getTime();
  ```

- memorizzazione istante di tempo di fine operazione,

  ```
  end = kb_getTime();
  ```

- calcolo dell'intervallo di tempo impiegato per svolgere le operazioni

  ```
  dif = end - start;
  ```

- prima di rieseguire tutte le operazioni si aspetta un lasso di tempo pari alla differenza tra il periodo di campionamento e l'intervallo di tempo impiegato per svolger le operazioni

  (`tempo = PERIODO - dif; usleep (tempo*1000)`).

Questo metodo permette di calcolare il tempo necessario per svolgere le operazioni e, quindi, permette di decidere un tempo di campionamento uguale per ogni operazione. Il valore minimo del tempo di campionamento è dato dai limiti fisici dei sensori, il limite massimo è dato dalla prontezza che vogliamo nel sistema.

Il sistema è tempo reale, ma c'è da considerare che nel robot oltre al nostro programma sono presenti altre routine del sistema operativo. Può capitare che qualche processo del sistema operativo richieda risorse che servono

```
#define PERIODO XX //tempo di campionamento
int main(int argc, char *argv[]){
  /*dichiarazione variabili*/
  ...
  /*inizializzazione variabili*/
  ...
  /*inizializzazione dei dispositivi del robot*/
  ...
  while(1) { /*programma principale*/
  /*  lettura tempo iniziale  */
  start = kb_getTime();
  /*  lettura sensori IR  */
  kh3_proximity_ir((char *)IRBuffer, dsPic);
  /* varie funzionalità implementate */
  ...
  /* macchina a stati   */
  ...
  set_velocita(vel_lin, vel_ang);
  /* lettura tempo finale*/
  end = kb_getTime();
  /*tempo impiegato per le operazioni*/
  dif = end - start;
  /* attesa */
  tempo = PERIODO - dif; //tempo in ms
  usleep (tempo*1000);// trasformazione tempo in us
   }
  return 1;
}
```

Figura 7.1: Pseudo codice del programma principale.

all'esecuzione del nostro programma e quindi modifichino il tempo massimo di esecuzione delle operazioni.

La distribuzione di Linux installata sul KoreBotLe del robot Khepera III è Angstrom Una soluzione è chiudere tutti i processi attivi[5] in Angrstrom, ma non è fattibile al 100% perchè vi sono processi che non possono essere eliminati. Per esempio, abbiamo necessità di interfacciarci col robot e quindi di instaurare un collegamento con esso. Per cui è necessario che almeno la connessione WiFi sia attiva.

Per cui una soluzione brutale per ovviare all'inconveniente della richiesta di risorse del sistema operativo durante l'esecuzione del programma è la modifica del tempo di campionamento. È impossibile sapere quanto tempo richiederà l'esecuzione di quelle operazioni, in questo caso per cui si è attuata la strategia di lasciare il tempo di campionamento costante e se in alcuni casi le operazioni richiedono più tempo di quello fornitogli allora si aspetta. Tradotto in linguaggio C diventa:

```
if(PERIODO >= dif) {// funzionamento nominale
   tempo = PERIODO - dif; //tempo in ms
   usleep (tempo*1000);// aspetta
   }
else { // eccezione
   ..non aspettare, ricomincia il while
          }
```

Macchina a stati

Il cuore dell'automa ibrido sta nella macchina a stati che implementa la dinamica discreta e continua del sistema. In linguaggio C questo è stato tradotto in un costrutto `switch-case` in cui il passaggio da uno stato al successivo è determinato dagli eventi discreti mentre l'evoluzione temporale è imposta dallo stato in cui il programma si trova. Per cui il programma principale[6] illustrato in figura 5.14 può essere scritto in linguaggio C come illustrato in figura 7.2.

Nel codice della figura 7.2 c'è l'implementazione del programma principale. Da notare:

- l'integrazione delle due dinamiche temporali, inseguimento del percorso desiderato (realizzato con la funzione `line-following()`) e inseguimento tra i veicoli (il controllo a ciclo chiuso sulla distanza), avviene

[5]Per vedere i processi attivi si digita il comando `ps -a` e per chiudere un processo si deve digitare `kill numProc`.

[6]Descritto nella sezione 5.3

```
switch(stato){
    case(0):
              vel_lin = - K*g;
              vel_ang = line_following(sx_dx);
              if (ostacolo) {
                   nextstate = 1;  break;
                }
              else {
                   nextstate = 0;  break;
                }
    case(1):
               vel_lin = - k*g;
               vel_ang = line_following(sx_dx) ;
               if (count == TEMPO_ATTESA_SORPASSO)  {
                        count = 0;
                        nextstate = 2;  break;
                     }
               else   {  if (ostacolo) {
                                 count++;
                                 nextstate=1;  break;
                              }
                         else {
                                 count = 0;
                                 nextstate = 0;  break;
                               } }
    case(2):
               if (sorpasso(...)) {
                        nextstate = 2; break;
                     }
               else  {
                        nextstate = 0; break;
                     }
}
```

Figura 7.2: Codice C per la realizzazione della macchina a stati del programma complessivo.

nello stesso stato in quanto non sono in conflitto tra loro. Ciascuna dinamica riguarda una variabile di stato diversa, una riguarda la velocità angolare, l'altra la velocità lineare;

- l'attesa per discriminare tra un ostacolo e un veicolo fermo è stata implementata come lo stato principale ma con in più l'abilitazione di un contatore per il tempo trascorso da quando il veicolo è stato costretto a fermarsi;

- lo stato di sorpasso è una sub-routine indipendente che richiede l'uscita dal programma principale.

Per le spiegazioni sulle varie funzioni dei programmi secondari si rimanda alle sezioni successive.

7.2.2 Movimento circolare

Il primo utilizzo della funzione `sensoriIR()` è stato eseguito proprio per implementare il movimento circolare del veicolo. Il veicolo segue il percorso desiderato perché rileva il contrasto tra la striscia nera della pista e lo sfondo bianco. Tale contrasto cambia a seconda del materiale per cui è necessaria una calibrazione delle soglie.

Lo sfondo bianco nella misura eseguita[7] coi sensori 10 e 11 restituisce sempre il valore di fondo scala, mentre il colore nero restituisce valori di misura che vanno dal 6 al 7. Per cui il valore di soglia nella distinzione dei due colori sulla pista è stato imposto pari a 10. La funzione

```
int sensoriIR_pista(int IRsx, int IRdx)
```

legge i valori restituiti dai sensori 10 (IRsx) e 11 (IRdx) e basandosi sulla soglia scelta costruisce una variabile di tipo `int`, che chiamiamo `sx_dx`, che restituisce:

- 11 nel caso in cui entrambi i sensori sono sopra la pista;

- 10 nel caso in cui il sensore sinistro è sulla pista e il destro no;

- 01 nel caso in cui il sensore sinistro è fuori dalla pista e il destro sulla pista;

- 00 nel caso in cui entrambi i sensori sono fuori dalla pista.

Questi valori sono poi utilizzati dalla funzione:

[7] Misura eseguita con la funzione `kh3_proximity_ir()` come descritto nel capitolo 6.

```
float line_following(int sx_dx)
```

la quale restituisce il valore della velocità angolare per l'inseguimento della linea.

L'implementazione del path following sul robot si basa sull'algoritmo descritto nella sezione 5.1. La funzione `line_following(...)` implementa questo comportamento in linguaggio C.

```
float line_following(int sx_dx)
{   float vel_ang;
    switch(sx_dx)   {
                        case(0):
                                vel_ang = 0.0;   break;
                        case(1):
                                vel_ang = -0.3;  break;
                        case(10):
                                vel_ang = 0.3;   break;
                        case(11):
                                 vel_ang = 0.0;   break;
                     }
    return vel_ang;
    }
```

Questa funzione viene usata direttamente per assegnare il valore di velocità angolare durante l'esecuzione del programma principale

```
vel_ang=line_following(sx_dx).
```

7.2.3 Controllo sulla distanza

Il controllo sulla distanza viene eseguito con un controllore proporzionale. La realizzazione di questo tipo di regolazione in linguaggio C è la seguente:

```
#define rif 4 //[cm]
#define K 10 //guadagno
...
int main(){
...
g = rif - calc_dist ((float) ((IR[4]+IR[5])/2));
...
vel_lin = - K * g;
...
}
```

Il controllo sulla velocità lineare avviene all'interno dello stato dedicato al programma principale. Nella sub-routine che implementa il sorpasso la velocità lineare rispetta la legge di controllo dedicata.

7.2.4 Evitare gli ostacoli

Il primo algoritmo realizzato è l'implementazione di un comportamento per evitare l'ostacolo senza aggirarlo. Questo approccio è stato implementato ma non essendo utile ai fini dell'algoritmo finale non è stato integrato nel programma complessivo. In questa sezione verrà illustrato brevemente questo algoritmo.

Il secondo approccio intrapreso è quello che poi si è utilizzato nel programma finale, in questo caso si è realizzato l'aggiramento dell'ostacolo. Questo verrà illustrato nella sezione successiva.

Primo approccio

L'implementazione dell'algoritmo 8 descritto nella sezione 5.2.1 richiede che tutti i controlli siano eseguiti sui valori letti dai sensori frontali (precisamente dai sensori IR numero 3, 4, 5 e 6), tralasciando gli altri sensori, in quanto si presuppone un moto rettilineo in una sola direzione di marcia (avanti). Questa funzione è implementata in:

```
int obstacle_avoidance (float *v, float *w,
                                char *IRBuffer)
```

la quale richiama la funzione:

```
int lato_sorp(char * IRBuffer)
```

che decide il lato migliore per proseguire il proprio cammino.

La funzione `obstacle_avoidance(...)` prende in ingresso i valori dei sensori infrarossi (`IRBuffer`) e decide le velocità lineari e angolari da attuare in base alla funzione `lato_sorp(...)`. Il valore restituito dalla funzione `obstacle_avoidance(...)` è un intero che avrà valore pari ad 1 se c'è un ostacolo da sorpassare e valore pari a 0 in caso contrario.

La funzione `lato_sorp(...)` legge i valori dei sensori IR e stabilisce da che lato è meglio aggirare l'ostacolo. Il lato è stabilito dalla variabile di tipo `int` restituita, la quale avrà valore:

- 1 se deve aggirare l'ostacolo da destra;

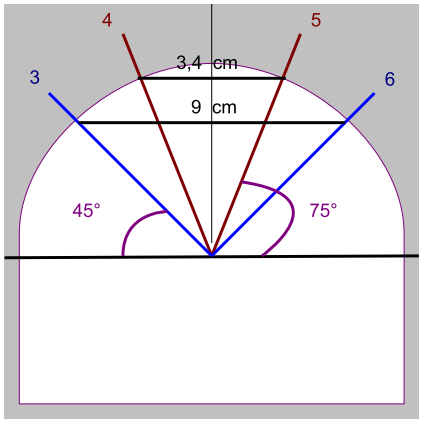

Figura 7.3: Angolo di vista da ciascun sensore

- 0 se deve aggirare l'ostacolo da sinistra.

L'algorimto 9 implementa un comportamento per far si che se il veicolo si trovi tra due ostacoli, uno alla sua destra e l'altro alla sua sinistra, allora sia in grado di valutare se ci sia spazio sufficiente per passare in mezzo agli ostacoli o sia necessario cercare una via migliore. Questo algoritmo, come il precedente, si basa sulla sola scena che si presenta davanti al robot e non su quello che il robot si lascia alle spalle. La dimensione dell'ostacolo è nota a priori, mentre non ne è nota la posizione. Se i due ostacoli, quello a destra e quello a sinistra della direzione di marcia del veicolo, sono posti ad un distanza tra loro che è superiore alle dimensioni del robot più una zona di sicurezza, allora il veicolo potrà passare in mezzo agli ostacoli. Il calcolo della distanza tra i due ostacoli si basa sugli angoli di vista dei quattro sensori sopracitati, illustrati in figura 7.3. La retta nera orizzontale indica la connessione tra i sensori 2 e 7 che passa per il centro del robot. La distanza tra i sensori 3 e 6 è pari a $9cm$ mentre la distanza tra i sensori 4 e 5 è di $3.4cm$. Se si considera orientata a 0^o la retta perpendicolare all'asse tra i sensori 2 e 7, allora:

- i sensori 3 e 6 sono posizionati ad un angolo pari a 45^o;
- i sensori 4 e 5 sono posizionati ad un angolo pari a 15^o;

rispetto a tale retta. In figura 7.4 si vede come è possibile valutare la distanza tra gli ostacoli che il veicolo si trova davanti. In questo caso i calcoli sono fatti per i sensori 3 e 6 ma le formule sono simili per gli altri due sensori. Geometricamente abbiamo che tale distanza è:

$$d = \cos(45^o)(a + b) + c$$

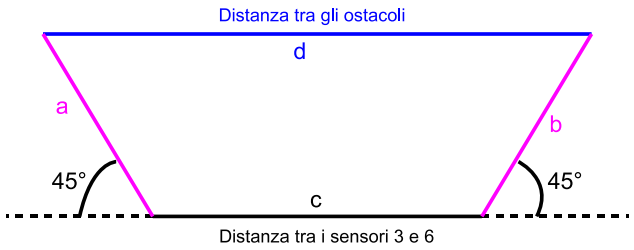

Figura 7.4: Geometria della vista degli ostacoli da parte del robot

e, per i sensori 4 e 5 cambia l'angolo (75^o)e la lunghezza di c (3.4 cm). Si nota che la distanza massima tra gli ostacoli varia a seconda dei sensori con cui viene guardata.

- Guardando l'ostacolo coi sensori 3 e 6 la distanza d assume valori che vanno da 10.42 cm per i valori minimi dei sensori IR, sino ad un massimo di 24.11 cm per i valori massimi misurati;

- guardando l'ostacolo coi sensori 4 e 5 la distanza d assume valori che vanno da 3.95 cm per i valori minimi dei sensori IR, sino ad un massimo di 9.03 cm per i valori massimi misurati.

È stato eseguito un controllo prima sui sensori 4 e 5 per vedere se l'ostacolo è proprio davanti al robot, e solo successivamente sui sensori 3 e 6 per valutare se lo spazio è sufficiente per passare. Come distanza minima tra gli ostacoli che consenta il passaggio del veicolo è stato scelta una d pari a 16 cm.

7.2.5 Il sorpasso

L'algoritmo del sorpasso è strutturato come un programma a sé stante, con una sua macchina a stati. In questa sezione non è riportato il file C, presente comunque in appendice D. La funzione C che implementa l'algoritmo è

```
int sorpasso_pista (float * v, float * w,
                    char * IRBuffer, int sx_dx )
```

La variabile `int` restituita dalla funzione indica se il veicolo deve ancora sorpassare o la fasse di sorpasso è terminata.

La macchina a stati è strutturata come descritto nell'algoritmo che la implementa, in fase di sviluppo sono state necessarie ulteriori modifiche descritte nelle **Fasi I-IV** della sezione 5.2. In questa sezione verrà illustrato l'algoritmo finale, si veda figura 7.5. Non sono stati disegnati i cappi negli

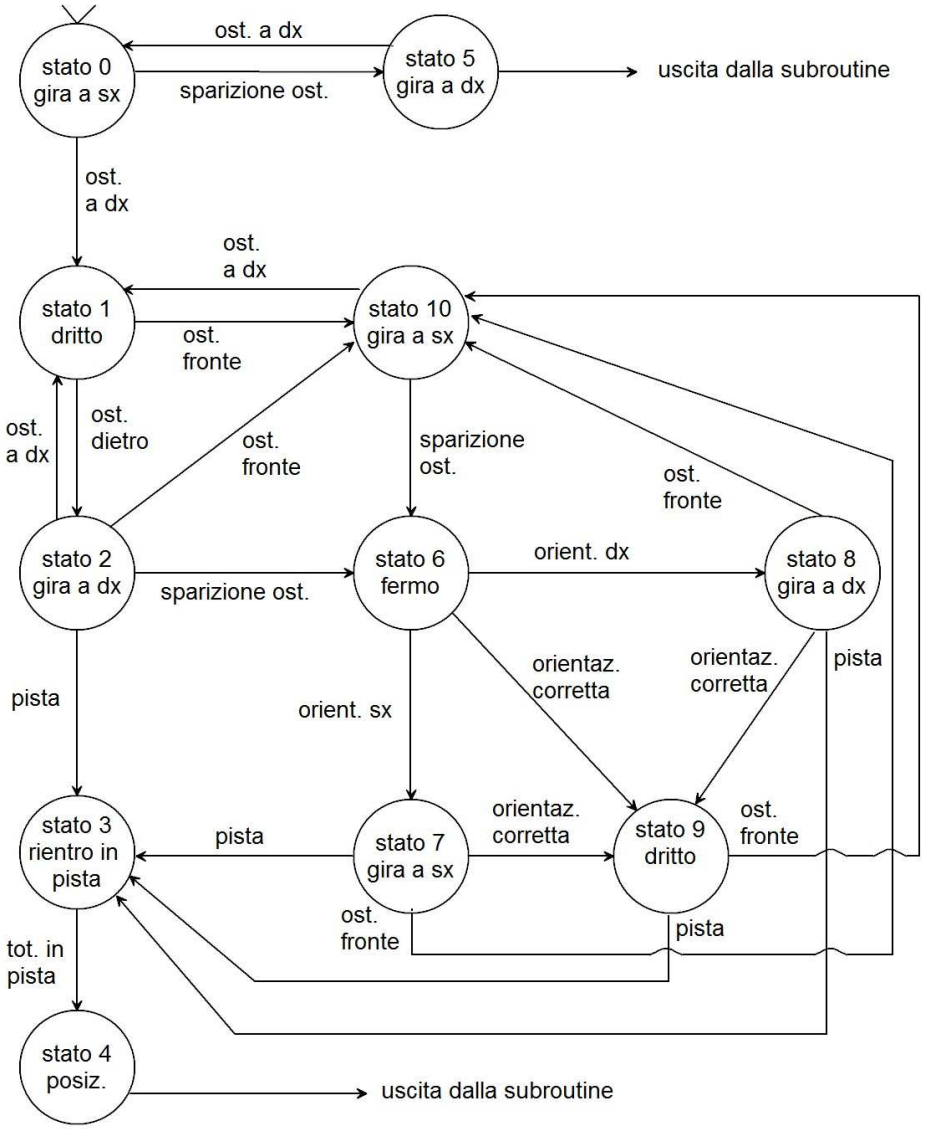

Figura 7.5: Macchina a stati dell'algoritmo del sorpasso. (os = ostacolo, sx = sinistra, dx = destra)

ost. a destra	ostacolo visto col sensore numero 7, $IR[7] > 1$.
ost. dietro	ostacolo non in vista dal sensore numero 7, $IR[7] = 0$.
sparizione os.	ostacolo non in vista da nessuno dei sensori 4, 5, 6 e 7; $IR[4] = 0$ and $IR[5] = 0$ and $IR[6] = 0$ and $IR[7] = 0$.
ost. fronte	ostacolo in vista da uno dei sensori frontali, $IR[4] > 1$ or $IR[5] > 1$;
pista	vista della pista da parte di uno dei due sensori sotto; $sx\_dx \neq 0$
tot. pista	entrambi i sensori sotto vedono la pista; $sx\_dx = 11$

Tabella 7.1: Tabella eventi

stati per non appesantire il grafo, ma sono stati indicati gli eventi che indicano il passaggio di stato. I cappi del grafo che sono stati omessi, corrispondono al mancato verificarsi di tutti gli altri eventi, che causano il passaggio di stato. Nella tabella 7.1 è spiegato il significato degli eventi in termini di valori misurati dai sensori.

La macchina a stati è realizzata tramite il costrutto `switch-case` già descritto in precedenza. In riferimento alla sezione 5.2:

- gli stati 0, 1 e 2 implementano il comportamento base in fase di sorpasso, come spiegato in 5.2.2;
- lo stato 3 esegue il calcolo dell'angolo di ingresso in pista;
- lo stato 4 esegue il ri-posizionamento corretto in pista;
- gli stati 5, 6, 7, 8 e 9 gestiscono il comportamento nel caso di sparizione dell'ostacolo:
  - lo stato 5 è il corrispettivo in C del punto 5 dell'algoritmo 11;
  - lo stato 6 corrisponde al calcolo dell'orientazione del veicolo rispetto alla pista in base alle rotazioni compiute per sorpassare l'ostacolo;
  - lo stato 7 è lo stato in cui è richiesta di attuare una rotazione in senso antiorario;

- lo stato 8 è lo stato in cui è richiesta di attuare una rotazione in senso orario;
- lo stato 9 è lo stato in cui non è richiesta di attuare una rotazione.

Capitolo 8

Risultati

In questo capitolo è spiegato come sono stati svolti gli esperimenti, la realizzazione fisica e l'algoritmo implementato per eseguire il test. I veicoli sono etichettati in ordine crescente dal primo della fila all'ultimo, il primo veicolo si chiama R_1 e l'ultimo R_4. Si veda figura 8.1.

8.1 Controllore proporzionale

La verifica del corretto funzionamento di questa parte del progetto è stata svolta nel seguente modo:

- si è costruita una pista centimetrata per la misura della distanza;
- i veicoli sono posizionati in fila uno dietro l'altro, come in figura 8.1, sopra la pista, distanziati di circa 10 cm l'uno dall'altro;
- alla fine della pista è stato posto l'ostacolo costruito per simulare il robot guasto;
- nei veicoli è stato implementato il solo controllo sulla distanza;

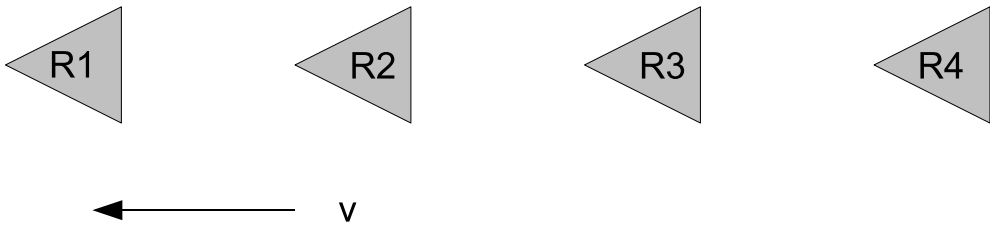

Figura 8.1: Esempio di posizionamento dei veicoli per l'esperimento.

- viene fatto partire prima il robot R_4, poi R_3 e così via sino ad R_1.

Dall'esperimento ci aspettiamo che:

- i veicoli si muovano di moto rettilineo sino a raggiungere il veicolo che hanno davanti;
- una volta raggiunto il veicolo davanti manterranno la distanza di sicurezza, impostata sui 4 cm;
- quando raggiungeranno il veicolo fermo tutti i veicoli restino fermi con una distanza l'uno dall'altro di 4 cm.

Con l'esperimento abbiamo verifato che:

- i veicoli mantengono la distanza di sicurezza;
- il veicolo R_4 parte per primo e una volta che la sua posizione ${}^1x_4$ risulta uguale a:
$$x_4 = x_3 - u_1$$
si ferma;
- quando il veicolo R_3 comincia a muoversi, anche R_4 si muove con lui mantenendo la distanza stabilita;
- lo stesso dicasi per R_2 e R_1;
- quando R_1 ha raggiunto il veicolo fermo, tutti i robot si fermano.

La distanza misurata sulla pista centimetrata risulta di 4 cm, l'errore a regime del sistema non è apprezzabile visivamente nella simulazione, di conseguenza il controllo è adatto al nostro scopo senza necessità di migliorie.

Un'ulteriore verifica è stata fatta ponendo il veicolo R_1 a distanza di circa 25cm dall'ostacolo e implementando su di esso il controllore sulla distanza. In figura 8.2 è illustrato i valori della distanza misurata (indicata nel grafico con $y(t)$) in funzione del periodo (T). Si vede che dopo un tempo di $75 \cdot T$ corrispondente a $t = 4.125\ s$ il veicolo R_1 raggiunge l'ostacolo e mantiene la sua distanza costante per un tempo indefinito. I picchi presenti nel grafico sono causati dal rumore dovuto alla luce ambientale o a possibili errori di misura.

Nella figura 8.3 è illustrato il valore della distanza misurata in funzione del tempo di campionamento (T).

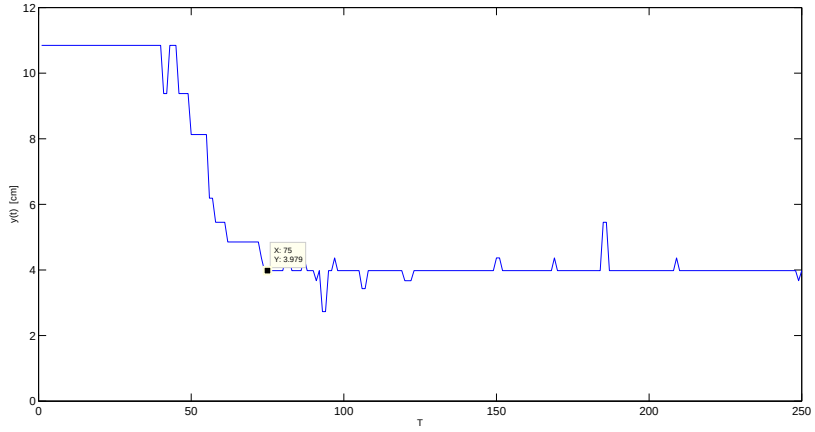

Figura 8.2: Distanza misurata dal veicolo in movimento verso un ostacolo fermo.

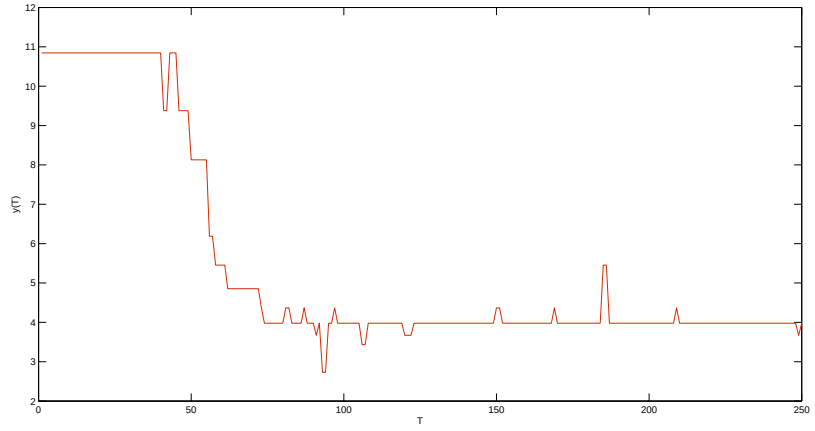

Figura 8.3: Evoluzione del sistema soggetto a controllore proporzionale, distanza misurata

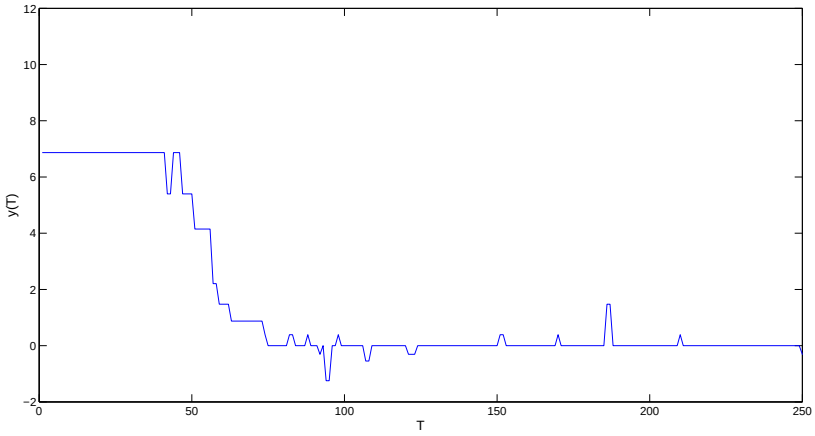

Figura 8.4: Evoluzione del sistema soggetto a controllore proporzionale, variabile velocità

Nella figura 8.4 è illustrato il valore della velocità lineare attuata in funzione del tempo di campionamento (T), in modo da vedere l'adattamento del sistema alle variazioni di misurazione.

Un altro test è stato svolto implementando nel veicolo R_1 un moto a velocità costante lungo la pista, e su R_2 è stato implementato il controllore sulla distanza. Il valore della distanza misurata del veicolo R_2, durante questo esperimento, è rappresentato in ordinata in funzione del periodo di campionamento nella figura 8.5. Si può vedere come il valore della distanza misurata resti all'interno di un certo valore limitato che permette al veicolo R_2 di seguire il veicolo R_1.

La velocità del veicolo R_2 durante questo test è graficato in figura 8.6. La linea nera indica la velocità costante del primo veicolo (v_{R_1}), si può vedere che la velocità (v_{R_2}) del veicolo R_2 assume valori compresi tra $v_{R_2} = v_{R_1} \pm 6mm/s$

8.2 Movimento circolare

Nel realizzare il moto circolare, si è applicato l'algoritmo 6 per calcolare la velocità angolare ottimale per la pista costruita. Questo algoritmo richiede di definire un tempo di campionamento fissato, che verrà fissato dal tempo di elaborazione. Il tempo necessario alle elaborazioni si può decidere solo con l'implementazione dell'algoritmo complessivo, in modo che ci sia la certezza che tutte le operazioni richieste vengano portate a buon fine. Anticipo qui

[1] Useremo la stessa notazione utilizzata nella sezione 4.

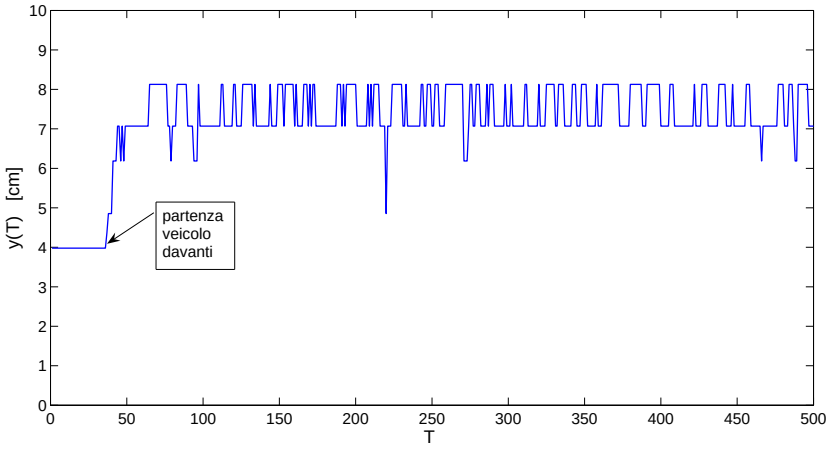

Figura 8.5: Distanza misurata dal veicolo durante inseguimento.

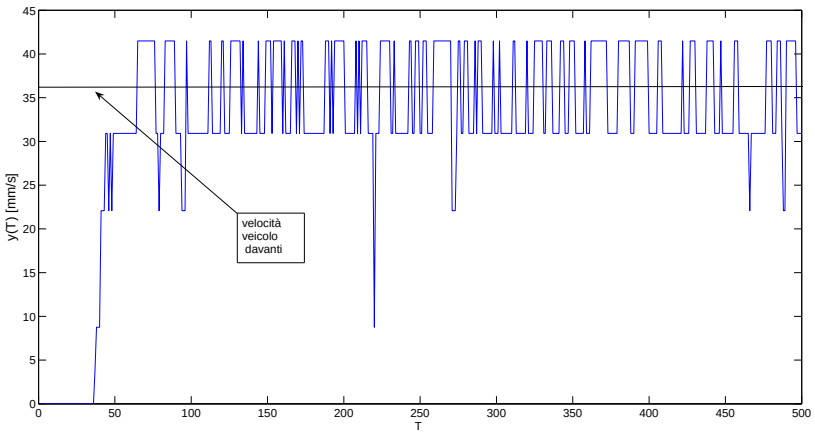

Figura 8.6: Velocità lineare durante l'inseguimento.

che il tempo di campionamento scelto è di 55 ms, la motivazione verrà data nelle sezioni successive

Come spiegato nell'algoritmo sopracitato si imposta la velocità lineare e angolare a iterazioni successive sino ad arrivare ad un valore ottimale di velocità angolare per la pista utilizzata[2] e ad un valore massimo di velocità lineare ammesso.

La pista costruita per l'esperimento ha larghezza 4 cm e un raggio di curvatura medio di 30 cm. Il raggio di curvatura è di ampiezza tale da permettere velocità lineari anche abbastanza elevate senza che il veicolo esca fuori pista. Dall'algoritmo 6 ho ottenuto che:

- la velocità angolare ottimale è pari a ± 0.4 [rad/s];
- la velocità lineare massima è di 150 mm/s.

Per verificare la robustezza dell'algoritmo e la consistenza dei valori ottenuti si è costruita una pista curvilinea con un raggio di curvatura molto più stretto, con raggio minimo di 4 cm. La larghezza della pista è rimasta invariata in modo da verificare se il valore della velocità angolare trovato precedentemente sia valido per qualsiasi tipo di pista. In questo caso la velocità lineare massima ammessa si riduce notevolmente rispetto al caso della pista precedente. I valori trovati sono:

- la velocità angolare ottimale è pari a $\pm 0.4[rad/s]$;
- la velocità lineare massima è di 60 mm/s.

Tali valori sono utilizzabili anche per la pista per il movimento circolare, e consentono di ottiene un controllo molto più robusto in quanto adattabile anche ad altre tipologie di percorso.

In figura 8.7 è illustrata la correzione sulla velocità angolare (indicata nel grafico con $y(t)$) durante la curva della pista, graficata in funzione del tempo di campionamento (T). Si noti che la pista viene percorsa in senso antiorario per cui la velocità angolare assume solo valori positivi.

8.3 Ricerca del percorso

L'esperimento per testare la ricerca del percorso è stato eseguito posizionando il veicolo all'interno della zona delimitata dalla pista, e avviato il programma (figura 5.8) per l'inseguimento del percorso.

[2]Tale valore dipende solo dalla larghezza della pista.

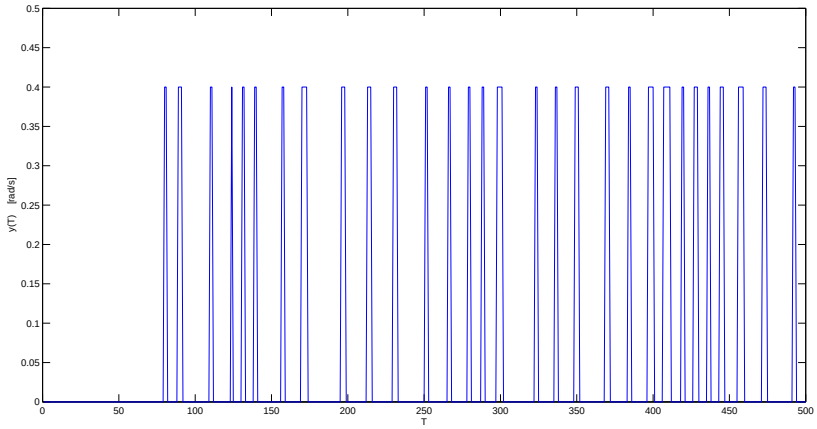

Figura 8.7: Correzione velocità angolare in curva

Questo esperimento oltre che verificare se il veicolo riesca a trovare il percorso da seguire, vuole anche testare la correttezza della funzione per il calcolo dell'angolo di ingresso in pista, spiegata nell'algoritmo 5.1.2 e illustrata nella figura 5.7.

Posto il veicolo in posizioni random all'interno del percorso delimitato dalla pista e avviato il programma; il veicolo si muove di moto rettilineo sino a quando, con uno dei due sensori posti sotto al robot, non viene individuata la pista.

L'istante in cui il robot intercetta la pista con uno dei due sensori posti sotto verrà memorizzato nella variabile t_i. Il veicolo continua il suo moto rettilineo sino all'istante di tempo in cui entrambi i sensori vedono la pista (istante di tempo memorizzato nella variabile t_f). Dopodiché esegue correttamente il calcolo dell'angolo per il posizionamento e correttamente imposta la velocità angolare che verrà mantenuta per l'intervallo di tempo di un secondo[3].

8.4 Formazione plotone

È stato fatto un esperimento per studiare il funzionamento dell'algoritmo 12 per la posizione relativa del veicolo rispetto agli altri all'interno del percorso desiderato. Sono stati eseguite diverse prove, che riporto in ordine numerico nelle seguenti sottosezioni.

[3]Come spiegato nell'algoritmo spiegato nella sezione 5.1.2.

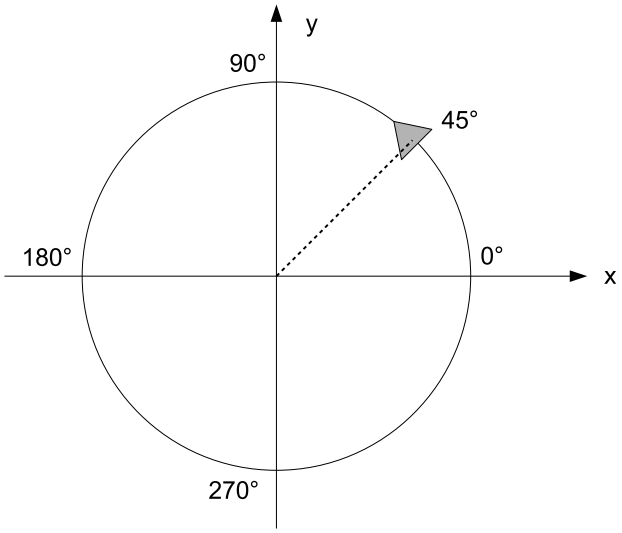

Figura 8.8: Esempio posizione assoluta del robot nella pista.

Le posizioni assolute, rispetto alla pista, del singolo veicolo R_i sono espresse tramite una misura dell'angolo in gradi, nel seguente modo:

- come indicato in figura 8.8 i punti della pista sono indicati con la notazione convenzionale che si usa per gli angoli nel piano (x,y):
  - si percorre la circonferenza in senso antiorario;
  - il punto in cui la circonferenza interseca l'asse delle x positivo corrisponde all'angolo di ampiezza 0^o;
  - il punto in cui la circonferenza interseca l'asse delle y positivo corrisponde all'angolo di ampiezza 90^o;
  - il punto in cui la circonferenza interseca l'asse delle x negativo corrisponde all'angolo di ampiezza 180^o;
  - il punto in cui la circonferenza interseca l'asse delle y negativo corrisponde all'angolo di ampiezza 270^o;
  - i restanti punti della circonferenza sono indicati con i valori intermedi degli angoli .
- i robot vengono posizionati come quello della figura, tutti orientati nella stessa direzione.

La posizione assoluta del robot R_i è indicata con p_i.

Su ogni veicolo è stato implementato il programma per l'inseguimento del percorso e il controllo sulla distanza. All'avvio del programma ogni robot esegue la verifica sulla posizione relativa rispetto al resto del plotone.

Prima prova In questa prova le posizioni assolute iniziali dei veicoli sono:

- $p_1 = 0^o$;
- $p_2 = 45^o$;
- $p_3 = 180^o$;
- $p_4 = 270^o$.

All'avvio del programma ogni veicolo scruta la sua posizione relativa rispetto al plotone. Ogni veicolo capisce di essere solo e quindi attua una velocità lineare random sino all'incontro di un altro veicolo. Nel giro di 2 minuti i veicoli si ricompattano muovendosi uno di seguito all'altro.

Seconda prova In questa prova le posizioni assolute iniziali dei veicoli sono:

- $p_1 = 45^o$;
- $p_2 = 40^o$;
- $p_3 = 315^o$;
- $p_4 = 225^o$.

All'avvio del programma ogni veicolo scruta la sua posizione relativa rispetto al plotone. Il veicolo R_1 sa di essere il primo della fila, il veicolo R_2 sa di essere l'ultimo della fila. Gli altri due veicoli sanno di essere soli nella pista e quindi attueranno una velocità lineare random che gli permette in un tempo di raggiungere il plotone in un tempo che va dai 30 secondi al minuto.

Terza prova In questa prova le posizioni assolute iniziali dei veicoli sono:

- $p_1 = 45^o$;
- $p_2 = 40^o$;
- $p_3 = 230^o$;

- $p_4 = 225^o$.

All'avvio del programma ogni veicolo scruta la sua posizione relativa rispetto al plotone. Il veicolo R_1 sa di essere il primo della fila, il veicolo R_2 sa di essere l'ultimo della fila. Il veicolo R_3 sa di essere il primo della fila, il veicolo R_4 sa di essere l'ultimo della fila. Entrambe le coppie di veicoli non potendo comunicare tra loro non sanno che ci sono altri veicoli nella pista, si formeranno due plotoni indipendenti.

Quarta prova In questa prova le posizioni assolute iniziali dei veicoli sono:

- $p_1 = 45^o$;
- $p_2 = 40^o$;
- $p_3 = 30^o$;
- $p_4 = 25^o$.

All'avvio del programma ogni veicolo scruta la sua posizione relativa rispetto al plotone. Il veicolo R_1 sa di essere il primo della fila, il veicolo R_2 sa di essere al centro della fila, lo stesso dicasi R_3, il veicolo R_4 sa di essere l'ultimo della fila. Il plotone è formato quindi tutti i veicoli si comportano seguendo l'andamento nominale del programma.

8.5 Aggiramento ostacolo

Per quel che riguarda l'algoritmo per aggirare l'ostacolo gli esperimenti sono stati svolti sull'algoritmo complessivo presentato nella sezione 7.2.5, nella figura 7.5. L'esperimento è stato svolto nel seguente modo:

- sul veicolo R_1 è stato implementato l'algoritmo citato;
- è stato posizionato sulla pista un veicolo fermo.

Il veicolo R_1 quando vede un ostacolo si ferma e aspetta il tempo prestabilito prima di iniziare il sorpasso. Il veicolo R_1 aggira l'ostacolo da sinistra, e quando si trova nuovamente sulla pista si riposiziona correttamente e continua il suo cammino.

Lo stesso esperimento è stato svolto con 2 veicoli fermi, posizionati uno di seguito all'altro sulla pista. Il veicolo R_1 aggira perfettamente i 2 ostacoli e si riposiziona in pista. L'esperimento con 3 veicoli fermi ha dato lo stesso

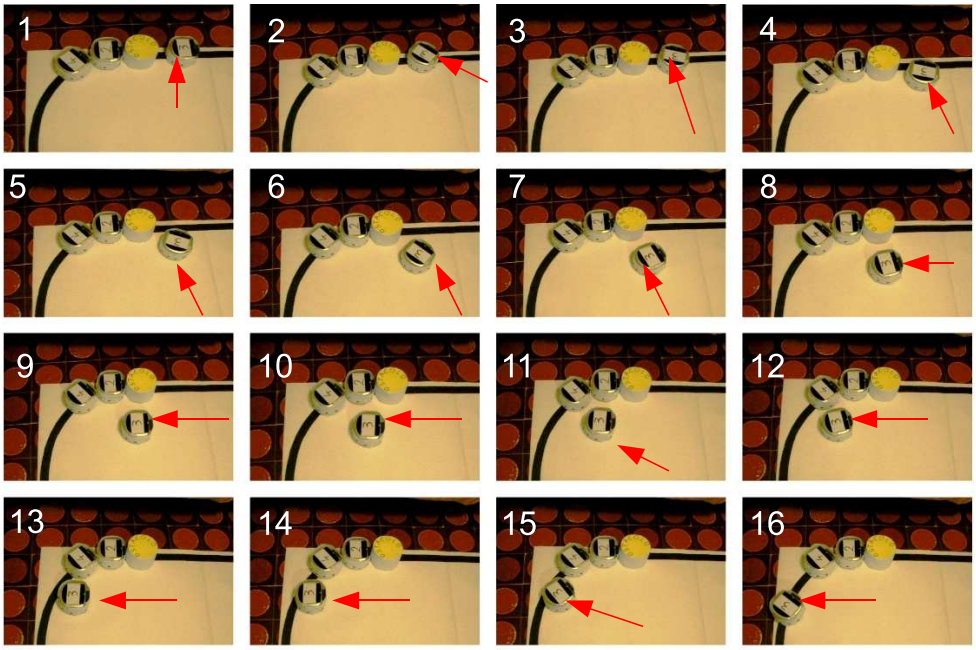

Figura 8.9: Un veicolo compie il sorpasso di tre ostacoli uno di seguito all'altro.

risultato ed è illustrato dai fotogrammi del filmato rappresentati nella figura 8.9.

Successivamente, la prova è stata eseguita nel seguente modo:

- sul veicolo R_1 e R_2 è stato implementato l'algoritmo citato;
- i veicoli R_1 e R_2 sono posizionati sulla pista uno di seguito all'altro;
- un veicolo fermo è stato posizionato in un punto della pista lontano dagli altri due veicoli.

All'avvio del test i due veicoli si muovono in plotone mantenendo la distanza di sicurezza, quando R_1 vede l'ostacolo rallenta e infine si ferma. Quando il tempo di attesa è trascorso il veicolo R_1 inizia la fase di sorpasso e il veicolo R_2 aspetta ulteriormente che la scena sia libera. quando il veicolo R_1 non è più in vista dal veicolo R_2 allora quest'ultimo avanza e si posiziona davanti all'ostacolo. Quando è trascorso l'intervallo di tempo necessario per discriminare tra ostacolo e un altro veicolo allora anche R_2 inizierà la fase di sorpasso. Eseguito il sorpasso entrambi i veicoli si riposizionano correttamente in pista.

Il test è stato eseguito su tutti i veicoli contemporaneamente, può capitare che più veicoli inizino il sorpasso contemporaneamente. Questo è dovuto alla

simultaneità dell'operazione, il tempo di attesa non è tale da permettere ai veicoli davanti di eseguire le manovre del sorpasso senza che altri vogliano compiere le stesse operazioni. Con gli opportuni tempi di attesa, diversi a seconda della posizione relativa del veicolo, si è riusciti ad ottenere un ordine di sorpasso dettato dalla posizione del veicolo all'interno del plotone.

8.5.1 Il plotone

Il programma che racchiude tutte le funzionalità è stato attuato su tutti i veicoli da cui è emerso il comportamento dell'intero sistema.

L'insieme dei veicoli, il plotone, si comporta come un unico sistema intelligente seppur non dotato di una sua intelligenza, in quanto non è presente il controllo centralizzato.

Come verifica di tale comportamento cito l'esperimento attuato sull'intero plotone in cui un veicolo dopo l'altro aggira l'ostacolo aspettando il suo turno.

Si veda in figura 8.10 come ogni veicolo del plotone aspetti il proprio turno per sorpassare e una volta eseguito il sorpasso si riposizioni correttamente in pista. È importante notare, nel fotogramma numero 6, che i veicoli 2 e 3 iniziano la fase di sorpasso contemporaneamente in quanto entrambi sanno di essere in posizione centrale, ma non possono sapere quanti veicoli ci sono prima. Si noti nel fotogramma numero 7 che il veicolo 3, seppur abbia iniziato la fase di sorpasso contemporaneamente al veicolo 2, si accorge che non è il suo turno e si riposiziona in pista.

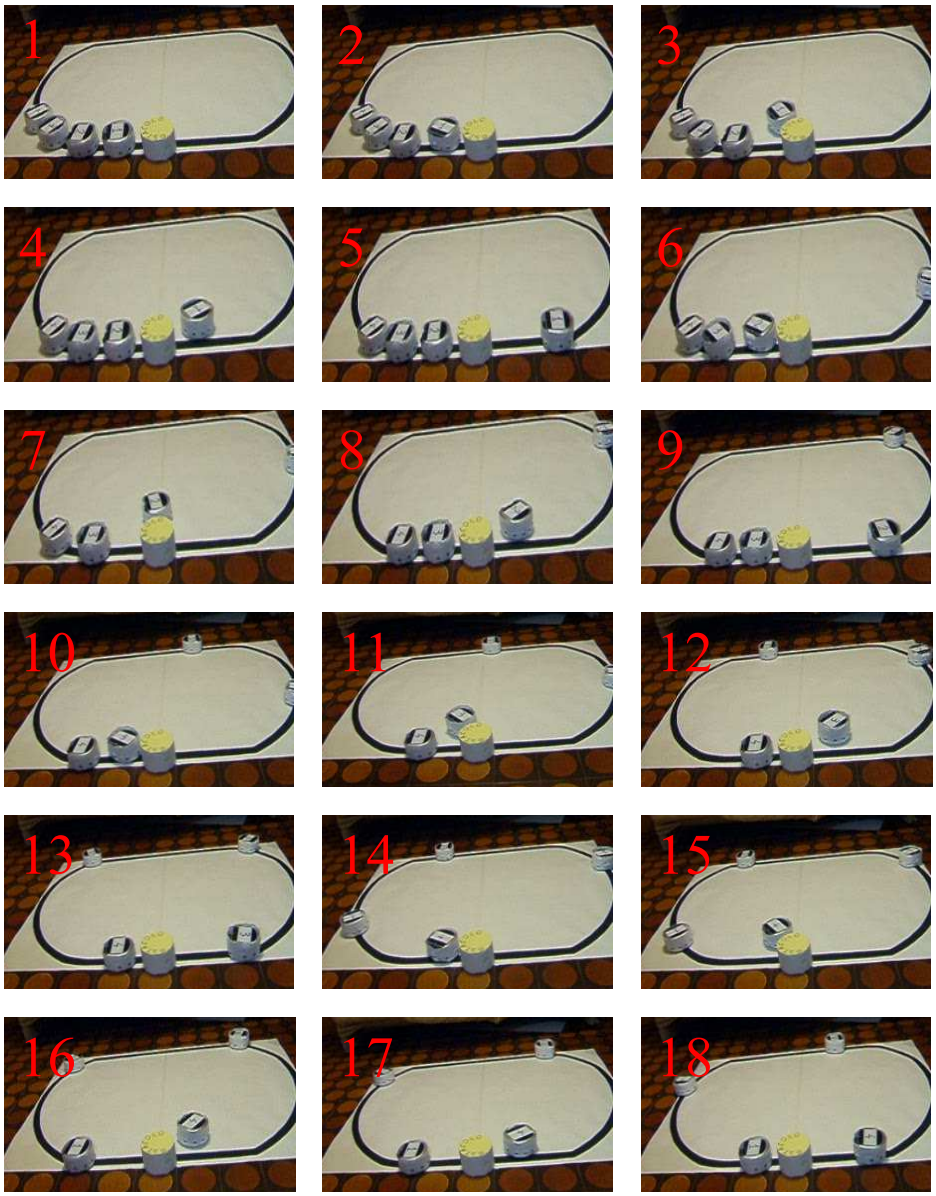

Figura 8.10: Fotogrammi del plotone che esegue l'algoritmo del sorpasso

Capitolo 9

Conclusioni

In questa tesi è stato trattata la progettazione di un sistema di controllo distribuito ed è stata eseguita l'implementazione su dei robot Khepera III. Le funzionalità progettate e implementate sono:

- realizzazione di un algoritmo per la ricerca del percorso desiderato, con conseguente movimento in formazione compatta del plotone, il percorso è stato scelto di forma circolare;
- realizzazione di un controllore proporzionale per il mantenimento della distanza nella formazione del plotone;
- realizzazione di un algoritmo di sorpasso per evitare i veicoli fermi a causa un malfunzionamento.

La prima funzionalità implementata è l'inseguimento del percorso desiderato, successivamente si è aggiunta la funzionalità per il controllo sulla distanza di sicurezza e, infine, si è realizzato un algoritmo per il sorpasso.

L'intero programma è stato realizzato in un linguaggio comprensibile al robot, una volta implementato su di esso è stato verificato il corretto funzionamento dell'algoritmo.

Il sistema di controllo distribuito progettato soddisfa le specifiche richieste sia per quel che riguarda l'inseguimento del percorso sia per quanto riguarda il movimento del plotone in formazione compatta. I veicoli muovendosi in formazione mantengono la distanza di sicurezza e aggirano l'ostacolo. Seppur sia stato implementato un algoritmo per l'ordine di precedenza nel sorpasso, i veicoli non possono comunicare tra loro e dunque un veicolo centrale non sarà distinguibile nella sua posizione. Tutti i veicoli centrali al plotone sanno solamente di avere un veicolo davanti e uno dietro, ma non la posizione assoluta. Per cui può capitare che i veicoli centrali decidano di sorpassare

simultaneamente. Con opportuni tempi di attesa si riesce ad ovviare a questo inconveniente che risulta comunque un punto di criticità per un numero elevato di veicoli. Può, quindi, essere risolto implementando un sistema di comunicazione tra i veicoli.

Il sistema distribuito implementato sui singoli robot ha permesso di ottenere un comportamento emergente del plotone. Si ha, quindi, un unica entità che attua comportamenti come se ogni veicolo fosse una sua parte, ma senza necessità di realizzare un controllo centralizzato. Compare, quindi, il fenomeno del *comportamento emergente*, la situazione nella quale un sistema esibisce proprietà sulla base delle leggi che governano le sue componenti.

Questo progetto pone le basi per ulteriori lavori di progettazione su sistemi multiagente implementati su robot Hilare-type. È possibile implementare funzionalità differenti, oppure, le stesse funzionalità proposte in questa tesi si possono usare su percorsi differenti. È possibile sfruttare gli altri dispositivi del veicolo, quali gli utrasuoni per implementare il controllore su una distanza di sicurezza più ampia o per realizzare nuove funzionalità.

In ambito più generale il sistema di controllo progettato può essere implementato, con le opportune modifiche, su altri veicoli come ad esempio le automobili per il controllo sulla distanza di sicurezza affinché si evitino tamponamenti, oppure il controllo sull'inseguimento del percorso per evitare che il veicolo vada fuori strada.

Bibliografia

[1] G. Antonelli, F. Arrichiello, and S. Chiaverini. The entrapment/escorting mission for a multi-robot system: Theory and experiments. In *Proceedings 2007 IEEE International Conference on Advanced Intelligent Mechatronics*, 2007.

[2] A. Baccara. Khepera III for Dummies, 2009. release 1.0, Rapporto interno laboratorio automatica.

[3] A. Balestrino. Inversione di matrice:metodo di gauss jordan. `http://webm.dsea.unipi.it/~balestrinow/public_html/`, 2003.

[4] JA Benayas, JL Fernández, R. Sanz, and AR Diéguez. The beamcurvature method: a new approach for improving local real time obstacle avoidance. In *Proceedings of the 15th Triennial World Congress of the International Federation of Automatic Control IFAC, Barcelona, Spain*, 2002.

[5] C. Bonivento, C. Melchiorri, and R. Zanasi. *Sistemi di controllo digitale*. Progetto Leonardo, Bologna, 1995.

[6] J. Borenstein and Y. Koren. The vector field histogram–fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, 1991.

[7] K-Team Corporation. *KheperaIII user manual*, March 2008. `info@k-team.com`,`www.k-team.com`.

[8] K-Team Corporation. *KoreBot user manual*, March 2008. `info@k-team.com`,`www.k-team.com`.

[9] K-Team Corporation. *KoreMotor user manual*, March 2008. `info@k-team.com`,`www.k-team.com`.

[10] J.J. Di Stefano, A.R. Stubberud, and I.J. Williams. *Regolazione automatica,problemi risolti*. McGraw-Hill, 1994.

[11] F. Fahimi. *Autonomous Robots: Modeling, Path Planning, and Control*. Springer, 2008.

[12] A. Giua. Appunti sui Sistemi Ibridi, 2009. Appunti per il corso di Sistemi Ibridi.

[13] A. Giua and C. Seatzu. *Analisi dei sistemi dinamici*. Springer Verlag, 2006.

[14] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90, 1986.

[15] Y. Lan, G. Yan, and Z. Lin. A hybrid control approach to multi-robot coordinated path following. *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 3032 –3037, dec. 2009.

[16] L. Lapierre, R. Zapata, and P. Lepinay. Combined path-following and obstacle avoidance control of a wheeled robot. *The International Journal of Robotics Research*, 26(4):361, 2007.

[17] S. Lee and J.H. Park. Dynamic path-following using temporary path generator for mobile robots with nonholonomic constraints. In *IEEE INTERNATIONAL CONFERENCE ON SYSTEMS MAN AND CYBERNETICS*, volume 6. Citeseer, 1999.

[18] Z. Lin, B. Francis, and M. Maggiore. *Distributed Control and Analysis of Coupled Cell Systems*. VDM Verlag Saarbr
ucken, Germany, Germany, 2008.

[19] R. Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory, 2004.

[20] T.M. Quasny, L.D. Pyeatt, and J.L. Moore. Curvature-velocity method for differentially steered robots. In *Proceedings of the IASTED International Conference on Modelling, Identification, and Control*. Citeseer, 2003.

[21] M.I. Ribeiro and A.R. Pais. Obstacle Avoidance. *Institute for Systems and Robotics, Lisboa, Portugal*, 2005.

[22] Saverio Sanna. Controlli Automatici, 2004. Appunti per il corso di controlli automatici.

[23] D. Soetanto, L. Lapierre, and A. Pascoal. Adaptive, non-singular path-following control of dynamic wheeled robots. In *IEEE Conference on Decision and Control*, 2003.

[24] I. Ulrich and J. Borenstein. VHF*: Reliable Obstacle Avoidance for Fast Mobile Robots. In *IEEE Int. Conf. on Robotics and Automation*, pages 2505–2511, 2000.

Appendice A

Sistemi Ibridi

Nel linguaggio comune si definisce ibrido un sistema formato da componenti di natura diversa. All'interno dell'automatica si usa tale termine con uno specifico significato: un *sistema ibrido* è un sistema viene descritto mediante un modello che unisce dinamiche ad avanzamento temporale con dinamiche ad eventi discreti.[13]

A.1 Automa Ibrido

Il principale modello usato per lo studio dei sistemi ibridi è detto Automa Ibrido. Si tratta di automa i cui stati, o locazioni, descrivono lo spazio di stato discreto. Ad ogni locazione è associata un evoluzione temporale di solito descritta da un equazione differenziale. Gli archi tra locazioni descrivono gli eventi discreti[12]. L'automa è detto *autonomo* se non è soggetto agli ingressi esterni, in caso contrario è detto *non autonomo*. In questa appendice è descritto il solo automa ibrido non autonomo in quanto è il modello più generale.

A.1.1 Automa ibrido non autonomo

L'evoluzione del sistema dipende, oltre che dallo stato, anche dagli ingressi che possono essere continui e discreti.

Definizione 16 *Un automa ibrido* [1] *è:*

$$H = (L, X, A, I, E, y_0, U, D)$$

[1] Questa è la definizione generale di automa ibrido, che comprende come caso particolare quella di automa ibrido autonomo, in cui $U = 0$, $D = 0$

dove:

- *L è lo spazio di stato discreto, cioè un insieme finito di locazioni di cardinalità s. $X \subseteq \Re^n$ è lo spazio di stato continuo, $y_0 \in L \times X$ è lo stato iniziale.*

- *$U \subseteq \Re^p$ è l'insieme degli ingressi continui. Ogni ingresso*

$$u(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_r(t) \end{bmatrix} \in U$$

  è un vettore con p componenti.

- *$D = d_1, d_2, \ldots, d_q$ insieme degli ingressi discreti. Ogni ingresso d_j è un evento (un fronte di salita o di discesa) o una condizione logica (un segnale booleano costante).*

- *La definizione di attività è data dalla funzione $A : L \to (X \times U \to \Re^n)$ che specifica l'attività*

$$f_l : X \times U \to \Re^n.$$

  Dunque nella locazione l vale l'equazione differenziale:

$$\dot{x} = f_l(x, u) \in \Re^n.$$

- *Per la presenza di ingressi continui, la definizione di invariante è data dalla funzione $I : L \to 2^{X \times U}$: che specifica l'invariante $I_l \subseteq X \times U$ associato alla generica locazione l. È possibile restare nella locazione l solo se la coppia (x,u) formata dallo stato continuo x e dall'ingresso continuo u appartiene a I_l.*

- *La presenza di ingressi continui e discreti modifica la struttura degli archi.*

  Per prima cosa l'ingresso continuo influenza la soglia, che ora è definita come

$$g_e \subseteq X \times U.$$

Essa specifica che la transizione da l a l' puó verificarsi all'istante τ solo se $(x(\tau^-), u(\tau^-)) \in g_e$. Inoltre l'ingresso continuo influenza la soglia, che ora è definita come

$$j_e : X \times U \to X.$$

Si supponga che la transizione e venga eseguita all'istante τ quando lo stato continuo vale $x(\tau^-) = x^-$ e l'ingresso continuo vale $u(\tau^-) = u^-$. Dopo l'esecuzione della transizione lo stato continuo passa al valore $x(\tau) = j_e(x^-, u^-)$. Infine l'insieme degli archi viene ora definito come

$$E \subseteq L \times 2^D \times G \times J \times L.$$

L'arco

$$e = (l, d_e, g_e, j_e, l')$$

dove il controllo discreto vale

$$d_e = d_1, d_2, \ldots, d_k = d \wedge \ldots \wedge d_k \subseteq D$$

indica che la transizione da l a l' avviene se $(x^-, u^-) \in g_l$ e se si verificano tutti gli eventi $d_j (j = 1, \ldots, k)$ (o se tali condizioni logiche sono tutte vere).

In questo caso il controllo discreto deve essere 'vero' per poter eseguire la transizione, che viene detta transizione controllata. In genere la condizione $d_e \in 2^D$ puó essere l'insieme vuoto. In tal caso tale transizione è piú semplicemente descritta dall'arco $e = (l, g_e, j_e, l')$, omettendo la condizione d_e si parla di transizione autonoma.

È importante infine osservare che in una transizione controllata e se $(x^-, u^-) \in g_l$ e d_e è vera, si suppone che la transizione avvenga sempre: il controllo forza la transizione.

Appendice B

Sistemi interconnessi

Questa appendice riporta alcuni richiami teorici sui sistemi interconnessi. Esistono tre modi possibili: connessione in cascata, in parallelo e in retroazione (o reazione). Le interconnessioni tra i vari sottosistemi sono possibili solo attraverso le variabili esterne, cioé tramite gli ingressi e le uscite.

B.1 Connessione in cascata

Due o più sistemi si dicono *connessi in cascata*, o *in serie*, quando una funzione dell'uscita del primo costituisce l'ingresso del secondo. Il caso più semplice si ha quando l'uscita del primo sistema costituisce direttamente l'ingresso del secondo. I sistemi in cascata se rappresentati con uno schema a blocchi sono costituiti da blocchi connessi in serie come in figura B.1, e sono equivalenti ad unico blocco G risultante dal prodotto delle funzioni di trasferimento dei singoli sottosistemi. Si indichino con $G_1(s)$ e $G_2(s)$ due generiche funzioni di trasferimento dei singoli sottosistemi. La funzione di trasferimento totale dei due sistemi in cascata sarà:

$$G(s) = G_1(s) \cdot G_2(s).$$

E generalizzando nel caso di n blocchi:

$$G(s) = \prod_{i=1}^{i=n} G_i(s).$$

Come il prodotto, la connessione in cascata gode della proprietà commutativa.

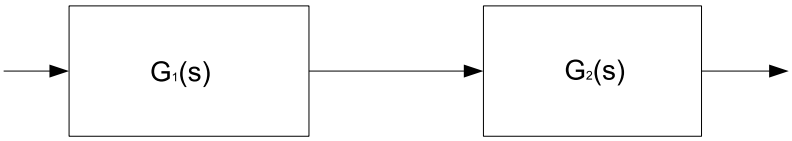

Figura B.1: Connessione in cascata

B.2 Connessione in parallelo

Due o piú sistemi si dicono *connessi in parallelo* quando i loro ingressi sono funzioni di un'unica grandezza; le loro uscite possono poi combinarsi secondo un'opportuna legge funzionale per dar luogo a un'uscita unica, o meno. Il caso piú semplice si ha quando i due sistemi hanno lo stesso ingresso e le loro uscite si sommano (si veda figura B.2). Si indichino con $G_1(s)$ *e* $G_2(s)$ due generiche funzioni di trasferimento dei singoli sottosistemi. La funzione di trasferimento totale dei due sistemi in parallelo sarà:

$$G(s) = G_1(s) + G_2(s).$$

E generalizzando nel caso di n blocchi:

$$G(s) = \sum_{i=1}^{i=n} \alpha_i G_i(s).$$

Dove α_i indica il generico segno dell'operazione di somma algebrica, in quanto in una connessione in parallelo con combinazione delle uscite, i due sottosistemi possono essere sia sommati che sottratti..

B.3 Connessione in retroazione

Due o piú sistemi si dicono uno in retroazione all'altro quando i rispettivi ingressi sono funzioni ciascuno dell'uscita dell'altro. Il caso più semplice si ha quando l'uscita del primo sistema costituisce direttamente l'ingresso del secondo e l'uscita di questo si somma (retroazione positiva) o si sottrae (retroazione negativa) a un'altra grandezza per dare l'ingresso del primo sistema.

Il sistema illustrato nella figura B.3 è l'esempio più semplice di *sistema di controllo retroazionato* con retroazione negativa, ed è costituito appunto da due blocchi connessi in retroazione.

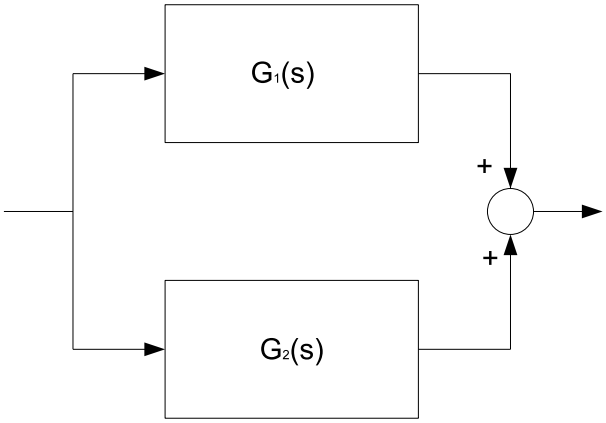

Figura B.2: Connessione in parallelo

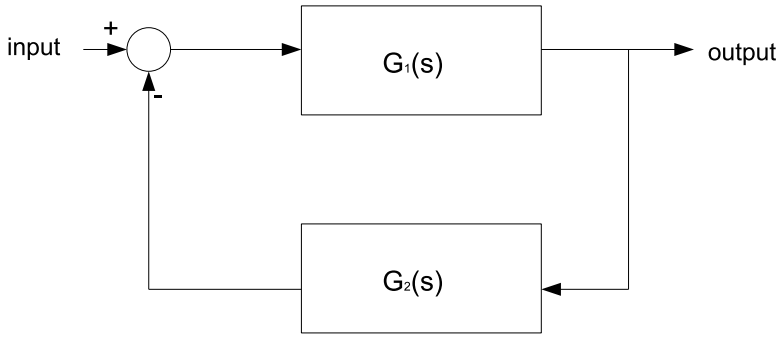

Figura B.3: Connessione in retroazione

Appendice C

Sistemi di controllo

I sistemi di *regolazione* sono dei particolari sistemi di controllo in cui il segnale in ingresso (segnale di riferimento o *set point*) è costante e hanno lo scopo di opporsi agli effetti dei disturbi affinché l'uscita converga al valore desiderato.[22]

C.1 Regolatori

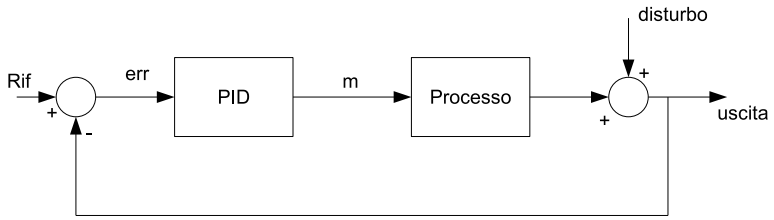

Figura C.1: Schema standare di un regolatore PID

Lo schema standard di un regolazione a retroazione unitaria è illustrato in figura:C.1.

Il segnale *rif* è il segnale desiderato, il valore che dovrà assumere l'uscita. La differnza tra l'uscita e rif è l'*errore* (err), cioè lo scostamento tra il valore desiderato dell'uscita (rif) e il valore effettivo misurato. Il segnale (m), in uscita dal PID, è il segnale controllante, elaborato dal regolatore che agisce sul processo. A seconda del tipo di errore a regime del sistema (*comportamento in regime permanente*) soggetto ad un ingresso a gradino, abbiamo due tipi di regolazioni:

- regolazione astatica, l'errore a regime è nullo. Ció accade quando nella catena diretta c'è almeno un polo nell'origine;

Tipo	Funzione di trasferimento (in s)
P	K_P
I	$\frac{K_I}{s}$
PD	$K_P + K_D s = K_P(1 + \tau_D s)$
PI	$K_P + \frac{K_I}{s} = K_P(1 + \frac{\tau_I}{s})$
PID	$K_P + \frac{K_I}{s} + K_D s = K_P(1 + \frac{\tau_I}{s} + \tau_D s)$

Tabella C.1: classificazione regolatori

- regolazione statica, l'errore a regime è finito non nullo. La catena diretta non ha poli nell'origine.

C.1.1 Classificazione dei regolatori

In base al tipo di azione di controllo sviluppata si usa classificare i regolatori in:

- regolatori P, ad azione proporzionale;
- regolatori I, ad azione integratrice;
- regolatori PD, ad azione proporzionale-derivatrice;
- regolatori PI, ad azione proporzionale-integratrice;
- regolatori PID, ad azione proporzionale-integratrice-derivatrice;

Le funzioni di trasferimento (nel dominio di laplace) ideali dei blocchi controllanti che svolgono tali azioni sono illustrati in tabella:C.1. $\tau D = \frac{K_D}{K_P}$ *tempo dell'azione derivatrice* e $\tau I = \frac{K_I}{K_P}$ *tempo dell'azione integratrice.*

Nel caso di *azione proporzionale (P)* aumentando K_P si riduce l'errore a regime permanente ma si agisce anche sul transitorio con effetti che possono essere negativi. Col crescere di K_P, si aumenta la durata del transitorio con aumento delle oscillazioni dovute alla riduzione dello smorzamento o addirittura instabilità. Tra gli effetti positivi c'è la rapidità di risposta che aumenta al crescere di K_P.

Nel caso di *azione Integratrice (I)* K_I rimane un parametro libero, se pur un suo aumento può avere effetti qualitativamente analoghi a quelli dell'aumento di K_P nel caso precedente. Sia K_I che K_P sono valori di guadagno a ciclo aperto.

Nel caso di *azione proporzionale-derivatrice (PD)* il guadagno K_P influisce oltre che sulla precisione anche sulla stabilità e sul transitorio. L'effetto stabilizzante dell'azione derivatrice può consentire di scegliere valori più elevati di K_P a parità di altre condizioni.

Nel caso di *azione proporzionale-integratrice (PI)* essa si può interpretare come:

- parallelo di un azione proporzionale e un azione integratrice.
- cascata di un azione integratrice e di una proporzionale-derivatrice. DISEGNO

$$C_{PI}(s) = K_P + \frac{K_I}{s} = \frac{K_I}{s}(1 + \tau_I s)$$

Si può fare in modo che i poli a ciclo chiuso dopo l'aggiunta dell'azione PI siano più o meno nella stessa posizione rispetto al caso senza azione PI;in tal modo non si pregiudica la stabilità migliorando contemporaneamente il comportamento a regime.

È comodo considerare l'azione *PI* secondo la seconda schematizzazione, grazie alla quale si può affermare che tale tipo di azione permette di agire sia sulla precisione a regime (grazie al polo nell'origine dell'azione integratrice) sia sulla stabilità (grazie all'azione proporzionale-derivatrice). Si ha inoltre il parametro K_I per influire sul comportamento transitorio.

L'azione di controllo più generale fra quelle elencate sopra è l'*azione proporzionale-integratrice-derivatrice (PID)*. La cui funzione di trasferimento può essere scritta nella forma:

$$C_{PID}(s) = K_P + \frac{K_I}{s} + K_D s = K_P \frac{(1 + \tau_I s + \tau_I \tau_D s^2)}{\tau_I s} \qquad \text{(C.1)}$$

Il polo nell'origine corrispondente all'azione integratrice è responsabile del comportamento in regime permanente. Il termine trinomio al numeratore corrisponde ad un'azione di controllo legata all'errore, alla sua derivata prima e seconda. In questo caso gli effetti stabilizzanti possono essere migliorati ulteriormente rispetto all'azione PD.

Tutte le funzioni di trasferimento dei regolatori esaminate sin'ora sono ideali in quanto non tengono conto delle proprietà di causalità che ogni componente reale deve soddisfare,tale proprietà impone il vincolo che il grado del denominatore sia maggiore o uguale a quello del numeratore ($n \geq m$). Prendiamo come riferimento il modello più generale di regolatore,cioè un PID,invece del comportamento ideale descritto (C.1) si potrò caratterizzare il comportamento reale con un funzione di trasferimento del tipo:

$$C_{PIDreale}(s) = G(s)C_{PID}(s) = G(s)K_P\frac{(1+\tau_I s+\tau_I\tau_D s^2)}{\tau_I s} \qquad \text{(C.2)}$$

La funzione di trasferimento $G(s)$ sarà prossima a 1 tanto più il comportamento effettivo del regolatore si avvicina al comportamento reale. Si assume per $G(s)$ la seguente forma:

$$G(s) = \frac{1}{(1+\tau s)}$$

Si terrà conto del comportamento reale dei regolatori introducendo una costante di tempo τ il ci valore sarà sufficientemente piccolo, rispetto alle altre costanti di tempo che caratterizzano le proprietà dinamiche del sistema studiato. Ciò equivale ad aggiungere un polo in alta frequenza il cui peso ai fini progettuali sarà del tutto trascurabile.

Appendice D

File C

```
/**
*
*   programma complessivo
*
*                @author = Marianna Stara
*                @date = 04/2010
*
*
*/

/*      librerie        */
#include "build-korebot-2.6/include/korebot/korebot.h"
#include <stdio.h>
#include <math.h>

/*      parametri       */
#define MAXBUFFERSIZE 100
#define CONST_VEL 144
#define PERIODO 55
#define SECONDO 1000/PERIODO
#define TEMPO_ATTESA_SORPASSO_FIRST 5*SECONDO
#define TEMPO_ATTESA_SORPASSO_CENTRO 10*SECONDO
#define TEMPO_ATTESA_SORPASSO_LAST 15*SECONDO
#define TEMPO_DECISIONE 15*SECONDO
#define K 10
#define REF 3.9788
#define VEL_FIRST 40
#define VEL_SORPASSO 30
#define MAX_VALORE_VAR 6*SECONDO
```

```
#define TEMPO_AGGIORNAMENTO_POSIZIONE 30*SECONDO

/*      variabili globali       */
static knet_dev_t * dsPic;
static knet_dev_t * mot1;
static knet_dev_t * mot2;
static int flag;

/*      interfaccia funzioni principali */
int initMot(knet_dev_t *hDev)
int initKH3( void )
int motSpeed( int m1, int m2)


/*              funzioni                */
float calc_dist (float sensIR_val)

int set_velocita(float v, float w)//v[mm/s] w[rad/s]

float line_following(int sx_dx)

int sorpasso_pista (float * v, float * w,
                    char * IRBuffer, int sx_dx )

int sensoriIR_pista(int IRsx, int IRdx)

float rand_vel(void)

int calc_tempo_attesa(int first, int last)

int dove_sono(int first, int last,
                      char *IRBuffer)

int main(int argc, char *argv[])
{
 char Buffer[MAXBUFFERSIZE];
 char IRBuffer[MAXBUFFERSIZE];
 int state = 0;
 int nextstate = 0;
 int tempo_attesa_sorpasso=0;
 int i = 0;
 int temp=0;
 int var= 0;
```

```
 int count = 0;
 int decisione=0;
 int last = 0;
 int first = 0;
 int t;
 float g;

 float vel_lin = 0;
 float next_vel=0;
 float vel_ang = 0;

 flag = 0;

 srand ( time(NULL) );

 kb_time_t start, end, dif, tempo;

 set_velocita(vel_lin,vel_ang);

 if(!initKH3())
        {
                printf("Init␣ok...\r\n");
  kh3_revision((char *)Buffer, dsPic);
  }
 printf("inizializzazione␣completata\n");

while(1)
{
start = kb_getTime();
kh3_proximity_ir((char *)IRBuffer, dsPic);
g=ref-calc_dist(((float)(IRBuffer[8]+IRBuffer[10]))/2);

temp=sensoriIR_pista(IRBuffer[22],IRBuffer[20]);

t++;

switch(state){
 case(0)://situazione iniziale
       {
        vel_lin=0.0;
        vel_ang=0.0;
        if(decisione < TEMPO_DECISIONE)
          {
```

```
            decisione++;
            nextstate=0;
            printf("si␣parte");
            break;
            }
          else
            {
            t=0;//azzera il tempo
            if(dove_sono(first, last, IRBuffer))
             {
              if((first == 0))
               {
                nextstate = 3;
                decisione=0;
                break;
               }
              else
               {
                if((first == 1)&&(last == 0))
                 {
                  nextstate = 1;
                  decisione=0;
                  break;
                  }
                else
                 if((first == -1)&&(last == -1))
                  {
                   nextstate = 2;
                   decisione=0;
                   next_vel=rand_vel();
                   break;
                  }
                }
              } else
                printf("errore␣posizione␣relativa");
            }
          }//fine case(0)
 case(1):PRIMO
          {
           vel_lin = VEL_FIRST;
           vel_ang = line_following(temp);
           if(t<TEMPO_AGGIORNAMENTO_POSIZIONE)
            {
```

```
    if((IRBuffer[8]>1)&& (IRBuffer[10]>1))
     {
      nextstate=3;
      first=0;
      if(IRBuffer[18]<=2)
      {
       last=1;
       break;
       }
      else
       {
        last=0;
        break;
              }
     }
    else
     {
      nextstate = 1;
      break;
     }
                }
else
  {
   t=0;//azzera il tempo
   if(dove_sono(first, last,IRBuffer))
    {
     if((first == 0))
     {
      nextstate = 3;
      break;
                  }
      else
         {
       if((first == 1) && (last == 0))
        {
         nextstate = 1;
         break;
         }
       else
        if((first == -1) && (last == -1))
         {
          nextstate = 2;
          next_vel=rand_vel();
```

```
                    break;
                    }
                  }
                } else
                    printf("errore posizione relativa");
              }
            }//fine case(1)
case(2)://SOLO
            {
             vel_lin = next_vel;
             vel_ang = line_following(temp);
             if(dove_sono(first, last,IRBuffer))
             {
              t=0;//azzera il tempo
              if((first == 0))
               {
                 nextstate=3;
                 break;
                 }
               else
                {//nessuno davanti
                 if((first == -1) && (last==-1))
                  {
                   nextstate=2;
                   next_vel=rand_vel();
                   break;
                               }
                 else
                  if((first == 1) && (last==0))
                   {
                    nextstate=1;
                    break;
                    }
                 }
              }else
                printf("errore posizione relativa");
             }//fine case(2)
case(3)://CENTRALE
            {
             vel_lin = - K*g;
             vel_ang = line_following(temp);
             if (vel_lin < 10 && vel_lin > - 10)
              {
```

```
            count++;
            nextstate = 4 ;
            break;
                }
          else
           {
            if(t>= TEMPO_AGGIORNAMENTO_POSIZIONE)
              {//controlla la tua posizione
               t=0;//azzera il tempo
               if(dove_sono(first, last,IRBuffer))
                {
                 if((first == 0))
                         {
                   nextstate = 3;
                   break;
                   }
                        else
                  {
                   if((first == 1) && (last == 0))
                    {
                     nextstate = 1;
                     break;
                    }
                   else
                    if((first == -1) && (last == -1))
                     {
                      nextstate = 2;
                      next_vel=rand_vel();
                      break;
                     }
                    }
                } else
                    printf("errore posizione relativa");
              }
             else
              {
               nextstate = 3 ;
               break;
               }
             }
         }//fine case(3)
case(4):
      {//attesa per capire se è un ostacolo
```

```
        vel_lin = - K*g;
        vel_ang =  0 ;
        var = 0;
        tempo_attesa_sorpasso=
                        calc_tempo_attesa(first,last);
        if (count==tempo_attesa_sorpasso)
         {
         if((IRBuffer[6]>0)&&(IRBuffer[10]==0)&&
                  (IRBuffer[8]!=0))
          {
           count -= 20;
           nextstate = 4 ;
           break;
           }
         else
          {
           count = 0;
           nextstate = 5;
           break;
           }
          }
        else
         {
          if ((vel_lin < 20)&&(vel_lin >-20))
           {
             count++;
             nextstate=4;
             break;
                   }
           else
            {
             count = 0;
             nextstate =3;
             break;
             }
          }
        }
case(5)://sorpasso
     {
      if(sorpasso_pista(&vel_lin,&vel_ang,
            IRBuffer,temp))
       {
        nextstate = 5;
```

```
          break;
         }
        else
         {
          nextstate =1;
                 t=0;
                 break;
         }
        }//fine case(5)
        }//fine macchian a stati
set_velocita(vel_lin, vel_ang);
state = nextstate;

end = kb_getTime();
dif = end - start;

if(PERIODO >= dif) // funzionamento nominale
{
tempo = PERIODO - dif;
usleep (tempo*1000);// trasformazione tempo in us
}
else// eccezione
{
printf("ERRORE!!!␣è␣richiesto␣un␣tempo
maggiore␣del␣periodo␣di␣campionamento");
        }

}//fine ciclo while
return 0;

 } //fine main
```

Appendice E

Script Matlab

E.1 Script di interpolazione

```
close all

n=2;%grado polinomio approssimatore
m=15;% dimensioni del vettore dei nodi
% valori dei sensori IR,IRm
x=[0,1,2,3,4,5,6,7,8,11,12,13,14,15];
%ordinate funzione (le distanze)
y=[10.6,8.32,5.87,4.75,4,4,3,3,3,2,2,2,2,1];

xi= [0: 1: m]%punti di valutazione
% polyfit restituisce i coefficienti dell'equazione
%che fitta i valori sperimentali
c2= polyfit(x,y,n);
% valutazione del polinomio nei punti xi
yi2=polyval(c2,xi);

n=3;
c3= polyfit(x,y,n);
yi3=polyval(c3,xi);

n=4;
c4= polyfit(x,y,n)
yi4=polyval(c4,xi);

n=5;
c5= polyfit(x,y,n);
yi5=polyval(c5,xi);
```

```
n=6;
c6= polyfit(x,y,n);
yi6=polyval(c6,xi);
%1
figure
plot(x,y,'k*')
hold on

plot(xi,yi2,'b')
plot(xi,yi3,'k')
plot(xi,yi4,'g')
%plot(xi,yi5,'--m')
%plot(xi,yi6,'-.r')
ylabel('distanza[cm]')
xlabel('valore␣sensore␣IR')
title('fitting␣distanza-␣valore␣IR')
axis([0 15 0 11])
legend('valori␣misurati','valori␣interpolati,
␣␣␣␣␣␣␣␣grado␣2', 'valori␣interpolati,␣grado␣3',
        'valori␣interpolati,␣grado␣4')
hold off


%curva scelta
cs=[0.0008,   -0.033,    0.482,   -3.168,   10.845]
yis=polyval(cs,xi); %valutazione del polinomio in xi

%2
figure
plot(x,y,'k*')
hold on
plot(xi, yis,'--m')
plot(xi, yi4,'g')
plot(xi,yp_a,'b')
ylabel('distanza[cm]')
xlabel('valore␣sensore␣IR')
title('fitting␣distanza-␣valore␣IR')
axis([0 15 0 11])
legend('valori␣misurati', 'curva␣scelta␣approssimata',
        'curva␣scelta␣calcolata')
hold off
```

Appendice F

Khepera3

Questo appendice vuole essere un piccolo manuale per l'uso del robot Khepra III, fermo restando che l'intera tesi può essere considerata tale.

F.1 Descrizione Khepera III

Il *Khepera III* è un robot con due ruote con relativi motori indipendenti, 11 sensori attivi ad infrarosso, 5 sensori attivi ad ultrasuoni, interfaccia USB, interfaccia seriale, antenna per il collegamento bluetooth[2]. La scheda madre del robot permette di gestire tutto questo ma non permette l'esecuzione di programmi e algoritmi, per fare ciò si necessita di un processore. La *K-Team* fornisce un processore ARM con una scheda chiamata *KoreBotLE*[8] la quale permette di espandere le interfacce, con l'hardware opportuno è stato aggiunto un modulo WiFi e una CamUSB. Il materiale in possesso del laboratorio di automatica è indicato in tabella F.1.

In questa sezione verrà trattato il metodo di interfacciarsi col robot, lasciando al capitolo successivo la descrizione fisica dei dispositivi utilizzati. Prima di tutto saranno trattati i tipi di collegamento, poi sarà illustrato nel dettaglio l'utilizzo del *KoreBotLE*[1] per interfacciarsi al robot e fargli eseguire dei programmi. I tipi di connessione sono:

- connessione tramite connettore seriale;
- connessione tramite connettore USB;
- comunicazione tramite protocollo bluetooth;
- collegamento Wireless Ethernet direttamente col *KoreBotLE*.

[1] *www.korebot.com*

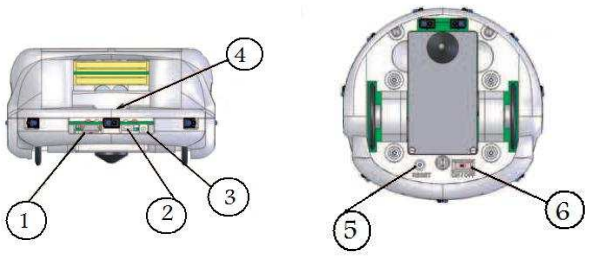

Figura F.1: Vista connettori e lato inferiore Khepera III

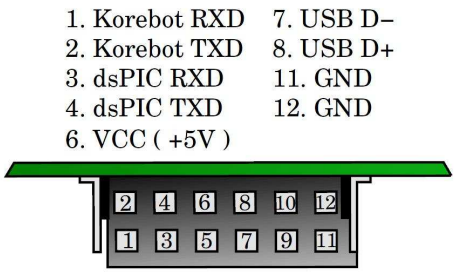

Figura F.2: dettaglio del connettore seriale

Connessione seriale Nelle figure F.1 e sono indicati:

1. connettore seriale principale;
2. USB adattatore miniAB ;
3. connettore jack per la carica della batteria;
4. cable unroller connector (connettore seriale e di alimentazione, ma non di carica della batteria);
5. pulsante di reset;
6. pulsante di accensione (ON/OFF).

Il connettore seriale è a 12 pin, le cui connessioni sono indicate in figura F.2. Tramite *KoreConnect* sono possibili due tipi di connessioni(si veda figura F.3) : la prima direttamente col *KoreBotLE*, la seconda col *dsPIC del Khepera III*.

Quando il *KoreBotLE* non è collegato è possibile impartire i comandi direttamente al robot utilizzando la linea della porta seriale dedicata alle

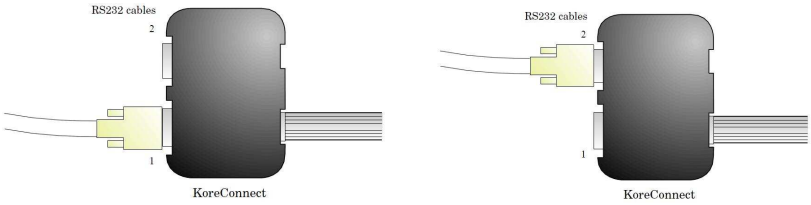

Figura F.3: Connessione seriale, con e senza KoreBot

comunicazioni a livello di TTL[2] (0V+5V). Il computer sarà collegato al robot mediante il cavo seriale e il *KoreConnect* convertirà il segnale da RS232 al livello TTL per comunicare col robot. Per eseguire questo tipo di connessione è necessario accertarsi che:

- il *KoreBotLE* non sia connesso nel robot;
- la batteria sia carica;
- il robot sia connesso al *KoreConnect* mediante il cavo seriale;
- il computer sia connesso tramite la porta seriale al *KoreConnect*;
- la porta seriale sia configurata nel seguente modo: 115200bps, 8 Data bits, 1 stop bit, no parity, no hardware control.

Sarà così possibile impartire al robot i comandi in base al protocollo di comunicazione illustrato a pagina 33 del manuale[7].

Quando invece il *KoreBotLE* è collegato è necessario eseguire i collegamenti come nella figura F.3 a sinistra. In questa configurazione non è possibile comunicare direttamente col robot e impartire i comandi, ma sarà necessario utilizzare dei programmi specifici da far girare sul processore ARM presente nel *KoreBotLE*. Il collegamento richiede gli stessi requisiti illustrati per la connessione precedente, con le seguenti distinzioni:

- è necessario accertarsi che il *KoreBotLE* sia collegato;
- il cavo seriale deve essere collegato come illustrato nella figura F.3 a sinistra.

Connessione USB La connessione USB ha lo scopo di metterci in collegamento col *KoreBotLE*, quindi non vi è possibilità di impartire i comandi direttamente al robot.

Ci sono due modi per collegare il computer tramite cavo USB:

[2]TTL=(Transistor-Transistor-Logic)

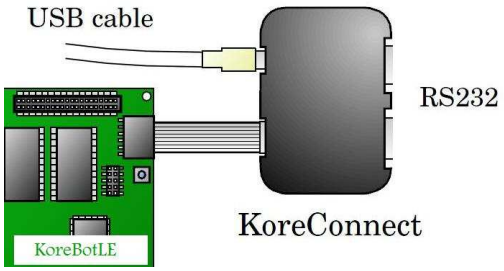

Figura F.4: Connessione tramite KoreConnect e cavo USB

- tramite il *KoreConnect*, si veda figura F.4;
- tramite la mini USB indicata in figura F.1;

Per eseguire questo tipo di connessione è necessario accertarsi che:

- *KoreBotLe* sia collegato nel *Khepera III*;
- la batteria sia carica, e il robot sia acceso;
- il robot sia connesso tramite il cavo USB in uno dei modi precedentemente illustrati;
- il computer sia connesso tramite il cavo USB al *KoreConnect*.

Connessione Blootooth La connessione tramite protocollo Bluetooth ci permette di interfacciarci direttamente col robot o tramite il *KoreBotLe*. La configurazione della porta deve rispettare i seguenti parametri:

- 115200bps data rate;
- 8 data bits;
- no parity,
- one stop bit;
- no hardware flow control;
- codice di sicurezza per la connessione: 0000.

Una volta eseguito questa configurazione basterà accendere l'antenna bluetooth del nostro computer ed eseguire lo scanning dei dispositivi nelle vicinanze. I *Khepera* compariranno col nome *KHIII*xxxxx, dove xxxxx indica il

numero di serie specifico per ogni robot. Ora basterà creare una connessione con il robot e digitare il codice di sicurezza e saremo collegati al robot, tramite o meno il *KoreBotLE*.

Collegamento WiFi Questo tipo di collegamento è realizzabile solo tramite l'uso del *KoreBotLE* che gestisce la connessione Wireless Ethernet. È necessario montare sul robot:

- il *KoreBotLE*;
- la scheda Wireless, la quale va montata direttamente sul *KoreBotLE*.

La scheda wireless supporta il protocollo LAN 802.11b/g ed è prodotta dalla *Abicom Technology*. Per connettersi è necessario stabilire una connessione *ad hoc* tra il nostro computer e il processore ARM, oppure usare un access point e creare una connessione *managed*. Per spiegare come questo sia possibile è necessario prima illustrare il funzionamento del robot tramite *KoreBotLE*.

Sul computer Sul computer di lavoro è consigliabile, ma non obbligatorio, installare una distribuzione di Linux. Questo ci permetterà di usare il programma *minicom*, sotto Linux, per i collegamenti via cavo seriale, USB, e bluetooth. È possibile effettuare le connessioni anche con un istanza dell'hiperterminal di windows ma si potrebbero verificare alcuni problemi di compatibilità di codifica dei caratteri. *Minicom* è un programma di connessione tra computer che utilizza vari protocolli di comunicazione, ad esempio il nostro robot richiede il protocollo *zmodem* per l'invio e la ricezione dei file. É un programma che si avvia da terminale. Settando la porta corretta è possibile stabilire connessioni di varia natura, dalla USB alla seriale. I tipi di porta da settare sul modem minicom sono:

- $/dev/ttyS0$, per il collegamento seriale;
- $/dev/ttyUSB0$, per il collegamento via USB;
- $/dev/rfcomm0$, per collegamento bluetooth;

considerando la porta 0, ma a seconda del pc il valore può cambiare.

Per la connessione WiFi si veda la sezione successiva.

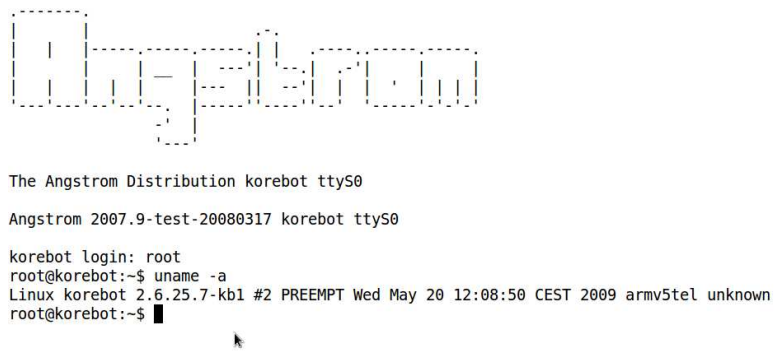

Figura F.5: Avvio del istema operativo embedded

F.2 Descrizione KoreBotLE

Il *KoreBotLE*[8] monta un chip dell'*intel* XSCALE PXA255 400Mhz, una memoria RAM da 64 MBytes e una memoria FLASH da 32 MBytes. L'architettura XSCALE si basa su set di istruzioni ARM v5TE. Su tale processore è installato un sistema operativo che consta di due parti: il kernel del sistema operativo embedded Linux e un set di pacchetti software chiamati distribuzione. La distribuzione di Linux attualmente installata sul *KoreBotLE* è Angstrom (*Angstrom 2007.9*), versione del kernel *linux korebot 2.6.25.7 kb1.*

All'accensione del robot si avvierà Angstrom, al termine dell'avvio è richiesto l'identificativo (ID) **root** per accedere. Dopo ciò è possibile interagire col sistema operativo, come ad esempio leggere e modificare file o avviare la connessione WLAN. In figura F.5 è possibile vedere la schermata d'avvio del *KoreBotLE.*

Con i comandi **iwconfig**, **ifconfig** e **ifup** è possibile settare i parametri della nostra connessione Wireless e avviare la rete, ricordando che il dispositivo per la connessione wireless è l'*eth0* e non *wlan0* come nelle distribuzioni per personal computer. Per la sequenza esatta del codice si veda l'appendice G. Per poter aver un istanza del terminale del robot sul nostro computer e quindi poter vedere l'output generato dall'esecuzione di qualsiasi programma avviato su di esso è necessario digitare da terminale del computer il comando:

```
ssh root@ipAddress
```

Invece, per inviare file al robot è necessario digitare sempre da shell:

```
scp nomefile.est root@ipAddress:/home/root/
```

Vediamo ora come possiamo scrivere i programmi e gli algoritmi da eseguire sul robot. La *K-Team* fornisce una libreria API per sviluppi di appli-

cazioni sul robot che permette di operare ad alto livello su tutti i dispositivi. La **LibKorebot API** è scritta in linguaggio C ed è rilasciata sotto licenza pubblica GPL. Sfruttando le funzioni di libreria è possibile scrivere un programma C per l'esecuzione di qualsiasi operazione, dal movimento delle ruote alla lettura dei sensori. La struttura del programma deve rispettare alcune caratteristiche:

- richiamo librerie principali, ad esempio *korebot.h*;
- inizializzazione dei dispositivi necessari in fase di esecuzione;
- implementazione di un sistema ibrido;
- necessità di stabilire un tempo di campionamento per l'esecuzione delle operazioni e aggiornamento dei valori.

Una volta scritto il nostro programma, dovrà essere compilato dal cross-compilatore e inviato al robot. Il cross-compilatore è la *ARM toolchain* versione lightV0.1_kernel2.6.

F.3 Inventario Autolab

In tabella F.1 è riportato l'elenco del materiale di cui il laboratorio di automatica è in possesso.

Tabella F.1: Inventario AutoLAb

quantitá	nome	descrizione
5	Khepera III	Khe3B-0012xxx, Khe3T-0012xxx robot più manuale e valigetta
5	CD kheperaIII	software di interfaccia col robot, varia documentazione elettronica
5	coperchietti metallici	
5	KorebotLE	KorLE-0014xxx ARM processor
5	KoreUSBCAM	KoreCam-0xxx
5	KoreWiFi	wireless network IEEE 802.11 b/g
2	KoreConnect	scatola nera di connessione del robot tramite cavo seriale
3	cavo	cavo di connessione tra coreconnect e robot
5	scheda KioLe-00111xx	scheda di estensione I/O
5	cavetto	cavetto giallo, rosso e nero per connessioni con il KioLe
1	caricabatteria autonomo	ricarica di tre batterie contemporaneamente senza robot
5	caricabatteria da robot	
5	adattatore spina italiana	adattatore per caricabatteria da robot
5	adattatore spina americana	adattatore per caricabatteria da robot, spina americana a due poli
10	batteria	
1	chiavetta bluetooth	sitecom CN-516 chiave bluetooth per pc con cd di installazione
1	WebCam LAN	

Appendice G

Rudimenti di Linux

G.1 Alcune definizioni

GNU/Linux è un sistema operativo open-source della famigli Unix (o unix-like) costituito dall'integrazione del kernel Linux con elementi del sistema GNU e di altro software sviluppato e distribuito con licenza GNU GPL o con altre licenze libere.

Kernel Il Kernel è la parte principale del sistema operativo. Si tratta di un software avente il compito di fornire ai processi in esecuzione sull'elaboratore un accesso sicuro e controllato all'hardware. Dato che possono esserne eseguiti simultaneamente più di uno, il kernel ha anche la responsabilità di assegnare una porzione di tempo-macchina e di accesso all'hardware a ciascun programma (multitasking).

Il Kernel viene caricato in memoria subito dopo il BIOS e si occupa del trasferimento dei dati fra le componenti del sistema (disco fisso, RAM, CPU, schede, interfacce...) e della gestione della CPU. Riceve ed inoltra i comandi dell'utente tramite la shell.

Cross-compilatore Un cross-compilatore è un compilatore che produce codice eseguibile destinato all'esecuzione su una piattaforma hardware/software diversa da quella in cui opera il compilatore stesso. Viene utilizzato soprattutto per creare programmi eseguibili per piattaforme dotate di risorse troppo limitate per consentire l'esecuzione in loco di un compilatore, come determinati microcontrollori o altri dispositivi di calcolo di piccole dimensioni.

Make `make` è una utility usata con i sistemi operativi della famiglia UNIX che automatizza il processo di conversione dei file da una forma ad un'altra, risolvendo le dipendenze e invocando programmi esterni per il lavoro necessario. Molto frequentemente è usato per la compilazione di codice sorgente in codice oggetto, unendo e poi linkando il codice oggetto in eseguibili o in librerie. Esso usa file chiamati *makefiles* per determinare il grafo delle dipendenze per un particolare output, e gli script necessari per la compilazione da passare alla Shell.

Makefile Il *makefile* è il file che contiene le istruzioni sulle operazioni da eseguire all'avvio del comando `make`. Può essere personalizzato a piacere a seconda delle operazioni da eseguire. Contiene le istruzioni per lo specifico progetto e ha come nome di default il nome stesso *makefile*.

Terminale Il *terminale* o più comunemente chiamato anche *Shell* è un programma attraverso cui è possibile impartire dei comandi al sistema operativo. Nell'accezione comune del termine, quando si parla di terminale si parla di interfaccia tramite riga di comando, seppur esistano anche Shell grafiche chiamate in questo caso *desktop environment*.

Esistono vari tipi di shell, in particolare cito la shell *sh* che è quella più utilizzata nelle distribuzioni di Linux per sistemi embedded. La shell *sh* è la shell Bourne. Ogni piattaforma Unix dispone della shell Bourne o di una shell Bourne compatibile.

Socket Il Socket (letteralmente presa) è punto di collegamento tra un client che richiede una risorsa ed un server che la fornisce. In Linux un Socket è un descrittore (tipo di file) che va inizializzato tramite la funzione *socket()*, la cui sintassi è `int socket(int domain, int type, int protocol )`; che restituisce un valore uguale o maggiore di zero in caso di successo, o minore di zero in caso di errore.

G.2 Principali comandi

Riporto qui i comandi i comandi più comuni utilizzati in una shell per ambienti di tipo Unix.

- `cd nomedir` passa dalla directory corrente alla directory `nomedir`;
- `cd ..` passa alla cartella superiore;

- `ls` restituisce l'elenco dei file e cartelle memorizzati nella directory in cui viene digitato;
- `ifconfig` permette di impostare le varie caratteristiche dell'interfaccia di rete;
- `iwconfig` permette di impostare le varie caratteristiche dell'interfaccia di rete Wireless Fidelity (WiFi);
- `ifup nomeinterfaccia` avvia la rete impostata coi comandi precedenti per l'interfaccia `nomeinterfaccia`;
- `ssh nome@ind.ip`, la sigla sta per le parole `Secure SHell` e stabilisce una connessione sicura con il dispositivo Linux con ID `nome` e indirizzo Internet Protocol (IP) `ind.ip`.
- `scp nomefile nome@ind.ip:/home/root/` invia il file *nomefile* al dispositivo indicato.

Se vogliamo avviare una connessione WiFi possiamo digitare i comandi:

```
iwconfig eth0 essid nomeRete
ifup eth0
ifconfig eth0 ipaddress
```

Appendice H

Acronimi

CVM Curvature Velocity Method

DVZ Deformable Virtual Zone

ID Identificativo

IP Internet Protocol

IR Infrared Sensor

IU Ingresso-Uscita

MIMO Multiple-Input Multiple-Output

NCS Networked Control System

SISO Single-Input Single-Output

USB Universal Serial Bus

VFH Vector Field Histogram

VS Variabili di Stato.

WiFi Wireless Fidelity

WLAN Wireless Local Area Network