Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria dell'Automazione
Dipartimento di Elettronica, Informazione e Bioingegneria

CONTROLLO PREDITTIVO DISTRIBUITO PER LA NAVIGAZIONE E IL SENSING DI SISTEMI MULTI AGENTE

Relatore: Prof. Riccardo SCATTOLINI
Correlatore: Prof. Marcello FARINA

Tesi di laurea di:
Marco VERGANI Matr. 803835

Anno Accademico 2013–2014

Ai miei genitori
A Pierangela, Claudio e Matteo

Sommario

Grazie al progresso tecnologico degli ultimi anni nei campi della robotica e dei controlli automatici è stato possibile sviluppare sistemi formati da flotte cooperanti di agenti mobili autonomi atti a svolgere azioni di coordinamento, ispezione, pattugliamento anche in ambienti ostili o non facilmente accessibili. Per la gestione di questi sistemi è indispensabile poter disporre di tecniche di controllo di tipo distribuito in grado di garantire prestazioni robuste e il più possibile performanti, al fine di raggiungere un determinato obiettivo comune.

Lo scopo della tesi è quello di fornire una struttura logica e di controllo in grado di soddisfare le richieste appena citate per la soluzione di differenti problemi e applicazioni inerenti alla gestione di agenti mobili.

A tale proposito nel Capitolo 1, verranno presentati l'apparato sperimentale costituito da una piccola flotta di *robot* autonomi a uniciclo e le tecniche utilizzate per l'acquisizione della posizione degli agenti. Saranno poi illustrate, nel Capitolo 2, le strategie atte alla pianificazione e all'inseguimento della traiettoria per *robot* mobili. Successivamente, nel Capitolo 3, si mostrerà un'adeguata architettura logica che, cooperando opportunamente con una struttura di controllo predittivo e distribuito, presentata nel Capitolo 4, permetterà di raggiungere gli obiettivi prefissati. Particolare attenzione sarà poi dedicata alla presentazione degli algoritmi che permettono la soluzione dei problemi di *obstacle* e *collision avoidance* sia attraverso l'utilizzo di opportuni vincoli introdotti nel problema di ottimizzazione, come mostrato nel Capitolo 5, sia tramite leggi algebriche basate sull'ausilio di potenziali artificiali, come illustrato nel Capitolo 6. Verrà inoltre analizzato, nel Capitolo 7, il problema di *coverage* per il rilevamento di una determinata area. Durante la trattazione saranno descritte tutte le varie leggi di controllo che permettono la risoluzione dei problemi affrontati, focalizzandosi principalmente sulla necessità di calcolare le azioni di controllo nei limitati tempi computazionali disponibili, rispettando così un vincolo tecnologico stringente nella gestione dei *robot*. Tutti gli algoritmi di controllo proposti nei vari capitoli verranno validati e testati su banco prova tramite l'ausilio di tre agenti mobili. Nel Capitolo 8 verranno poi riassunti i risultati ottenuti e si presenteranno dei possibili sviluppi futuri. Infine, nell'Appendice A sarà riportato e descritto dettagliatamente il *firmware* caricato sul microcontrollore posto all'interno di ogni agente.

Abstract

In recent years, thanks to technological advances in the fields of robotics and automatic control, researchers have developed systems of cooperating fleets of mobile agents designed to carry out coordination, inspection, patrol operation, in hostile or not easily accessible environments. For the management of these systems it is essential to use distributed control techniques ensuring robustness and high performances in order to reach a given common goal.

The purpose of this thesis is to provide a logical and control structure that is able to meet the requirement mentioned above, also for different applications and problems related to the management of mobile agents.

To this regard, in Chapter 1, it will be presented the experimental apparatus composed by a small fleet of mobile agents and the techniques used for the sensinsg and estimation of the state of the robots. In Chapter 2, the strategies for the planning and path tracking for mobile agents will be discussed. Then, in Chapter 3, it will be demostrated how an adequate logical architecture, cooperating with a predictive and distributed control structure, presented in Chapter 4, allows to reach the desired goals. Particular attention will be paid to algorithms that allow the management of obstacle and collision avoidance, either through the use of appropriate constraints introduced in a suitable optimization problem, Chapter 5, or by algebraic laws based on the definition of artificial potential fields, Chapter 6. In Chapter 7, the problem of coverage for the measure of a specific area will also be tackled. In addition, the various control laws that allow the resolution of the addressed problems will be described, focusing mainly on the need to compute the control actions in the limited computational time available, thereby fulfilling a hard technological constraint in the handling of the robot. All the proposed control algorithms will then be validated and tested on a benchmark consisting of a set of three real mobile agents. In Chapter 8 the results obtained will be summarized and possible future developments will be presented. Finally, in the Appendix A the firmware loaded on the microcontroller within each agent will be described.

Indice

Elenco delle figure

Capitolo 1

Apparato sperimentale

In questo capitolo si descrive il modello del sistema utilizzato durante tutto il lavoro, presentando la sua architettura complessiva. Nello svolgimento della tesi si sono utilizzati tre *robot* liberi di muoversi in un piano di lavoro e costantemente osservati da una *webcam* utilizzata per determinare la loro posizione e il loro orientamento; questo è stato possibile in quanto, essendo il banco prova comparabile alle dimensioni di una scrivania, è concepibile utilizzare telecamere di livello commerciale che hanno costi veramente limitati. Illustriamo ora nel dettaglio il *robot*, la *webcam* e i *software* utilizzati nella trattazione.

1.1 Il Robot

I *robot* che sono stati impiegati per il lavoro sono detti a uniciclo, in quanto presentano un unico asse su cui sono montate le due ruote; essi sono largamente diffusi nell'ambito di problemi di controllo di sistemi di agenti multipli poiché, grazie al loro modello semplice, ma allo stesso tempo anolonomo e caratterizzato da dinamiche non lineari, costituiscono un ottimo strumento di stimolo per lo studio e per l'analisi delle tecniche d'interesse. Si è scelto così di utilizzare dei *robot e-puck* sviluppati presso l'*Ecole Polytechnique Fèdèrale de Lausanne* [7], mostrato in Figura 1.1, i

Figura 1.1: *Robot e-puck*

quali presentano una meccanica di facile comprensione e una struttura elettronica

moderna, ma allo stesso tempo intuitiva. Per quanto riguarda il processore, viene utilizzato un *dsPIC* della famiglia *Microchip* che garantisce una buona potenza di calcolo, la *CPU* lavora a $64\,MHz$, ed è dotato di una discreta memoria, $8\,KB$ di *RAM* e $144\,KB$ di memoria *flash*. In aggiunta, questo microcontrollore presenta una struttura completa, dispone infatti di molti registri, interfacce e *set* di istruzioni, consentendo così un'elaborazione di dati molto efficiente. Un'altra caratteristica di questi agenti è la flessibilità, essi infatti si prestano come banco di prova per un ampio spettro di problemi, relativi ad esempio alla robotica mobile, alla programmazione in tempo reale, ai sistemi *embedded*, all'elaborazione di segnali, di immagini o di estrazione di funzioni d'audio, per non dimenticare l'interazione uomo-macchina o il controllo di sistemi collettivi. La gestione di tutte queste attività è possibile grazie all'elevato e diverso numero di sensori di cui sono forniti, che consentono una grande capacità di adattamento e interazione con il mondo esterno. Essi inoltre si prestano a un facile utilizzo, infatti, essendo di ridotte dimensioni, possono essere impiegati e testati su un piccolo banco prova accanto al *computer* senza dover essere collegati a nessuna alimentazione, in quanto presentano un funzionamento a batterie ricaricabili che garantisce ottimale *comfort* lavorativo. Il motivo predominante che ha portato però alla scelta di questo *robot* è stato che, rispetto ai suoi concorrenti, presenta costi moderati, robustezza e buone capacità prestazionali, ovviamente molto limitate rispetto ai *robot* industriali, ma comunque più che sufficienti per effettuare *test* sui sistemi di controllo presi in esame, come dimostrato in [1].

I *robot* presentano un apparato di movimentazione gestito tramite due motori passo-passo in miniatura con riduzione che consentono un perfetto controllo indipendente di ogni singola ruota; il motore presenta 20 passi per giro e l'ingranaggio ha una riduzione di $50 : 1$, mentre la velocità massima delle ruote è di circa 1000 *passi/s*, che corrispondono a un giro della ruota al secondo.

Vediamo ora una veloce panoramica delle caratteristiche principali dell'*e-puck*:

- alimentazione singola: tutti i circuiti di elettronica vengono alimentati a $3.3\,V$, tranne la fotocamera che ha bisogno di ulteriori $1.8\,V$;
- un connettore di estensione permette di aggiungere nuove funzionalità su una scheda di espansione;
- il *robot* è dotato di un *RS232* e di un'interfaccia *bluetooth* per comunicare con un *computer host*;
- un selettore rotante a 16 posizioni permette un *input* da parte dell'utente, è possibile quindi impostare differenti modalità di funzionamento su un unico agente;
- pulsante e connettore di programmazione per ripristinare le impostazioni di fabbrica;
- la batteria si basa sulla tecnologia *LiION*, $5\,Wh$ e presenta una durata di circa $2 - 3$ ore di uso intensivo. Essa può essere rimossa e ricaricata esternamente tramite apposito caricatore;
- 8 *led* rossi e 2 *led* verdi sono disposti intorno al corpo del *robot* e possono essere utilizzati per eventuali segnalazioni luminose.

Il *robot* dispone inoltre di diversi sensori che gli consentono di interagire in modo ottimale con l'ambiente esterno:

- 8 sensori di prossimità IR che coprono interamente il suo perimetro; questi sensori sono composti da due parti: un emettitore e un ricevitore. Il primo emette un raggio infrarosso, che viene riflesso dagli oggetti circostanti, parte del segnale torna pertanto verso il sensore e sulla base delle sue caratteristiche viene fornita una misura proporzionale alla distanza dell'oggetto colpito. Nel caso del *robot e-puck* il valore letto corrisponde alla differenza tra il livello di infrarossi nell'ambiente e quello ottenuto dal segnale riflesso;
- un accelerometro a 3 assi è collocato al suo interno;
- tre microfoni con frequenza massima di campionamento $33\,KHz$ e un altoparlante consentono l'acquisizione e generazione del suono;
- una fotocamera con una risoluzione di 640 x 480 *pixel* a colori è collocata all'interno del corpo dell'agente.

Un ulteriore vantaggio dei *robot e-puck* è inoltre quello che il *dsPIC*, presenta due *UART*[1]; la prima è collegata a un *chip bluetooth* e la seconda è fisicamente disponibile tramite un connettore *Micromatch*. Il *chip bluetooth*, *LMX9820A*, può essere utilizzato per entrare nella *UART* "trasparente", per mezzo di un canale *bluetooth rfcomm*; utilizzando questa modalità si può accedere all' *e-puck* come se fosse collegato a una normale porta seriale, con l'eccezione che il *LMX9820A* invierà comandi speciali al momento della connessione o disconnessione. Grazie quindi al dispositivo *bluetooth* è consentita una facile programmazione del *PIC* e un facile scambio di dati tra i *robot* e il loro dispositivo di controllo; essi possono essere infatti controllati da un qualsiasi sistema dotato anch'esso di modulo *bluetooth*. Le funzionalità del *robot* sono riassunte nella Figura 1.2.

Passiamo ora a studiare nel dettaglio la cinematica del *robot*; esso è costituito da una struttura circolare alla quale sono connesse due ruote coassiali indipendenti sull'asse diametrale, mentre il terzo punto d'appoggio è realizzato semplicemente mediante la struttura stessa. Le dimensioni sono di $52\,mm$ per l'asse tra le ruote e di $21\,mm$ per il diametro delle ruote stesse. Grazie a questa struttura la massima velocità di spostamento in linea retta dell'agente corrisponde alla massima velocità di rotazione che i suoi due motori passo passo possono fornire alle ruote. Il *robot* può quindi essere rappresentato come in Figura 1.3 e in particolare si può supporre che il suo punto d'appoggio sia neutrale ai fini del moto e pertanto ininfluente sulla dinamica finale; di conseguenza è possibile modellare il *robot* come se fosse costituito dalle sole due ruote attuate. Per questa ragione, la struttura in questione può essere descritta matematicamente secondo un modello, molto diffuso in letteratura, denominato modello a uniciclo, caratterizzato da un vincolo non olonomo che limita la velocità del *robot* a rimanere ortogonale all'asse che collega le due ruote, come rappresentato in Figura 1.4.

Inoltre teniamo presente che, sebbene nel nostro caso le variabili d'ingresso a disposizione siano le due velocità di rotazione applicate alle ruote, ovvero ω_R e ω_L,

[1] *Universal Asynchronous Receiver-Transmitter* (ricevitore-trasmettitore asincrono universale) è un dispositivo *hardware* di uso generale o dedicato che converte flussi di *bit* di dati da un formato parallelo a un formato seriale asincrono o viceversa; ogni famiglia di microprocessori ha la sua *UART* dedicata.

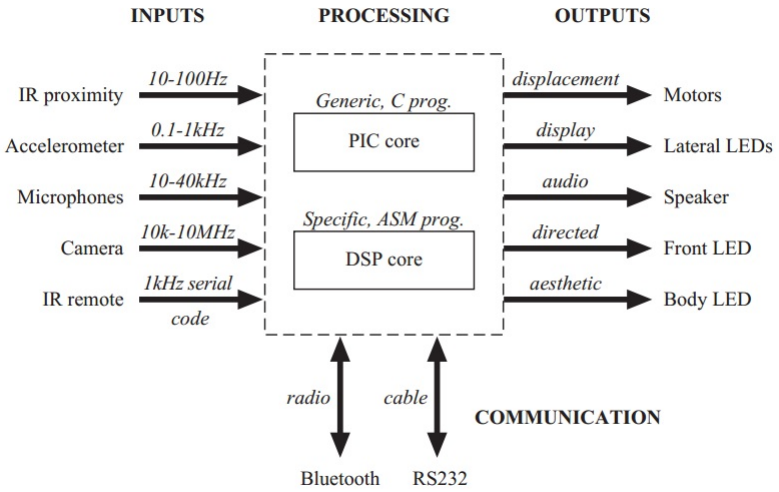


Figura 1.2: *Hardware e-puck*

è possibile ricavare facilmente la velocità lineare che si ha al centro del *robot* e la velocità angolare dell'intero corpo, ovvero v e ω. Nello specifico, conoscendo per il *robot e-puck* il raggio delle ruote $R = 1.05\,cm$ e la lunghezza del loro asse pari a $2E = 5.2\,cm$ si ricavano le seguenti relazioni geometriche:

$$\begin{cases} v_R = & \omega_R R \\ v_L = & \omega_L R \end{cases} \qquad \begin{cases} v = & \frac{v_R + v_L}{2} \\ \omega = & \frac{v_R - v_L}{E} \end{cases}$$

Utilizzando queste due nuove coordinate il modello usato per questa tipologia di *robot* risulta semplice da ricavare:

$$\begin{cases} \dot{x} = & v\cos\theta \\ \dot{y} = & v\sin\theta \\ \dot{\theta} = & \omega \end{cases} \tag{1.1}$$

con un vincolo non olonomo corrisponde alla relazione:

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0$$

che si può ricavare da (1.1).

Da notare che nel caso considerato il *robot* è vincolato ad avanzare in un'unica direzione perpendicolare all'asse delle ruote, mentre la rotazione viene effettuata andando a settare in maniera differenziale la velocità delle due ruote stesse, che ricordiamo possono essere gestite in maniera indipendente.

Questo modello presenta tuttavia alcune approssimazioni che potrebbero causare una scarsa accuratezza. La prima approssimazione è dovuta al fatto che le variabili d'ingresso considerate sono la velocità lineare v e la velocità angolare ω, mentre in realtà i veri ingressi sono le coppie motrici effettivamente applicate alle due ruote dei motori; questo potrebbe costituire un problema che analizzeremo nel prosieguo della trattazione. Un'ulteriore approssimazione è invece data dal fatto che con questo modello si tende a trascurare la dinamica delle velocità e l'inerzia del *robot* durante il moto, ma, avendo a che fare con agenti che presentano velocità limitate, questa approssimazione può risultare essere accettabile.

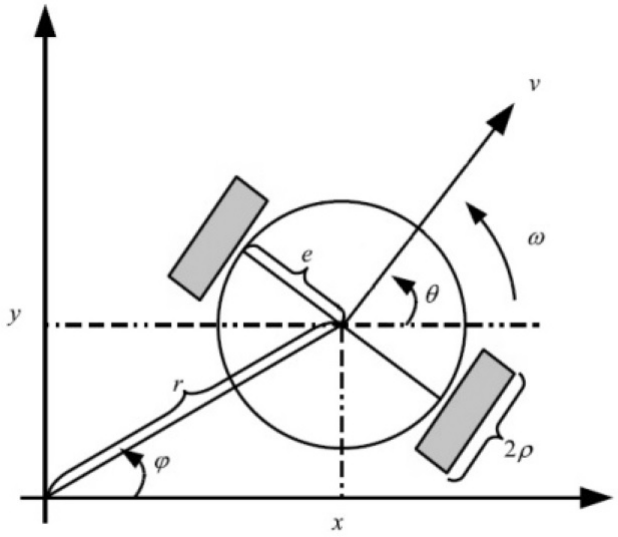


Figura 1.3: Modello cinematico del *robot*

1.2 Telecamera e analisi dell'immagine

Gli organi di visione artificiale sono sensori utili per la robotica poiché, imitando il senso umano della vista, consentono misure dell'ambiente senza contatto; al giorno d'oggi esistono diversi controllori robotici che integrano sistemi di visione. Una telecamera è un dispositivo in grado di misurare l'intensità della luce, concentrata da una lente su un piano, contenente una matrice di *pixel* che trasforma l'energia luminosa in energia elettrica. Qualunque sia la tecnologia utilizzata per convertire la luce in un segnale elettrico, la telecamera dovrà essere in grado di gestire il protocollo *standard* video che consente di specificare i vari parametri di sistema, tra cui la temporizzazione associata alla scansione dell'immagine; per una più approfondita descrizione si rimanda a [15]. La telecamera effettua quindi una proiezione 2D della scena inquadrata andando così a riportare i punti d'interesse dello spazio sulla medesima superficie che prende il nome di piano immagine, come mostrato nella Figura 1.5.

Un punto P di coordinate (X', Y', Z') nel sistema di riferimento della telecamera viene proiettato nel corrispondente punto del piano immagine π secondo la relazione:

$$\xi = \begin{bmatrix} x_\pi \\ y_\pi \end{bmatrix} = -\alpha \frac{\lambda}{Z'} \begin{bmatrix} X' \\ Y' \end{bmatrix}$$

dove λ rappresenta la distanza focale mentre α è il fattore di scala in *pixel/m*.

Durante la fase di scelta dell' *hardware*, si è riflettuto circa le possibili tecnologie a disposizione per la misura della posizione e dell'orientamento di ciascun *robot*. Il principale requisito è la possibilità di poter distinguere l'identità di ogni agente all'interno dell'area di lavoro e di avere una frequenza di aggiornamento delle informazioni sufficientemente elevata da permetterne la tempestiva trasmissione e aggiornamento all'interno dell'algoritmo implementato. Considerando il fatto che il piano di lavoro ricopre un'area limitata si può pensare di fare ricorso a un sistema di rilevamento visivo fisso posizionato al di sopra di esso (*eye to hand configuration*), in grado di scattare e acquisire un'immagine della zona interessata, che verrà impostata tramite

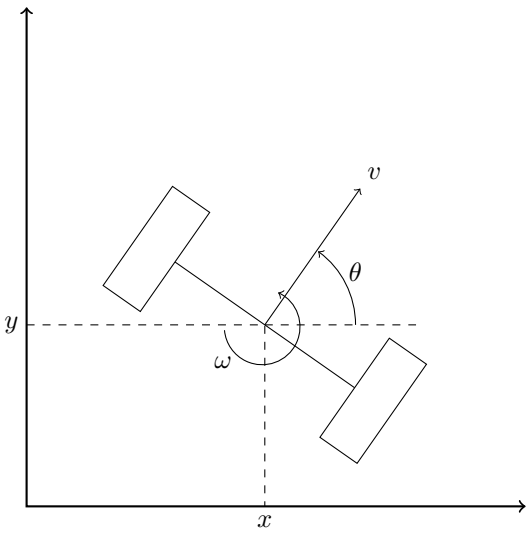


Figura 1.4: Modello uniciclo

vincoli posti sull'area di lavoro, distinguendo così all'interno di essa i vari agenti. I vantaggi della configurazione *eye to hand* sono dati dal fatto che il campo visivo è fisso e non cambia allo spostarsi degli agenti mobili, inoltre anche la relazione tra la telecamera e lo spazio di lavoro è fissata, quindi tale sistema di visione può facilmente essere calibrato *off-line.*

La soluzione enunciata presenta in aggiunta il vantaggio di utilizzare un'unica telecamera esterna senza avere la necessità di alcun componente attivo o sensore a bordo di ciascun *robot* rendendone così più pratico il loro utilizzo; in questo modo non è quindi necessario aggiungere ai microcontrollori un ulteriore carico di lavoro che si tradurrebbe in un incremento del dispendio energetico degli agenti. Inoltre l'utilizzo di opportune tecniche per la marcatura dei *robot*, può consentire di rilevare, oltre alla posizione, anche il corretto orientamento, il che è un notevole vantaggio per gestire problemi di inseguimento di una precisa traiettoria di riferimento. D'altra parte, se si considerano invece applicazioni reali in cui può essere effettivamente utilizzato il lavoro svolto in questa tesi, avendo a che fare con aree di lavoro solitamente più grandi di un banco prova, non sempre il posizionamento di una telecamera al di sopra del piano di lavoro potrebbe risultare possibile. Infatti in questi casi dovranno essere pensate e implementate nuove tecniche, tra cui per esempio quella che richiede di far ricorso a un sistema di triangolazione basato sulla misura, a bordo dei *robot*, di segnali provenienti da stazioni in posizione prefissata, o viceversa sulla misura effettuata dalle stazioni di segnali prodotti dai *robot* stessi. In questo caso sarebbe dunque necessario equipaggiare ogni agente con opportuni sistemi di trasmissione, sensori e algoritmi di rilevamento predisponendo in modo complementare nell'area di lavoro degli opportuni punti di ricezione o trasmissione, avendo così un conseguente incremento dei costi; per più approfondite descrizioni circa questo metodo si rimanda a [16, 17].

Tornando quindi al nostro problema specifico, che presenta un piano di lavoro con un'area limitata, si opta per utilizzare una *webcam Microsoft LifeCam HD-6000 VX-800* con una risoluzione di 640 x 364 px^2 che non presenta particolari requisiti; essa comunque deve essere tarata e tale calibrazione viene effettuata in due passi:

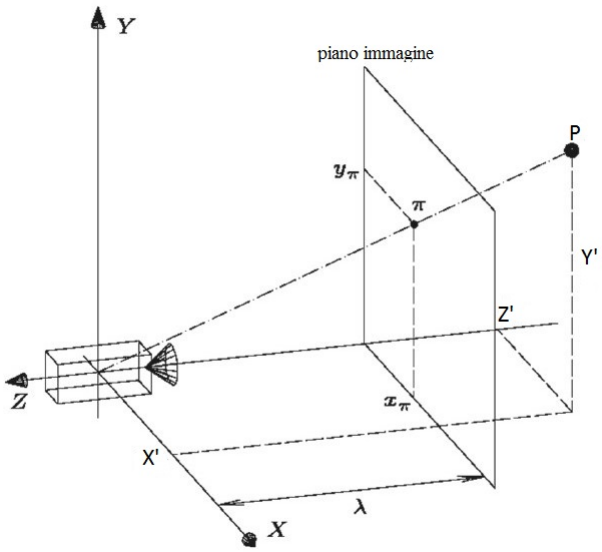


Figura 1.5: Tecnica di acquisizione della telecamera

- calibrazione interna, ovvero la determinazione dei parametri intrinseci della telecamera e di parametri addizionali di distorsione, dovuti a imperfezioni della lente e a disallineamenti del sistema ottico;

- calibrazione esterna, cioè la determinazione dei parametri estrinseci della telecamera, quali la posizione e l'orientamento della stessa rispetto al sistema di riferimento fisso sul piano di lavoro.

La *webcam* fungerà quindi da controllore servo visivo andando a fornire direttamente dati utili all'algoritmo circa la posizione degli agenti sulla superficie di lavoro; per questo motivo si parla di architettura di controllo *visual servoing* diretta.

Più in dettaglio, per gestire questo sistema di visione si è progettato un semplice algoritmo di analisi delle immagini in grado di ricavare i parametri desiderati, ovvero posizione e orientamento; andiamo ora a vedere minuziosamente come avviene l'acquisizione e l'analisi delle immagini. Per l'individuazione della posizione dei *robot* mobili l'algoritmo sviluppato ha lo scopo di discriminare i diversi agenti trovandone così le loro posizioni e i loro orientamenti. I *robot* sono equipaggiati con un triangolo isoscele colorato, i colori sono stati scelti sfruttando la proprietà additiva dei colori primari la quale afferma che tutti i colori presenti nello spettro del visibile possono essere rappresentati sommando le tre componenti dei colori primari *RGB* (*Red, Green, Blue).* Lo spazio *RGB* viene rappresentato attraverso un cubo, come mostrato in Figura 1.6, in cui lo spigolo in basso a sinistra rappresenta il nero, quello opposto il bianco e un generico colore sarà quindi raffigurato da un punto all'interno del cubo. L'occhio umano, in sostanza, percepisce la somma delle tre componenti appartenenti allo spazio *RGB* come un unico colore; per maggiori approfondimenti si rimanda a [15]. In questo modo si è riusciti a permettere che i colori non si confondessero tra loro, favorendo così l'individuazione di ogni singolo agente; successivamente, sfruttando le caratteristiche del triangolo isoscele, è stata possibile effettuare l'acquisizione dei vertici, riconoscendo inoltre quello frontale, e permettendo in questo modo di ricavare anche l'orientamento di ogni *robot* rispetto

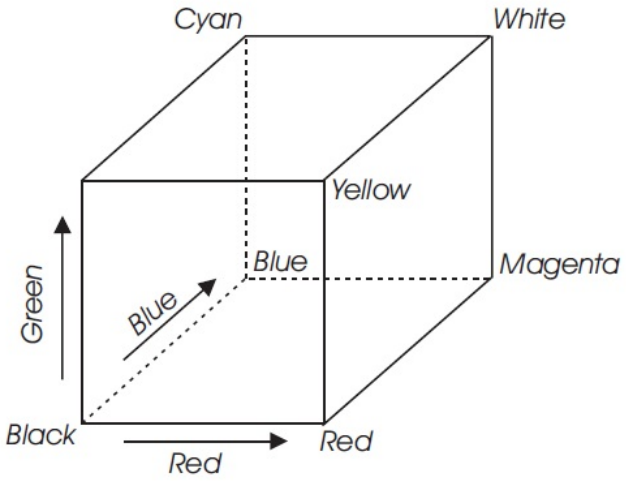


Figura 1.6: Spazio *RGB*

al sistema di riferimento posizionato sul piano di lavoro. Nello specifico l'algoritmo utilizzato compie i seguenti passi:

- determinazione dei vertici attraverso un opportuno algoritmo di *corner detection,* come visibile in Figura 1.7, il quale fornisce le coordinate corrispondenti di ciascun agente;

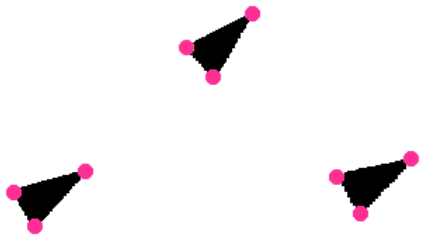


Figura 1.7: Determinazione dei vertici dei *robot*

- individuazione del vertice frontale attraverso il calcolo delle distanze fra i tre vertici rilevati; poiché il triangolo utilizzato è isoscele, il vertice frontale è quello la cui distanza dagli altri due è maggiore così come mostrato in Figura 1.8. Successivamente viene calcolata la posizione e l'orientamento attraverso semplici formule geometriche;

- conversione *pixel*-metri; le immagini acquisite dalla telecamera sono quantizzate in *pixel* ed è quindi necessario convertire le misure ottenute in metri. Il semplice approccio utilizzato consiste in una trasformazione lineare, il cui rapporto di proporzionalità è stato calcolato come rapporto fra le dimensioni dell'immagine in *pixel* e l'equivalente in metri. Questa soluzione ha il difetto di non considerare alcune distorsioni della videocamera presenti specialmente

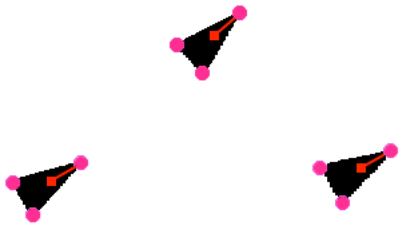


Figura 1.8: Individuazione del vertice frontale e calcolo del punto medio

ai bordi del campo visivo, ovvero nelle zone in cui ci si aspetta un maggiore errore di visualizzazione.

1.3 Software utilizzati

Per quanto riguarda la parte *software* sono disponibili in rete numerose librerie scritte in linguaggio C oppure in *Assembler* per le componenti più delicate, che consentono di gestire direttamente i dispositivi (come attuatoti e sensori dei *robot e-puck*) senza doversi preoccupare delle problematiche di basso livello. In particolare, queste sono state utilizzate nella scrittura di una *routine* in grado di predisporre ogni singolo *robot* a essere pilotato direttamente da *computer*, ricevendo e interpretando comandi attraverso un canale di comunicazione *wireless*.

In origine, come trattato in [13], il sistema era interamente gestito tramite l'ambiente di programmazione *MATLAB*; utilizzando una semplice procedura di sincronizzazione si operava direttamente sul microcontrollore andando a effettuare su esso l'operazione di comunicazione e la *routine* d'interpretazione e applicazione dei comandi ai motori, con una particolare attenzione alla saturazione degli ingressi di controllo. La scelta dell'ambiente *MATLAB* per la gestione lato calcolatore dell'intero progetto è stata fatta per consentire un più agevole ricorso alle tecniche moderne di ottimizzazione che saranno necessarie per la gestione e l'implementazione del problema *MPC* atto a generare una traiettoria di riferimento ottimale che il *robot* dovrà inseguire. In realtà per alcuni aspetti come la localizzazione tramite sistema di visione, di cui si discuterà nel seguito, oppure la comunicazione *wireless* su porta seriale, sarebbe maggiormente efficiente un'implementazione attraverso linguaggi di più basso livello, come ad esempio il *C* o le librerie grafiche *openCV*. Tuttavia, data la particolare attenzione che questo aspetto richiederebbe, e che esula dagli scopi della presente trattazione, si è scelto di lasciare tale tipo di approfondimento a eventuali sviluppi futuri.

Nell'attuale trattazione, a differenza da [13], per quanto riguarda il problema di programmazione si distinguono quindi due differenti parti:

- nella prima si è programmato il *PIC* del *robot* tramite l'ambiente di sviluppo *MPLAB* andando a compilare un codice, riportato nell'Appendice A, in grado di trasmettere e attuare due determinate velocità ai motori a fronte di un determinato segnale (carattere speciale). Successivamente, tramite un programma *Bootloader* che utilizza un canale *bluethoot,* lo si è andato a caricare

dal *computer* al microcontrollore posto all'interno del *robot*. Infine l'ambiente di sviluppo *MPLAB* mette a disposizione una modalità di *debug* che permette, connettendo opportunamente il *robot* al *PC* di verificare riga per riga il codice scritto avendo così un chiaro riscontro sulla sua correttezza e sulle sue funzionalità. Tuttavia va sottolineato che tale *debug* è risultato essere poco efficace in quanto doveva garantire anche l'iterazione tra più ambienti di sviluppo, fattore che il *debug* non permette di prendere in considerazione. Di notevole aiuto in questa fase si sono quindi rilevate le tecniche più elementari, quali le segnalazioni luminose o le stringhe di dati utilizzate in modo da far accendere o spegnere un determinato *LED* o far stampare a video un dato o una stringa di caratteri quando avviene correttamente una precisa operazione;

- la seconda parte si basa invece sulla scrittura di un algoritmo in ambiente *MATLAB* per il controllo del *robot*. Questo algoritmo dovrà comunicare, ricevere e inviare dati, direttamente con il *robot*, e tramite le altre librerie controllare tutti i suoi vari attuatori e sensori. Questa parte risulterà essere la più delicata in quanto è proprio tale algoritmo che consentirà di gestire il problema di ottimizzazione della scelta e dell'inseguimento della traiettoria di riferimento da parte dei vari agenti.

1.4 Set-up Sperimentale

Si analizza ora la logica di funzionamento del sistema, schematizzata nella Figura 1.9, da cui si può intuitivamente capire il funzionamento complessivo dell'apparato

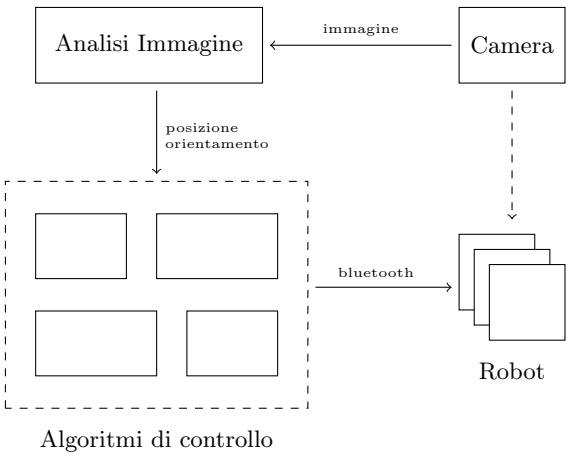


Figura 1.9: *Set-up* sperimentale

sperimentale che può essere scomposto in tre macro sezioni:

1. acquisizione immagine: la telecamera montata sopra il piano di lavoro acquisisce l'immagine;

2. analisi dell'immagine: sopra ciascun *robot* è posizionato un triangolo isoscele colorato in maniera differente, il che permette di isolare la posizione dei diversi agenti, mentre la forma triangolare è stata scelta per permettere di individuare l'orientamento θ dei *robot* come precedentemente spiegato;

3. trasmissione dell'azione di controllo: tramite la risoluzione di un determinato algoritmo di ottimizzazione viene calcolata l'azione di controllo, successivamente inviata tramite tecnologia *bluetooth* agli agenti, i quali la applicano ai motori passo passo in modo da ottenere *step-by-step* le velocità desiderate delle due ruote.

Capitolo 2

Pianificazione e inseguimento della traiettoria di robot mobili

Il *robot e-puck* utilizzato in tutta la sperimentazione presenta, come visto, due sole ruote a cui è possibile imporre indipendentemente la velocità di rotazione desiderata. Il problema di controllo di questa categoria di *robot* risulta essere in ogni caso possibile, ma non banale in quanto si dovranno tener presente di alcune limitazioni. Una di esse è data dal fatto che il sistema, nonostante risulti essere controllabile in *open loop*, non può essere stabilizzato mediante una tecnica di *feedback* continua e tempo invariante; non esiste infatti una legge di controllo algebrica in grado di stabilizzare completamente il sistema in ogni suo punto di equilibrio.

Abbiamo già visto che il sistema è modellabile come un uniciclo, caratterizzato da tre gradi di libertà, essendo appunto un corpo rigido che si muove nel piano, e due ingressi di controllo disponibili, che corrispondono alla velocità lineare v, diretta lungo un asse perpendicolare alle ruote, e alla velocità angolare ω. Tuttavia come risulterà chiaro nel prosieguo, attraverso ingressi opportuni, è possibile consentire al sistema di raggiungere qualsiasi posizione e orientamento desiderato; si può quindi affermare che il sistema risulta essere controllabile e questo richiederà leggi di controllo opportune, studiate in seguito.

Apriamo ora una parentesi circa il problema di pianificazione della traiettoria, che risulta essere un argomento di prima rilevanza all'interno di questa trattazione.

2.1 Pianificazione della traiettoria e tecniche di controllo

Per pianificazione della traiettoria si intende il problema di stabilire la modalità con cui si vuole che evolva il movimento dell'agente, da una posizione e orientamento iniziale a una posizione e orientamento finale. Si tratta di definire perciò sia il percorso geometrico che la legge di moto da realizzare, ossia la dipendenza temporale di posizioni, velocità e accelerazioni. La corretta pianificazione della traiettoria occupa quindi un ruolo di estrema importanza, perché comporta il fatto che essa possa essere eseguita da parte del sistema di controllo del moto in anello chiuso, senza incorrere nei limiti di saturazione degli attuatori o provocare dannose sollecitazioni meccaniche alla struttura. È inoltre importante effettuare la distinzione fra percorso, ovvero il luogo geometrico dei punti che l'agente deve descrivere, e traiettoria, con cui invece si indica il percorso su cui è stata specificata una legge oraria.

L'algoritmo di pianificazione della traiettoria richiede quindi di impostare preliminarmente i seguenti "ingressi":

- il percorso;
- i vincoli dati dal percorso;
- i vincoli dovuti alla dinamica del *robot*.

Inoltre avremo bisogno di definire anche delle "uscite", ovvero le traiettorie espresse come sequenza dei valori assunti da posizione, velocità, accelerazione e angolo di orientamento del *robot*. Solitamente però l'utente specifica solamente un numero ristretto di parametri quali:

- il percorso: punti estremi, eventuali punti intermedi, primitive geometriche;
- la legge di moto: tempo complessivo, velocità e/o accelerazioni massime, velocità e/o accelerazioni in determinati punti.

Tutte queste variabili saranno gestite grazie a uno schema di controllo distribuito basato su *MPC* detto *Distributed Predictive Control* (*DPC*) [4], per la generazione della traiettoria, che analizzeremo nel prosieguo e che a fronte di parametri impostati permetterà di risolvere il problema di pianificazione della traiettoria appena analizzato. Inoltre, inserendo nel codice caricato sui *robot* opportune informazioni sui vincoli in termini di saturazione sulle diverse variabili d'interesse, queste verranno rispettate.

Dopo aver chiarito questi concetti, si effettua ora un'analisi del problema di controllo del moto di *robot* mobili illustrando alcune tecniche frequentemente utilizzate in letteratura. Una prima possibile tecnica è quella di suddividere il problema di navigazione in una fase di pianificazione del percorso e, solo successivamente, in una fase di esecuzione. Solitamente il problema della pianificazione della traiettoria è indirizzato e gestito in uno spazio di configurazione denominato *C-space,* dove i vari *robot* vengono rappresentati a ogni istante di tempo tramite un punto mobile nel piano cartesiano che rappresenta il totale spazio di lavoro. Il sottoinsieme del *C-space* che non presenta collisioni, con ostacoli o con altri agenti, ma contiene solo percorsi liberi *(path-free)* viene chiamato C_{free}. Un percorso nel *C-space* è quindi un *path-free* se e solo se esso è interamente contenuto nel C_{free}.

Assumendo dunque che le posizioni iniziali e finali dei vari *robot* siano note a priori e appartengano al *C-space,* il problema della pianificazione della traiettoria consiste nel generare un cammino libero che porti l'agente al raggiungimento del *goal* e che tale percorso sia interamente contenuto nel C_{free}. Per far questo viene come prima operazione generato un percorso privo di collisioni ad esempio tramite il metodo del *probabilistic planning.* Questa strategia si basa sull'idea di selezionare in modo casuale da un *set* finito di traiettorie del *C-space* tutti i *path-free* e successivamente costruire una mappa, denominata *roadmap,* come mostrato in Figura 2.1 , la quale contiene tutte le possibili traiettorie che portano a compimento il percorso del *robot,* dal punto iniziale al punto finale senza collisioni. Questa metodologia può essere però applicata in due differenti versioni [12], le quali sono sinteticamente descritte nel seguito:

1. *PRM* (*Probabilistic Roadmap*) in cui viene presa e testata casualmente una generica traiettoria, logicamente concorde con le possibilità cinematiche del *robot.* Se essa non genera collisioni viene aggiunta alla *roadmap,* altrimenti viene

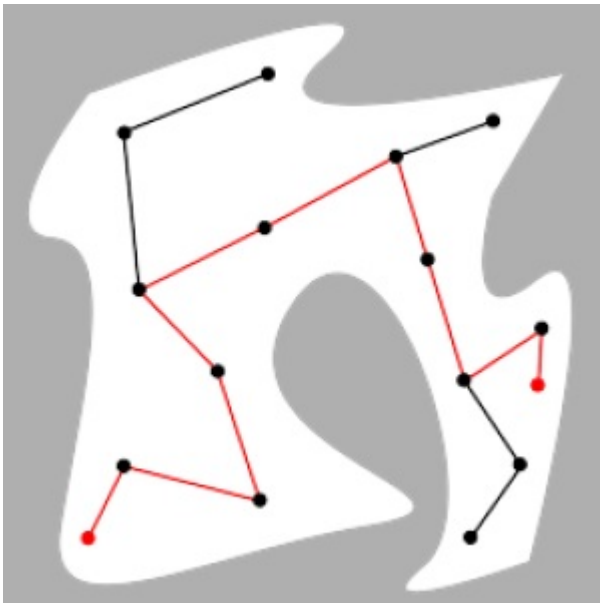


Figura 2.1: Esempio *roadmap*

scartata, come mostrato nell'esempio di Figura 2.2 . L'iterazione dell'algoritmo termina quando la *roadmap,* che ricordiamo essere finita, raggiunge la soglia del numero massimo di traiettorie inseribili. Il vantaggio di questa strategia è che si trova velocemente una soluzione ammissibile, ma solitamente non risulta essere la migliore possibile;

2. *RRT* (*Rapidly-exploring random trees*) l'algoritmo si basa sulla generazione di un albero, che si espande di nodo in nodo in modo casuale, come mostrato in Figura 2.3. Questo metodo utilizza una struttura ad albero e fa in modo, trovata una traiettoria parziale e libera che permette al *robot* di avvicinarsi al suo *goal,* di confrontarla con le sue traiettorie libere vicine fino a raggiungere il *goal* stesso. L'algoritmo termina il suo corso quando viene trovata una traiettoria che porti il *robot* dal punto iniziale al *goal.* Rispetto alla versione *PRM* questa strategia risulta essere molto più efficace, infatti non controlla ogni volta tutto il *C-space* ma solo le traiettorie nelle vicinanze di quella raggiunta all'istante corrente; dato ciò, la traiettoria risultante sarà migliore, ma il tempo impiegato per ricavarla risulta essere di maggiore durata.

Indipendentemente dalla strategia utilizzata, si deve in secondo luogo progettare un controllore che garantisca che l'agente segua il percorso determinato. In questa fase di esecuzione possono essere implementati opportuni algoritmi di ottimizzazione basati su criteri come il minimo tempo di viaggio, la minima distanza percorsa, oppure la minimizzazione dell'energia richiesta.

Una seconda possibile strategia è invece quella che fa ricorso a metodi di controllo di tipo *sensor based*, incentrati sul problema di pianificazione interattiva del movimento all'interno di un ambiente di lavoro di tipo dinamico. Questa tecnica incorpora le informazioni del sensore che riflette lo stato attuale dell'ambiente circostante, in relazione al processo di pianificazione del *robot.* Quindi essa, al contrario delle tecniche di pianificazione classica dove si ha la piena conoscenza della geometria dell'ambiente circostante, avrà a che fare con un ambiente sconosciuto in cui degli agenti sono

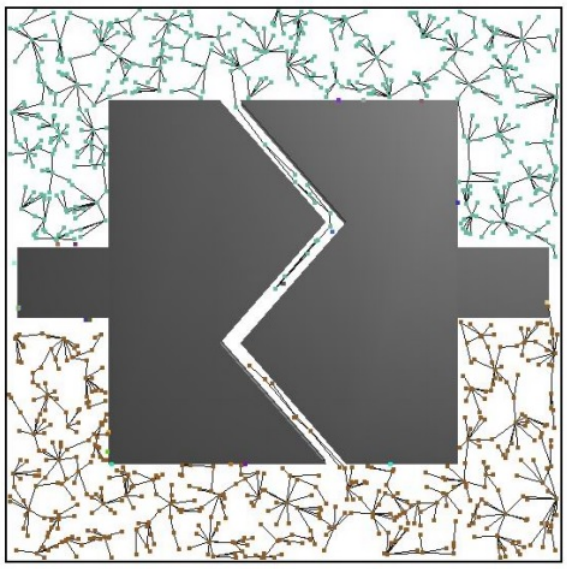


Figura 2.2: *Probabilistic Roadmap (PRM)*

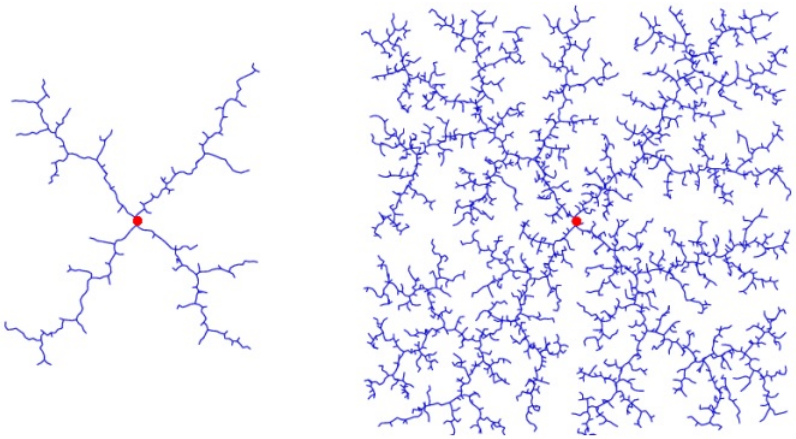


Figura 2.3: Evoluzione del numero di nodi e del numero di rami all'aumentare delle iterazioni dell'algoritmo *Rapidly-exploring random trees (RRT)*

inseriti per svolgere una determinata operazione. Con questa definizione si intende, in particolare, un ambiente la cui struttura può variare nel tempo, ad esempio per la presenza di altri *robot* in movimento o per l'interazione di operatori umani o altri eventuali ostacoli. A causa di tale variabilità imprevista è quindi necessario fare ricorso a dei sensori, montati tipicamente a bordo dell'agente, che consentono di interagire con l'ambiente circostante. L'utilizzo di sensori nell'implementazione di questa tecnica consente di risolvere le seguenti problematiche:

- il *robot* spesso non ha una chiara conoscenza dello spazio operativo, ma può avere solo una consapevolezza grossolana di esso a causa della sua memoria limitata;

- il modello dell'ambiente è soggetto a imprecisioni e incertezze che possono essere compensate solamente con strategie di pianificazione basate su sensori, e quindi su conseguenti misurazioni;

- l'ambiente in cui gli agenti operano è soggetto a eventi imprevisti o situazioni in rapida evoluzione; grazie all'utilizzo di sensori a bordo è possibile tener conto e affrontare queste eventuali variabilità.

Al giorno d'oggi esiste, in letteratura, un gran numero di metodi per la pianificazione delle traiettorie, tuttavia molti di essi non risultano adatti a essere impiegati con metodologia *sensor based* in quanto questa tecnica presenta anche alcuni problemi. In primo luogo è necessario gestire al meglio le incertezze derivanti dai vari sensori. Inoltre risulta fondamentale l'operazione di esplorazione durante la quale il *robot* non è diretto a ricercare un obiettivo specifico, ma è invece indirizzato a esplorare puramente l'ambiente sconosciuto in modo da ricavarne tutte le caratteristiche potenzialmente utili al fine del controllo; avere delle misure chiare e precise da parte dei sensori risulta quindi essere una priorità. Il problema di esplorazione può essere motivato dalla seguente applicazione: si immagini che il compito di un *robot* sia quello di esplorare l'interno di un edificio crollato, a causa, ad esempio, di un terremoto, al fine di cercare dei sopravvissuti alla catastrofe. Chiaramente sarà impossibile avere una conoscenza a priori della geometria interna dell'edificio e pertanto il *robot* in primo luogo dovrà essere in grado di rilevare, con i suoi sensori di bordo, tutti i punti interni dell'edificio, seguendo inizialmente un percorso di esplorazione, e successivamente effettuare un'ispezione per ricercare eventuali superstiti. Questa serie di operazioni può essere facilitata dall'utilizzo di più agenti, di cui ad esempio uno esegue un'operazione di esplorazione di una stanza inviando i dati all'altro *robot* che nel frattempo può istantaneamente iniziare la ricerca di sopravvissuti all'interno dell'edificio. Una seconda problematica relativa a questa strategia consiste nel fatto che il *goal* non è mai definito a priori, ma deve essere sempre definito in modo dinamico; tuttavia questa operazione non sempre può essere svolta durante lo svolgimento delle operazioni. Solitamente gli schemi di controllo utilizzati con questo approccio sono di tipo "intelligente", come ad esempio quelli basati su logiche *fuzzy* oppure sull'apprendimento tramite reti neurali.

Un terzo approccio al problema di navigazione si ottiene facendo uso direttamente di una filosofia di tipo *motion control*, il cui obiettivo è semplicemente quello dell'inseguimento, il più possibilmente accurato, di un segnale di riferimento desiderato, caratterizzato da specifici requisiti temporali. Grazie a questa tecnica di controllo è

infatti possibile impostare direttamente gli attuatori del *robot* in modo tale da permettere l'inseguimento del segnale, ovvero della traiettoria, all'interno dello spazio operativo.

Dopo aver presentato questa panoramica, si può affermare che l'idea proposta nella tesi è di fatto costituita da un insieme di caratteristiche prese dalle diverse strategie di controllo appena illustrate, ma che d'altra parte non può essere ricondotta precisamente a nessuna di esse. In particolare l'obiettivo del lavoro è quello di sfruttare una strategia di controllo predittivo *MPC* di alto livello per risolvere *on-line*, in tempo reale, il problema di navigazione e coordinamento dei vari *robot*, senza dover definire completamente in anticipo una traiettoria da seguire, ovvero senza avere un'esplicita fase di *path planning*; per ulteriori specifiche si rimanda a [5]. Tale navigazione deve inoltre poter avvenire in presenza di ostacoli fissi noti e utilizzando più agenti; proprio di questi ultimi si suppone di poter conoscere, grazie all'approccio predittivo, la loro traiettoria futura, e quindi a fronte di ciò il controllo del movimento (*motion control)* avviene all'interno di un ambiente solo parzialmente incerto. Per tutte queste motivazioni la tecnica da noi utilizzata è un *mix* tra le varie strategie presentate che ci permette, oltre che ricavare la traiettoria per portare il *robot* al suo *goal,* di scegliere quella ottima tra esse. Bisogna inoltre sottolineare che in realtà la tecnica predittiva utilizzata permette di gestire il problema di pianificazione delle traiettorie, ma solo limitatamente all'orizzonte di predizione che si considera nell'algoritmo *MPC*, senza necessariamente definire in modo completo il percorso dalla posizione attuale fino all'obiettivo finale. Per il momento non si tiene inoltre in considerazione il fattore temporale, supponendo che non vi siano requisiti particolari sul tempo di viaggio e sul tempo di percorrenza della traiettoria da parte dei vari agenti.

In conclusione si può affermare che il controllo di alto livello appena descritto, sarà in grado di fornire una soluzione al problema di *path planning* e garantirà inoltre, tramite un'opportuna legge di controllo, il mantenimento del moto vero e proprio del *robot* lungo la traiettoria pianificata. Tutte queste problematiche verranno ampiamente discusse nel prosieguo della tesi; l'obiettivo di queste pagine è solamente quello di analizzare le varie possibilità offerte e presentare brevemente una soluzione al problema di pianificazione e inseguimento delle traiettorie.

2.2 Navigazione degli agenti

Ci si occupa ora di analizzare il problema della navigazione degli agenti nello spazio operativo, la cui dinamica è fortemente e direttamente collegata al problema d'inseguimento della traiettoria. Supponiamo di aver generato una traiettoria di riferimento con le seguenti coordinate (x_r, y_r, θ_r) per la quale a ogni istante di tempo valgono le seguenti relazioni:

$$\begin{cases} \dot{x}_r = & v_r \cos\theta_r \\ \dot{y}_r = & v_r \sin\theta_r \\ \dot{\theta}_r = & \omega_r \end{cases}$$

e per la quale la velocità lineare può essere ricavata in questo modo:

$$v_r = \sqrt{\dot{x}_r^2 + \dot{y}_r^2}$$

dove sia la velocità lineare $v_r(t)$ sia la velocità angolare $\omega_r(t)$ rispettano i limiti imposti dalla saturazione. Supponiamo inoltre di avere libero accesso allo stato del

sistema, cioè di conoscere lungo tutta la traiettoria percorsa dal *robot* il vettore (x, y, θ). In queste condizioni avremo evidentemente *tracking* perfetto se risulta che a ogni istante di tempo sono verificate le relazioni:

$$\begin{cases} x &= x_r \\ y &= y_r \\ \theta &= \theta_r \end{cases}$$

A questo punto, per poter procedere con la soluzione del problema di controllo, risulta vantaggioso definire l'errore di posizionamento del *robot*, in un sistema di riferimento locale all'agente stesso, ovvero solidale con esso. In particolare, ricordando che nella descrizione originale (x, y) sono le coordinate cartesiane del centro del *robot* e θ è l'angolo tra la direzione di viaggio, lungo la quale è diretta la velocità dell'agente, e l'asse x possiamo costruire facilmente la trasformazione di coordinate che produce le variabili desiderate:

$$\begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - x_r \\ y - y_r \\ \theta - \theta_r \end{bmatrix}$$

Nelle nuove variabili, che descrivono l'evoluzione dell'errore, otteniamo un sistema dinamico del tutto equivalente a quello di partenza descritto mediante le equazioni seguenti:

$$\begin{cases} \dot{x}_e = & \omega y_e - v + v_r \cos\theta_e \\ \dot{y}_e = & -\omega x_e + v_r \sin\theta_e \\ \dot{\theta}_e = & \omega_r - \omega \end{cases} \tag{2.1}$$

Il vero vantaggio, sfruttando la nuova formulazione, risulta essere dato dal fatto che l'inseguimento perfetto della traiettoria nel problema di partenza corrisponde ora ad avere:

$$\begin{cases} x_e = & 0 \\ y_e = & 0 \\ \theta_e = & 0 \end{cases} \tag{2.2}$$

e ciò significa, in altre parole, che il problema originale di inseguimento della traiettoria può essere studiato come problema di stabilizzazione dell'origine del nuovo sistema descritto, semplificando così notevolmente le cose.

2.3 Inseguimento della traiettoria

Si passa ora ad analizzare il problema di inseguimento della traiettoria tralasciando per il momento il fatto di come essa, o in generale il percorso di riferimento, venga generata. Specificando con $q(t)$ lo stato del *robot* e con $q_r(r)$ le sue coordinate desiderate l'errore generico dell'agente è definito nel seguente modo:

$$e_q = q(t) - q_r(r) \tag{2.3}$$

dove r rappresenta un parametro che ha uno specifico significato a seconda del metodo utilizzato.

Nella letteratura esistono alcune tecniche che garantiscono la stabilità locale nell'intorno dell'origine per il modello (2.1) precedentemente descritto. Il nostro compito sarà infatti quello di assicurare che non si generi instabilità nel sistema data dal fatto che la posizione effettiva degli agenti si allontani troppo dal riferimento e quindi che i valori degli errori x_e, y_e, θ_e, descritti in (2.1), non crescano eccessivamente ma tendano alla soluzione di equilibrio (2.2). Se tale condizione viene rispettata e quindi, come detto, il modello (2.1) resta confinato in un intorno dell'origine viene garantita la stabilità del sistema il che assicura che la traiettoria pianificata venga correttamente inseguita dall'agente. In accordo con quanto detto si descrivono ora due tecniche che consentono la corretta risoluzione dei problemi di inseguimento della traiettoria e che verranno inoltre utilizzate nel prosieguo della trattazione. I due metodi presentati sono:

- *trajectory tracking* che si occupa della progettazione di leggi di controllo che consentono a un agente di effettuare un'azione di inseguimento di una traiettoria di riferimento, da parte di tutte le variabili di stato del sistema soggette a un determinato *setpoint*, dato dalla traiettoria stessa, imponendo vincoli temporali ben specificati. Lo svantaggio di questa tecnica è che può essere utilizzata solamente se si conosce a priori l'intera traiettoria da inseguire, infatti il vincolo temporale viene posto sulla totalità del percorso. In questo caso la variabile r, mostrata nella formula (2.3), rappresenta un istante temporale o più precisamente una funzione temporale del tipo $r(t) = t$ e in questo caso quindi e_q assumerà il significato di errore d'inseguimento della traiettoria di riferimento. L'approccio classico del *trajectory tracking* si basa su una linearizzazione locale del sistema e un disaccoppiamento del modello multi-variabile in modo da far sì che il numero di gradi di libertà del sistema stesso sia pari al numero di ingressi di controllo disponibili. Tipicamente problemi di *tracking* per veicoli autonomi sono risolti con leggi di controllo che permettono all'agente di rimanere all'interno delle regioni di ammissibilità, dove quindi è sempre possibile definire traiettorie che specificano l'evoluzione temporale del *robot*, l'orientamento, la sua velocità angolare e lineare tutte coerenti con i vincoli dinamici del sistema. Questo approccio presenta tuttavia l'inconveniente che solitamente i veicoli dinamici sono soggetti a numerosi termini non lineari complessi e ricchi di incertezza, si dovrà quindi trovare un metodo atto alla linearizzazione del sistema che ci consenta così successivamente di poter utilizzare questa tecnica. Possiamo quindi enunciare il problema di *trajectory tracking* in questo modo: avendo $q_r(t) : [0, \infty) \to \mathbb{R}$ funzione definita su un sufficiente intervallo di tempo per permettere il corretto inseguimento di una traiettoria desiderata con le sue derivate temporali limitate, allora il controllore progettato dovrà far in modo che tutte le variabili utilizzate rispettino i vincoli imposti e che l'errore di *tracking* converga in un intorno dell'origine il quale può essere preso arbitrariamente piccolo;

- *path following:* nel caso in cui non si abbiano vincoli temporali stringenti si utilizza questa tecnica che permette al *robot* di inseguire in modo corretto un percorso parziale o completo. Tuttavia in questo caso, non avendo vincoli temporali, il tempo di raggiungimento del *goal* potrebbe risultare elevato e non si ha più quindi nessuna sicurezza del fatto che l'agente raggiunga l'obbiettivo finale entro un tempo ben definito. Con questa strategia si garantisce la

convergenza della traiettoria verso il suo riferimento. Questo è spiegabile poiché, avendo sotto controllo le variabili di stato del sistema $q(t)$, si è sempre in grado, istante per istante, di ricavare il punto preciso della traiettoria da inseguire. In questo caso quindi il parametro r, mostrato nella formula (2.3), deve essere interpretato come funzione della posizione attuale del *robot* lungo il percorso da seguire, quindi sarà nella forma $r = \lambda(q)$, con λ che rappresenta la posizione attuale dell'agente lungo il percorso. Utilizzando questo approccio si deve assumere che il veicolo che insegue la traiettoria presenti un determinato profilo di velocità desiderato, che deve essere mantenuto dal controllore, il quale ha anche il compito di modificare in modo opportuno l'orientamento dell'agente. Come detto, grazie a questa tecnica, la convergenza verso la traiettoria di riferimento è garantita e inoltre gli attuatori del sistema, essendo limitatamente sollecitati, si mantengono lontani dalla zona di saturazione. Si può dunque affermare che l'approccio del *path following* può essere scomposto in due sotto-problemi: il primo che consiste in operazioni geometriche, le quali consentono al *robot* di rimanere all'interno di un determinato *range* nell'inseguire il riferimento; mentre il secondo sotto-problema consiste nell'effettuare un assegnamento di operazioni di tipo dinamico. Si enuncia allora il problema di *path following* in questo modo: avendo $q_r(\lambda) : [0, \infty) \to \mathbb{R}$ che sia un desiderato percorso parametrizzato con $\lambda \in \mathbb{R}$ e $v_r(\lambda) \in \mathbb{R}$, dove per semplicità si assume che la velocità assegnata non dipenda direttamente dal tempo e che $q_r(\lambda)$ abbia derivate limitate. Il controllore dovrà essere progettato tramite una legge in retroazione sulle variabili u_v, u_ω che rappresentano rispettivamente la velocità lineare e angolare dell'agente e $\ddot{\lambda}$ la sua accelerazione, così che i segnali in anello chiuso siano limitati, la posizione dell'agente resti confinata all'interno di un determinato insieme definito attorno alla traiettoria desiderata e il veicolo soddisfi un dato profilo di velocità v_r lungo tutto il percorso. Quindi anche l'errore di velocità: $\dot{\lambda}(t) - v_r(\lambda(t))$ dovrà essere confinato all'interno di un insieme piccolo e limitato.

Il limite fondamentale di questi approcci è dato dal fatto che la stabilità è garantita solamente in una specifica regione di stati operativi ed è proprio in essi che si dovrà far lavorare il sistema. Inoltre la grande differenza tra queste due strategie sta nel fatto che le limitazioni di prestazioni, dovute alla presenza di zeri dinamici instabili, possono essere rimosse usando l'approccio del *path following,* ma non utilizzando la strategia del *trajectory tracking;* per ulteriori analisi si rimanda alla trattazione [14].

Per capire quale delle due tecniche sia quella opportuna per il nostro problema, si deve aprire una breve parentesi; si ipotizzi che il sistema reale sia soggetto a una perturbazione non nota che costringe il *robot* a restare vincolato nella posizione di partenza. Utilizzando in questo caso un approccio di tipo *path following* il punto da inseguire resta fermo, in quanto la perturbazione va ad agire direttamente sullo stato del *robot,* che quindi rimarrà inalterato. Nel caso invece che si utilizzi l'approccio *trajectory tracking* il *setpoint*, che questa volta dipende dal tempo, continuerà a muoversi indipendentemente dal comportamento delle variabili di stato del sistema e quindi l'errore d'inseguimento della traiettoria di riferimento continuerà a crescere monotonicamente pur essendo irraggiungibile. In questo caso risulta quindi essere migliore l'approccio *path following.* Nel caso in cui venga inoltre introdotto un vincolo stringente sul tempo di raggiungimento del *goal* si può cercare di combinare entrambi gli algoritmi in modo da avere sempre sotto controllo la posizione attuale

dell'agente per una traiettoria parziale o totale e in più poter inserire vincoli temporali circa il raggiungimento dell'obiettivo finale; questa idea è proprio quella che si è cercato di implementare in questo lavoro. Il combinare queste due strategie è possibile utilizzando un parametro di peso che consente di ottenere i benefici del *path following* quando gli errori di inseguimento risultano diventare troppo elevati, ma di continuare comunque a preservare i vincoli temporali durante la fase di completamento dell'obiettivo. Tornando all'esempio fatto poco fa, si può affermare che se una perturbazione non nota tende ad amplificare gli errori, viene utilizzata la tecnica di *path following* così da non avanzare lungo il percorso, mentre quando il *robot* riesce ad avvicinarsi al *goal* si proseguirà utilizzando la tecnica del *trajectory tracking.* Dovremo quindi fare in modo che il parametro r possa assumere, nello stesso codice, sia il ruolo di una funzione spaziale che quello di una funzione temporale. Una funzione che comprenda entrambe le caratteristiche deve essere progettata tenendo conto di alcune proprietà fondamentali; essa infatti dovrà agire sull'errore e_q in modo da far si che quando esso è piccolo il *robot* reale e il suo riferimento possano avanzare simultaneamente in modo corretto, mentre se l'errore assume elevate dimensioni, ovvero se il *robot* e il *goal* corrispondente sono ad elevata distanza, il punto di riferimento mobile deve fermarsi e attendere. Questo ragionamento può essere sostenuto dal fatto che quando gli errori sono grandi nel *path following* ci si aspetta che il punto proiettato sul percorso desiderato vari lentamente, ovvero che la sua dinamica tenda a zero. Ora si deve solamente definire la fase di *switch* tra le due strategie, che sarà appunto dettata dall'ampiezza dell'errore ovvero dal fatto che il punto mobile, rappresentate la posizione reale dell'agente, sarà fermo o avrà una certa velocità e quindi sarà in moto; il trucco per avere una buona tecnica di commutazione sarà quello di cercare di fare in modo che l'errore e_q resti sempre limitato all'interno di un certo *range* di valori.

Capitolo 3

Feedback Linearization e Predittore di Kalman

In questo capitolo si andrà a riprendere la struttura del sistema di controllo di *robot* mobili presentata in [13] spiegando, nei vari paragrafi, come essa è stata modificata in questa tesi per consentire all'utente del *robot* di ottenere un sistema nettamente più performante in termini di robustezza ai disturbi e di maggiore velocità nel raggiungere il proprio obiettivo.

3.1 Linearizzazione del modello tramite Feedback Linearization

Come già osservato, il *robot e-puck* è descritto da un modello non lineare e quindi difficile da gestire tramite un approccio di tipo *MPC*, come quello che si desidera implementare. Per poter realizzare un'efficiente implementazione *on-line* del controllore *MPC*, è necessario invece che il modello del sistema e i componenti che costituiscono il sistema di controllo siano semplici, ovvero si dovranno utilizzare modelli lineari e, relativamente ai problemi di ottimizzazione discussi nel prosieguo, cifre di costo quadratiche unite a eventuali vincoli lineari. Avendo quindi un modello uniciclo che risulta essere non lineare e sapendo che le forti non linearità presenti non possono essere trascurate, in quanto determinanti nella dinamica dell'agente, si è scelto di applicare un anello di controllo linearizzante che permetta di ottenere un sistema in anello chiuso lineare. Andremo quindi ad applicare la tecnica di *Feedback Linearization*.

Fissato il sistema di riferimento nel piano di lavoro, si considerino le accelerazioni lungo i due assi:

$$\begin{cases} a_x = & \ddot{x} = \dot{v}\cos\theta - v\dot{\theta}\sin\theta = \dot{v}\cos\theta - v\omega\sin\theta \\ a_y = & \ddot{y} = \dot{v}\sin\theta + v\dot{\theta}\cos\theta = \dot{v}\sin\theta + v\omega\cos\theta \end{cases}$$

In forma matriciale si ottiene:

$$\begin{pmatrix} a_x \\ a_y \end{pmatrix} = \begin{bmatrix} \cos\theta & -v\sin\theta \\ \sin\theta & v\cos\theta \end{bmatrix} \begin{pmatrix} \dot{v} \\ \omega \end{pmatrix}$$

Si noti come per $v = 0$ la matrice risulti essere singolare e quindi il problema non sia risolvibile, mentre se $v \neq 0$ la trasformazione è invertibile e si ottiene il seguente controllore dinamico linearizzante del primo ordine:

$$\begin{cases} \dot{v} = & a_x \cos\theta + a_y \sin\theta \\ \omega = & \frac{-a_x \sin\theta + a_y \cos\theta}{v} \end{cases}$$

Applicando tale controllore all'uniciclo, il sistema retroazionato risulta essere:

$$\begin{cases} \ddot{x} = & a_x \\ \ddot{y} = & a_y \end{cases}$$

che, come desiderato, è lineare e presenta lo schema di controllo mostrato in Figura 3.1, dove (a_x, a_y) rappresentano le nuove variabili di controllo del sistema, ovvero i

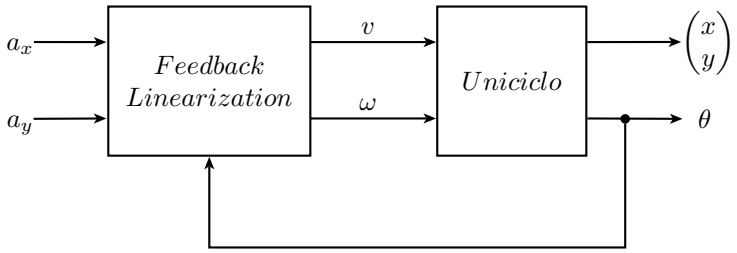


Figura 3.1: Schema di controllo uniciclo con *Feedback Linearization*

nuovi ingressi che dovranno essere ricavati tramite una qualsiasi legge linearizzante. Si nota in modo immediato che tramite tecnica di *Feedback Linearization* il sistema ottenuto è come se fosse composto da una coppia di doppi integratori tra loro completamente indipendenti. In sintesi, la legge in *feedback* consente dunque di realizzare un anello di controllo interno che rende esternamente lineare il sistema a uniciclo. Sottolineiamo nuovamente che la linearizzazione utilizzata è di tipo dinamico, ovvero dipende strettamente dal valore assunto dalle variabili di stato. Al contrario, una soluzione di tipo statico non è applicabile a causa della presenza del vincolo anolonomo, mentre la linearizzazione algebrica porterebbe a un sistema non completamente controllabile; per ulteriori chiarimenti si rimanda alla trattazione [6]. Una semplice implementazione discreta, anche se non molto precisa, dell'anello linearizzante si ottiene mediante il metodo di discretizzazione di Eulero in avanti, ottenendo così la seguente regola per l'aggiornamento del controllore:

$$\begin{gathered} v_p \leftarrow a_x \cos\theta + a_y \sin\theta \\ \omega \leftarrow (-a_x \sin\theta + a_y \cos\theta)/v_{old} \\ v \leftarrow v_{old} + v_p \tau \end{gathered} \tag{3.1}$$

in cui τ è il tempo di campionamento e v_p è l'incremento di velocità. La terza equazione della (3.1) rappresenta l'integrazione numerica a tempo discreto fatta per consentire l'aggiornamento della velocità, incrementando a ogni passo di campionamento il valore cumulato della velocità stessa.

È importante osservare che, avendo la possibilità di impiegare anelli annidati, la scelta della frequenza di campionamento per questo anello, ovvero quello interno, deve essere fatta in modo che esso operi a velocità molto maggiore di quella di eventuali anelli di controllo esterni.

Il sistema lineare ottenuto chiudendo l'anello può essere rappresentato nello spazio di stato come:

$$\begin{cases} \begin{pmatrix} \dot{\xi}_1(t) \\ \dot{\xi}_2(t) \\ \dot{\xi}_3(t) \\ \dot{\xi}_4(t) \end{pmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} \xi_1(t) \\ \xi_2(t) \\ \xi_3(t) \\ \xi_4(t) \end{pmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} a_x(t) \\ a_y(t) \end{pmatrix} \\ \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} \xi_1(t) \\ \xi_2(t) \\ \xi_3(t) \\ \xi_4(t) \end{pmatrix} \end{cases}$$

Discretizzando ora il modello tramite la discretizzazione esatta, si ottiene il seguente sistema a tempo discreto:

$$\begin{cases} \begin{pmatrix} \xi_1(k+1) \\ \xi_2(k+1) \\ \xi_3(k+1) \\ \xi_4(k+1) \end{pmatrix} = \begin{bmatrix} 1 & \tau & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \tau \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} \xi_1(k) \\ \xi_2(k) \\ \xi_3(k) \\ \xi_4(k) \end{pmatrix} + \begin{bmatrix} \tau^2/2 & 0 \\ \tau & 0 \\ 0 & \tau^2/2 \\ 0 & \tau \end{bmatrix} \begin{pmatrix} a_x(k) \\ a_y(k) \end{pmatrix} \\ \begin{pmatrix} x(k) \\ y(k) \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} \xi_1(k) \\ \xi_2(k) \\ \xi_3(k) \\ \xi_4(k) \end{pmatrix} \end{cases}$$

Si noti, come evidenziato, che la dinamica lungo le due direzioni x e y risulta essere disaccoppiata, permettendo così di risolvere separatamente il problema di controllo. In forma più compatta si può scrivere:

$$\begin{cases} \xi(k+1) = & A\xi(k) + Bu(k) \\ z(k) = & C\xi(k) \end{cases} \tag{3.2}$$

La tecnica di *Feedback Linearization* presenta quindi l'indubbio vantaggio di rendere disponibile all'esterno un sistema completamente lineare con un comportamento dinamico abbastanza semplice. Questo ha consentito, successivamente, di affrontare il problema di controllo *MPC* evitando il ricorso alla teoria non lineare. Per riferimenti più dettagliati circa la *Feedback Linearization* si rimanda a [13].

La limitazione di questo approccio è tuttavia dovuta al fatto che la velocità degli agenti è ricavata in *open loop* e, non avendo quindi nessuna retroazione, non si è certi che i valori ottenuti siano affidabili e quindi corretti. Uno dei caratteri innovativi di questa tesi è stato quindi quello di utilizzare un predittore di *Kalman* per ovviare a tale problema permettendo così di stimare lo stato dei *robot* ricavando le misurazioni di nostro interesse, in particolare la velocità.

3.2 Stima dello stato con il predittore di Kalman

Per stimare al meglio lo stato del sistema si è fatto ricorso alla teoria di *Kalman,* costruendo un predittore che verrà poi inserito all'interno dell'architettura di controllo. Si considera inizialmente il sistema:

$$\begin{cases} x(k+1) = & Ax(k) + Bu(k) + w_x(k) \\ y(k) = & Cx(k) + w_y(k) \end{cases}$$

dove

$$w = \begin{bmatrix} w_x \\ w_y \end{bmatrix}$$

è un rumore bianco gaussiano a valore atteso nullo e matrice di covarianza

$$V = \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix}$$

con $Q \in \mathbb{R}^{n,n}$, $R \in \mathbb{R}^{p,p}$, $Q \geq 0$, $R > 0$.

Si definisca con $\hat{x}(k \mid k-1)$ la predizione a un passo di $x(k)$, ovvero la stima dello stato x all'istante k fatta a partire dalla conoscenza degli ingressi u e delle uscite y fino all'istante passato $k-1$. La formula del predittore è quindi data dalla seguente equazione:

$$\hat{x}(k+1 \mid k) = A\hat{x}(k \mid k-1) + Bu(k) + L(k)[y(k) - C\hat{x}(k \mid k-1)]$$

dove $L(k)$ rappresenta il guadagno del predittore stesso.

L'obiettivo che quindi ci si pone è quello di determinare il guadagno $L(k)$ in modo da minimizzare una generica forma quadratica nella covarianza dell'errore; dalla teoria sappiamo che il guadagno $L(k)$ che minimizza la covarianza dell'errore è:

$$L(k) = AP\left(k \mid k-1\right) C' \left[CP\left(k \mid k-1\right) C' + R\right]^{-1}$$

dove $P(k \mid k-1)$ è la soluzione dell'equazione di Riccati:

$$\begin{aligned} P(k+1 \mid k) = \quad & AP(k \mid k-1)A' + Q + \\ & -AP(k \mid k-1)C'[CP(k \mid k-1)C' + R]^{-1}CP(k \mid k-1)A' \end{aligned}$$

con condizione iniziale $P(0 \mid -1) = P_0$.

Il secondo teorema di convergenza [25] afferma che se la coppia (A, B_q), con $B_q : Q = B_q B_q'$, è raggiungibile e la coppia (A, C) è osservabile, allora il predittore ottimo di regime è:

$$\hat{x}(k+1 \mid k) = (A - \bar{L}C)\hat{x}(k \mid k-1) + Bu(k) + \bar{L}y(k) \tag{3.3}$$

con

$$\bar{L} = A\bar{P}C'[C\bar{P}C' + R]^{-1}$$

e $\bar{P}$ è l'unica soluzione definita positiva dell'equazione algebrica di Riccati:

$$\bar{P} = A\bar{P}A' + Q - A\bar{P}C'[C\bar{P}C' + R]^{-1}C\bar{P}A'$$

Inoltre lo stimatore risulta essere asintoticamente stabile, cioè gli autovalori della matrice $(A - \bar{L}C)$ hanno modulo minore di uno e sono così contenuti nella regione di stabilità; per una più chiara e completa descrizione di tale tecnica si rimanda a [2].

Dopo questa breve introduzione andiamo ad analizzare ora il caso specifico, in cui il sistema in esame è descritto dalle seguenti matrici:

$$A = \begin{bmatrix} 1 & \tau & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \tau \\ 0 & 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} \tau^2/2 & 0 \\ \tau & 0 \\ 0 & \tau^2/2 \\ 0 & \tau \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, D = 0$$

dove τ rappresenta il tempo di campionamento utilizzato. Esse hanno la proprietà di essere diagonali a blocchi, il che ci consente di concentrare, quando necessario, l'attenzione su un solo sottosistema ridotto, considerando solo successivamente la restante parte del tutto equivalente alla prima. Utilizzando l'accorgimento descritto si è in grado di semplificare notevolmente il problema computazionale al momento dell'implementazione effettiva, poiché tutti i calcoli matriciali avranno dimensioni dimezzate. Si può quindi utilizzare sempre tale osservazione anche per progettare il predittore del sistema, dato che anche la matrice dei guadagni $\bar{L}$ precedentemente descritta risulterà diagonale a blocchi; in questo modo sarà quindi possibile lavorare, come detto, su un sistema ridotto che potrà essere eventualmente allargato in seguito.

Prendendo in considerazione le matrici appena analizzate si costruisce su di esse, per il sistema a tempo discreto, il predittore di *Kalman* nella forma (3.3) che consenta di stimare lo stato del sistema agli istanti di tempo futuri a partire dai dati disponibili dagli istanti passati. Nello specifico sappiamo che gli ingressi del sistema sono le due accelerazioni del *robot* (a_x, a_y), mentre le uscite sono le coordinate (x, y) relative alla sua posizione. Grazie a ciò siamo in grado di stimare lo stato del sistema ricavando quindi a ogni istante la posizione e la velocità dell'agente.

3.2.1 Verifica della bontà della stima

Si analizza in questa sezione, facendo ricorso ad alcune simulazioni e senza prestare per ora attenzione alla struttura e ai vari livelli del codice, la bontà della stima dello stato ottenuta grazie all'utilizzo del predittore di *Kalman,* soffermandoci solo ed esclusivamente sui risultati ottenuti.

Dato un agente posizionato in un certo punto iniziale nel piano di riferimento, tramite la risoluzione di un problema di ottimizzazione della traiettoria, si genera un determinato riferimento che permette al *robot* di raggiungere nel modo migliore possibile il suo obiettivo impostato a priori. Nella simulazione in esame si è controllato un singolo agente che, partendo da una posizione di coordinate:

$$\begin{cases} x &= 30 \\ y &= 35 \end{cases}$$

sul piano di riferimento, deve raggiungere il *goal* posto alle coordinate:

$$\begin{cases} x &= 100 \\ y &= 15 \end{cases}$$

Nella Figura 3.2 , ricavata dalla simulazione, viene mostrato un confronto tra la tra-

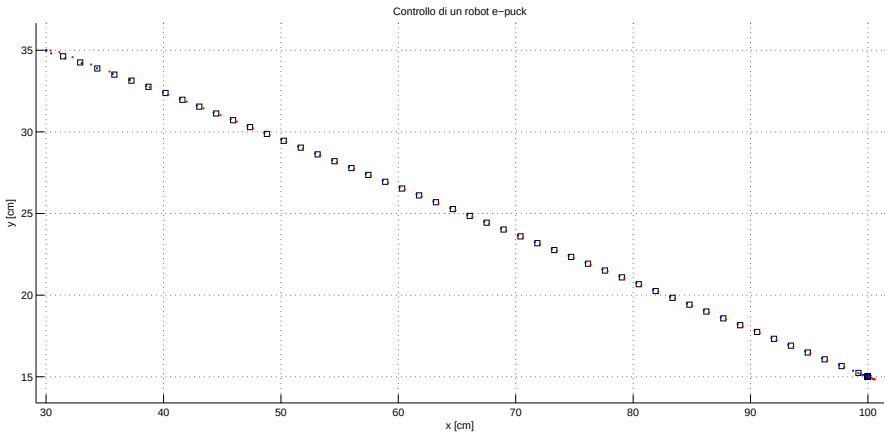


Figura 3.2: Posizione e riferimento del *robot*

iettoria seguita dall'agente in cui viene implementato nell'algoritmo di controllo, che verrà descritto dettagliatamente in seguito, lo stato simulato del sistema linearizzato (3.2) (andamento in blu) e lo stato del sistema stimato tramite predittore di *Kalman* (andamento in rosso), costruito nella forma (3.3). In nero, tramite quadratini, viene inoltre rappresentata la traiettoria di riferimento generata dall'algoritmo di ottimizzazione. Il risultato ottenuto è molto soddisfacente, in quanto si vede chiaramente che l'agente, in ambedue le traiettorie simulate, non si discosta mai troppo dal riferimento generato. In particolare, questo ci permette di affermare che la stima dello stato del sistema (posizioni e velocità), ricavata tramite predittore di *Kalman*, ci consente di ottenere una corretta risoluzione del problema esaminato portando infatti l'agente al suo *goal*. Tuttavia questo risultato, trattandosi semplicemente di una simulazione, non tiene conto di tutti i possibili disturbi che si incontrerebbero andando a implementare l'algoritmo di controllo sul sistema reale. Sarà quindi necessario per validare al meglio la bontà della stima ottenuta valutare nello specifico i vari andamenti di posizione, velocità e angolo di orientamento del *robot* andando a confrontare fra loro i valori stimati con il predittore di *Kalman*, costruito nella forma (3.3), quelli ottenuti mediante la simulazione del sistema linearizzato (3.2) e quelli ricavati direttamente, quando è possibile, tramite misurazioni sul sistema reale.

Si analizza inizialmente il grafico mostrato nella Figura 3.3 che si riferisce alla posizione del *robot*; da tale grafico si vede che i tre andamenti corrispondono quasi perfettamente, il che significa come prima cosa che la stima è veritiera riscontrando così ottimi risultati. Solamente effettuando un ingrandimento dell'immagine, come rappresentato nella Figura 3.4, si nota che, mentre la simulazione del sistema linearizzato sovrappone perfettamente il suo andamento a quello del sistema reale, la stima effettuata con il predittore si discosta leggermente; questo comunque non costituirà un problema in quanto, come vedremo, nell'implementazione del nostro algoritmo verrà utilizzata a ogni istante di campionamento la posizione (x, y) misurata dalla *webcam* ed è quindi lecito aspettarsi i risultati precedenti.

Si passa ora all'analisi della velocità degli agenti; essa rappresenta la vera incognita del sistema per cui è stato necessario costruire il predittore di *Kalman* in quanto, non essendo possibile misurarla, deve essere per forza stimata. Dall'analisi della Figura 3.5, si nota subito che l'andamento simulato del sistema linearizzato si

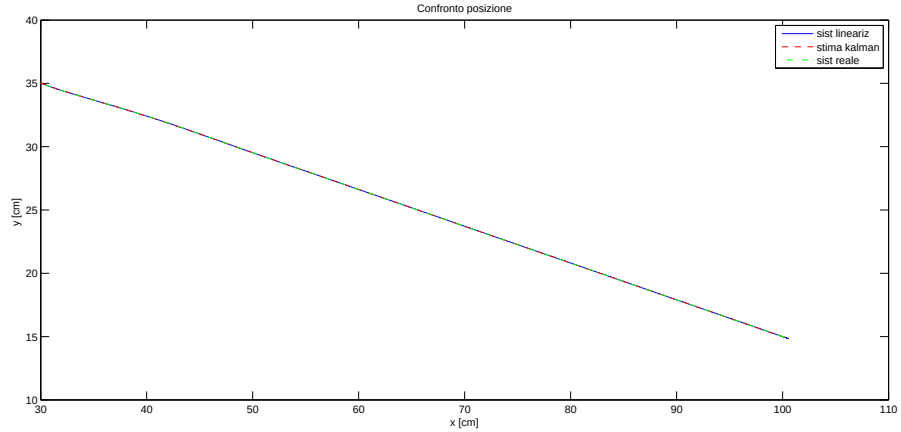


Figura 3.3: Confronto tra le posizioni del *robot*

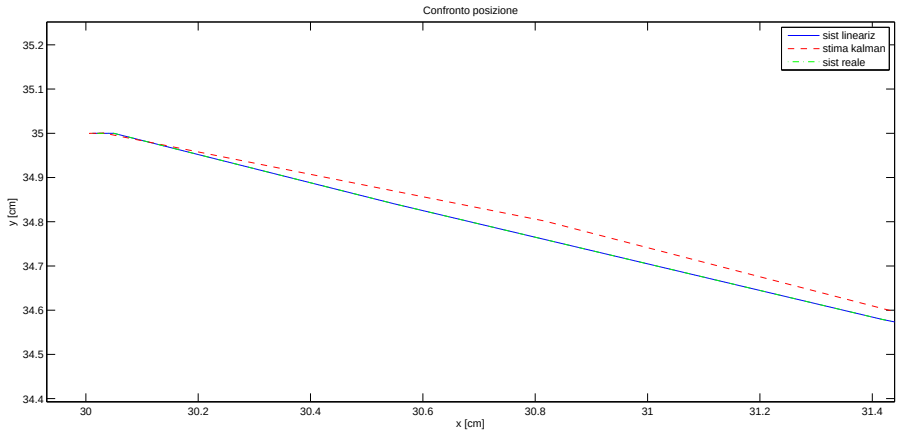


Figura 3.4: Confronto tra le posizioni del *robot* alla partenza

discosta da quello del sistema reale, che invece è simile a quello fornito dal predittore di *Kalman*. Si può quindi concludere che la velocità stimata tramite il predittore porta, come ci si aspettava, ad avere risultati molto migliori rispetto alla stima ottenuta semplicemente attraverso la simulazione *open-loop* del sistema lineare. È fondamentale osservare che la velocità longitudinale del sistema reale non può essere ricavata tramite misure; per velocità del sistema reale si intende infatti la derivata discreta della posizione misurata tramite *webcam* rispetto a due punti successivi, utilizzando la semplice formulazione:

$$v = \sqrt{\left(\frac{dx}{d\tau}\right)^2 + \left(\frac{dy}{d\tau}\right)^2} \tag{3.4}$$

questo per ogni istante di acquisizione. La discrepanza che esiste tra i valori ottenuti tramite simulazione *open-loop* e quelli del sistema reale è dovuta a disturbi diversi che agiscono proprio sul moto di quest'ultimo.

Ci si occupa infine dell'orientamento del *robot*; ponendo l'agente con un orientamento iniziale pari a $\theta_0 = 0$ si ottengono i risultati mostrati nella Figura 3.6. Da qui è ben visibile, anche in questo caso come per la posizione, che l'andamento della

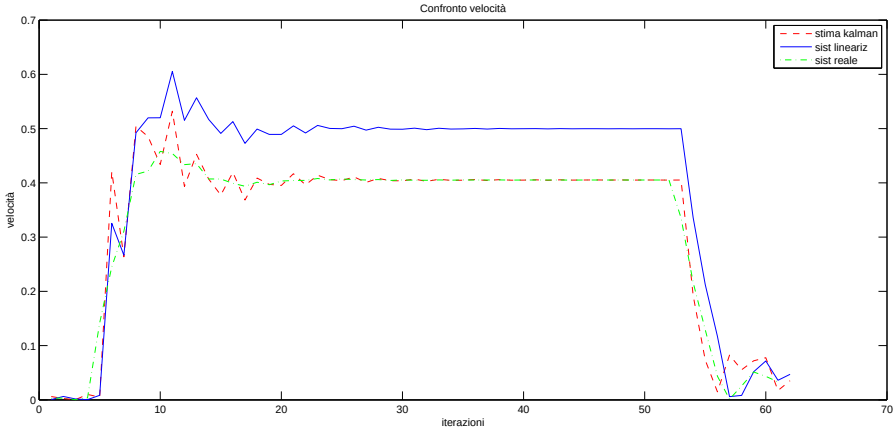


Figura 3.5: Confronto tra le velocità del *robot*

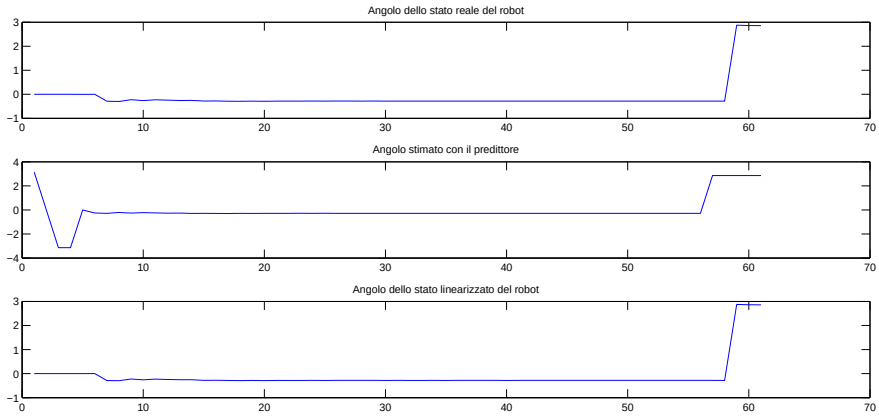


Figura 3.6: Confronto tra gli angoli del *robot* con $\theta_0 = 0$

simulazione in *open-loop* con il modello (3.2) rispecchia fedelmente il sistema reale, mentre quello stimato, o meglio quello calcolato tramite la stima delle velocità utilizzando la formula:

$$\theta = \arctan\left(\frac{v_y}{v_x}\right) \tag{3.5}$$

risulta essere molto buono per valori centrali, ma non accettabile nelle fasi di partenza e di arrivo, che corrispondono ai tratti in cui la velocità del *robot* risulta essere molto limitata. Tuttavia è ben osservabile che comunque l'agente in tutti e tre gli andamenti presenta un angolo conclusivo, al raggiungimento del *goal*, analogo. Provando ora a porre il *robot* con orientamento iniziale pari a $\theta_0 = \pi$, ovvero con l'agente ruotato dalla parte opposta rispetto al suo *goal*, si ricavano i risultati mostrati nella Figura 3.7. In questo caso addirittura tutti e tre gli andamenti sono fortemente diversi nella fase di arrivo e di partenza. Questo risultato ha indotto a utilizzare, nell'algoritmo sviluppato in seguito, il valore di θ misurato direttamente dalla *webcam* che sappiamo essere veritiero, piuttosto che il valore stimato attraverso la formula (3.5), dove i valori di v_x e v_y sono dati dal predittore di *Kalman*.

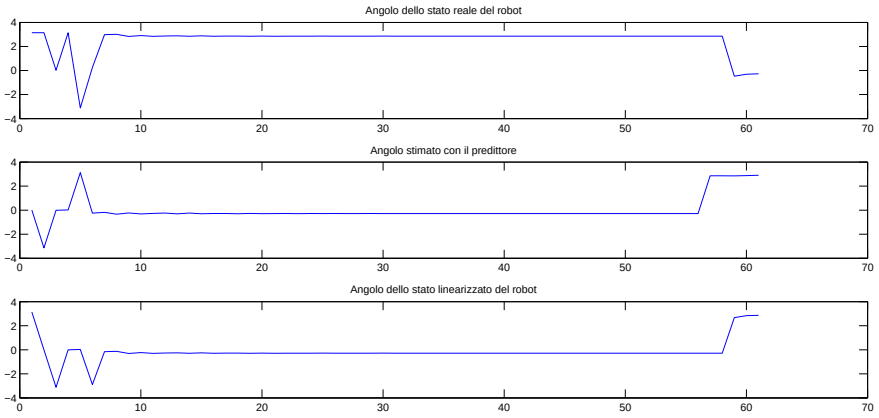


Figura 3.7: Confronto tra gli angoli del *robot* con $\theta_0 = \pi$

In conclusione, a parte i valori di velocità per cui la stima risulta essere migliore rispetto a quella fornita dalla simulazione del sistema linearizzato, ci si può chiedere se sia opportuno utilizzare il predittore di *Kalman* anziché la soluzione basata sul sistema di equazioni (3.2). La risposta è positiva in quanto con il predittore si può ottenere una stima adeguata della grandezza di interesse anche a fronte di disturbi o errori di misura, che causerebbero problemi significativi se si utilizzasse la simulazione *open-loop* del sistema lineare.

3.3 Trasferimento dell'anello di Feedback Linearization internamente all'agente

Un ulteriore carattere innovativo presentato in questa tesi è dato dal trasferimento e dall'implementazione dell'anello linearizzante internamente all'agente, il che provoca un netto cambiamento della struttura logica del sistema in esame.

3.3.1 Logica generale

Nella trattazione [13] si era analizzato un sistema che presentava una struttura sequenziale (cascata di operazioni), come mostrato nella Figura 3.8. Questo, come è ben visibile dalla figura, genera una catena sequenziale di operazioni che il sistema svolge in un certo numero di istanti computazionali che corrispondono a un determinato tempo fisico. Nello specifico, si ha un ciclo interno, relativo all'anello linearizzante, che viene eseguito un certo numero di volte con un tempo di campionamento minore rispetto al ciclo esterno relativo al problema di controllo predittivo, che viene invece eseguito una sola volta in corrispondenza dell'acquisizione di dati da parte della *webcam*. Si deduce quindi che il tempo per percorrere i due cicli è nettamente diverso, almeno di circa un fattore 10, il che genera una sorta di squilibrio e discrepanza temporale che si traduce in termini di perdita di efficienza del sistema. Nella Figura 3.8 è infatti mostrato un grafico dove si nota che l'anello interno viene svolto un elevato numero di volte, rappresentato in figura dalle frecce che occupano un tempo di risoluzione minore, e quando il ciclo di linearizzazione è completato si esce da esso per proseguire con l'anello esterno in cui vengono acquisiti

i dati dalla telecamera e dove verrà risolto il problema *MPC*. Logicamente il tempo computazionale di quest'ultimo ciclo sarà molto più elevato rispetto quello interno, infatti esso viene rappresentato con frecce di lunghezza maggiore nel pannello destro di Figura 3.8. Per spiegare la perdita di prestazioni del sistema, precedentemente

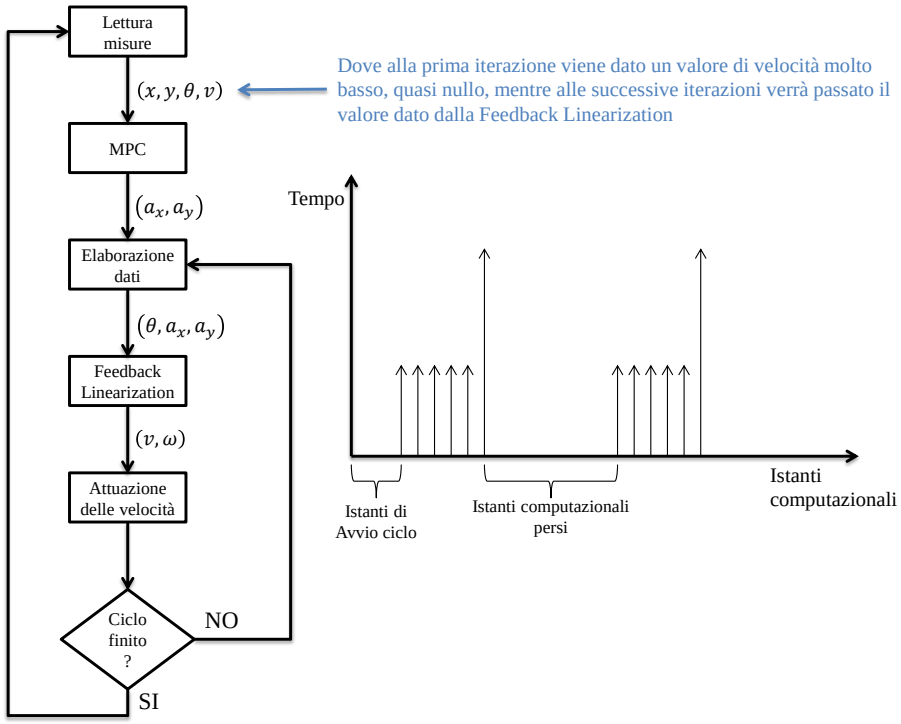


Figura 3.8: Logica di controllo sequenziale

accennata, è sufficiente analizzare sempre la Figura 3.8: dal grafico si nota chiaramente come vengano persi degli istanti computazionali nel momento in cui il sistema si trova a risolvere il problema di ottimizzazione del controllo predittivo ritardando così l'ingresso nell'anello interno che effettua la *Feedback Linearization*.

Utilizzando invece la tecnica presentata in questa trattazione, ovvero spostando l'anello linearizzante internamente al *robot,* è possibile ottenere una struttura logica che opera parallelamente sullo stesso livello, il che permette di non incorrere in problemi di perdite di efficienza; si ottiene infatti il sistema presentato nella Figura 3.9. A questo punto è sorto il problema che nell'algoritmo di controllo esterno all'agente non si potessero più utilizzare i valori del sistema linearizzato (3.2), come fatto in [13], in quanto la linearizzazione ora viene svolta internamente al *robot.* Tale fatto è stato prontamente risolto andando a utilizzare il predittore di *Kalman* appositamente progettato nella forma (3.3) per stimare la velocità da passare agli algoritmi di iterazione per la risoluzione del problema predittivo, mentre per quanto riguarda la posizione e l'orientamento vengono sempre passati i valori reali, acquisiti a ogni istante dalla telecamera. Avremmo potuto passare all'algoritmo anche la posizione stimata, visto che era disponibile e risulta essere buona come visto dai grafici precedenti, ma, a fronte di tutte le osservazioni fatte, si è ovviamente impiegata quella reale in quanto più precisa, a meno di errori da parte della telecamera. Inoltre, do-

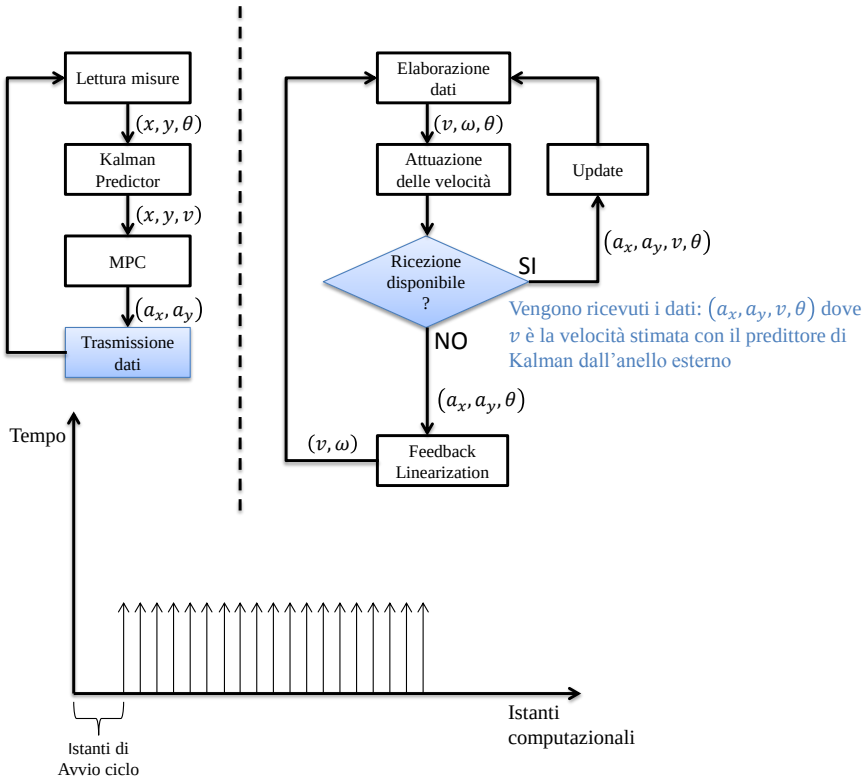


Figura 3.9: Logica di controllo in parallelo

vendo calcolare a ogni istante l'orientamento dei *robot* dalla *webcam*, che come visto risulta essere l'unico metodo esatto, si doveva in ogni caso acquisirne la posizione.

Operando in parallelo si hanno, quindi, due strutture ben definite, una interna e una esterna al *robot,* le quali risultano essere perfettamente gestite e sincronizzate tra loro grazie a un meccanismo di trasmissione e ricezione dei dati che sfrutta come condizione l'evento di acquisizione da parte dalla *webcam*, ovvero la condizione data dal fatto che la struttura esterna sia pronta o meno a inviare dati, relativi al problema di ottimizzazione, a quella interna all'agente. Dalla Figura 3.9 è chiaro che, mentre l'algoritmo esterno risolve il problema di *MPC*, internamente all'agente si andrà continuamente a implementare il ciclo relativo alla *Feedback Linearization* utilizzando valori di accelerazione, di velocità e di angoli θ inviati a esso direttamente dall'anello esterno o, in assenza di essi, ovvero nel caso non sia ancora stato risolto il problema di controllo predittivo, l'anello interno utilizzerà gli ultimi valori disponibili ricevuti dal ciclo esterno per implementare le seguenti equazioni dell'anello linearizzante:

$$\begin{aligned} \omega &\leftarrow (-a_x \sin\theta + a_y \cos\theta)/v_{old} \\ v_p &\leftarrow v_{old} + \tau_{int}(a_x \cos\theta + a_y \sin\theta) \\ \theta &\leftarrow \theta + \tau_{int}\omega \end{aligned} \tag{3.6}$$

In questo modo è possibile ricavare, per ogni ciclo di campionamento interno, i valori delle velocità da attuare al *robot*, questo finché non verranno inviati i nuovi dati relativi alla risoluzione del controllo *MPC* da parte dell'anello esterno. Da notare che per il ciclo interno si utilizza un tempo di campionamento τ_{int} che sarà diverso e nello specifico minore rispetto al tempo di campionamento τ con il quale viene

gestito l'anello esterno. Utilizzando questa tecnica è ben visibile dal grafico di Figura 3.9 come si riescano a sfruttare appieno e in modo intelligente tutte le potenzialità computazionali del *robot,* che risulterà quindi essere notevolmente più prestazionale in quanto vengono eliminate le cause di attesa e i problemi riguardanti i tempi morti visti relativamente alla struttura precedentemente studiata.

Lo spostamento dell'anello linearizzante internamente al *robot* ci permette quindi di diminuire il tempo di campionamento del ciclo di controllo esterno, riuscendo così a velocizzare la sua esecuzione, ma allo stesso tempo anche a diminuire le oscillazioni dell'agente in fase di inseguimento del riferimento, ottenendo così un sistema più robusto e prestazionale. Ciò è provato anche dal fatto che, mentre nella struttura sequenziale il tempo di campionamento dell'algoritmo di controllo doveva essere progettato sul ciclo esterno relativo al problema *MPC*, il quale doveva tener conto anche del tempo necessario per risolvere la linearizzazione, ora il tempo di campionamento sarà sempre progettato sul problema *MPC*, ma questa volta risulterà essere notevolmente minore in quanto non dovrà più aspettare anche la risoluzione della *Feedback Linearization.* Sarà inoltre più chiaro nel seguito che una diminuzione del tempo di campionamento τ permetterà un significativo incremento della reattività del sistema controllato, permettendo dunque un notevole aumento della velocità degli agenti.

3.3.2 Aspetti implementativi

Si fornisce ora una descrizione generale di come è stato implementato l'algoritmo di *Feedback Linearization* direttamente sugli agenti e di come è stato affrontato nello specifico il problema di comunicazione, ricezione e invio di pacchetti di dati, tra *robot* e *PC*, utilizzando la tecnologia *bluetooth.* In aggiunta a ciò, si ricorda che il codice sviluppato deve permettere, utilizzando le specifiche librerie, di comandare le varie porte seriali dell'*e-puck* gestendone sensori e attuatori ed essendo al tempo stesso compatibile e interfacciabile con l'algoritmo esterno all'agente che consente di gestire il problema di controllo predittivo. Per amministrare la comunicazione tra i due algoritmi, interno ed esterno al *robot*, si è utilizzato il seguente metodo: se sul *buffer* di comunicazione ($UART$) viene visualizzato un carattere 'v' si prendono i successivi quattro dati dopo tale carattere e li si interpreta come l'accelerazione lungo l'asse x, l'accelerazione lungo l'asse y, la velocità v e l'angolo θ seguiti dal carattere speciale '\0' che indica la fine della stringa di dati. In seguito, avendo spostato l'anello di *Feedback Linearization* internamente, si va a svolgere l'implementazione discreta dell'anello linearizzante che si ottiene, come già visto, tramite l'equazione (3.6). Successivamente si esegue una conversione lineare per passare da metri a *steps* al secondo di rotazione tramite le relazioni:

$$\begin{cases} v_R = & \frac{v+E\omega}{2} \\ v_L = & \frac{v-E\omega}{2} \end{cases}$$

e si ricavano le corrispondenti velocità angolari delle due ruote indipendenti:

$$\begin{cases} \omega_R = & \frac{V_R}{R} \\ \omega_L = & \frac{V_L}{R} \end{cases} \tag{3.7}$$

A questo punto non ci resta che utilizzare i risultati ottenuti dalla (3.7) e approssimarli all'intero più vicino per andare a ricavare le vere velocità con cui si vanno

ad attuare i motori; questo passaggio viene fatto utilizzando un apposito fattore di conversione che permette di ricavare, dalle velocità angolari, i corretti e corrispondenti valori di velocità da attuare alle due ruote indipendenti dell'agente, le quali, si ricorda, sono generate grazie al problema di ottimizzazione risolto dall'algoritmo esterno. È importante inoltre osservare che bisogna sempre tener conto delle varie limitazioni del *robot e-puck,* che sono modellizzate tramite delle saturazioni sulle variabili. Queste saturazioni ci consentono di evitare di incorrere in numerosi problemi, permettendoci di escludere che il *robot* lavori a velocità troppo elevate, che non consentirebbero di accettare l'approssimazione fatta in fase di modellazione, ovvero di trascurare la dinamica delle velocità e l'inerzia dell'agente, oltre che provocare problemi di *crash* negli algoritmi, o a velocità prossime a zero per le quali si potrebbero verificare i problemi analizzati nei paragrafi precedenti.

In sintesi, si può spiegare e riassumere in modo intuitivo l'algoritmo sviluppato tramite il diagramma di flusso rappresentato nella Figura 3.10 o, allontanandoci

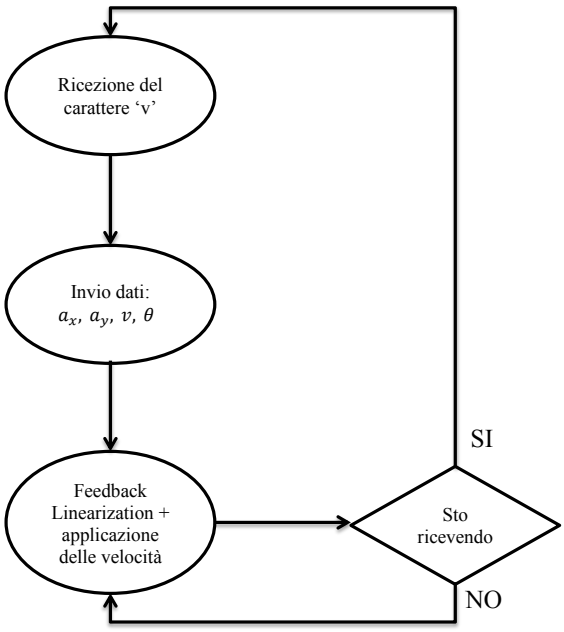


Figura 3.10: Diagramma ricezione dati da *MATLAB*

dal solo algoritmo interno all'agente, si può vedere il problema più esternamente attraverso lo schema a blocchi mostrato in Figura 3.11.

Notevole importanza assume quindi la corretta sincronizzazione tra l'ambiente esterno e il *robot*, quindi il suo anello di *Feedback Linearization* implementato internamente a esso. Questo problema trova soluzione grazie a un'esatta temporizzazione dei vari cicli utilizzati, ovvero la scelta dei tempi di campionamento deve essere concorde alla frequenza di *clock* interna all'agente che viene ricavata dai dati di targa. Come già accennato in precedenza, nella trattazione [13] l'anello di *Feedback Linearization* era posto esternamente all'agente, quindi era implementato direttamente tramite anelli annidati. Ora, l'anello esterno potrà avere un tempo di campionamento nettamente minore, perché non dovrà più risolvere il problema di *Feedback*

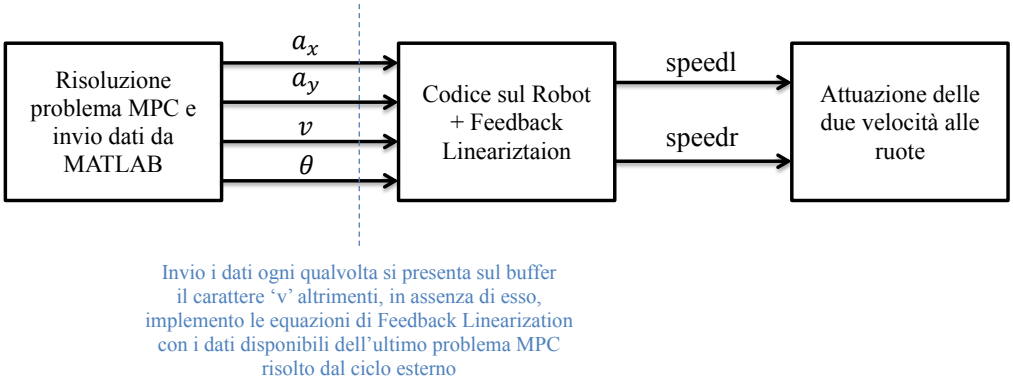


Figura 3.11: Schema a blocchi ricezione e invio dati

Linearization più quello di *MPC*, ma si dovrà occupare solamente di quest'ultimo. A tale proposito si è scelto un tempo di campionamento $\tau_{int} = 0.01$ per la *Feedback Linearization* interna al *robot*, mentre per l'anello esterno relativo al controllo predittivo si può raggiungere un valore di $\tau = 0.5$; molto minore rispetto a quello ricavato in [13], dove si utilizzava $\tau = 2.7$.

Fatta un'analisi dettagliata dell'anello linearizzante interno all'agente, nel prosieguo si allarga la panoramica valutando il problema discusso nel paragrafo precedente, ovvero sul fatto di aver costruito un osservatore di ordine intero e non ridotto. Come detto, la *webcam* acquisisce la posizione a ogni singolo ciclo di scansione e questa trasmissione di dati dalla telecamera al *computer* causa rallentamenti al sistema. Si è quindi inizialmente pensato di sfruttare nuovamente il predittore di *Kalman* in modo che, utilizzando i dati agli istanti passati, stimi la posizione, la velocità e calcoli successivamente l'orientamento del *robot* negli istanti futuri così che la telecamera acquisisca l'immagine con una frequenza relativamente bassa, mentre il sistema continua a funzionare con una frequenza di aggiornamento del controllo, grazie ai dati stimati, fino alla successiva acquisizione da *webcam*. Questa strategia che lavora con un predittore di ordine intero, necessita infatti di tutte le variabili di stato, posizioni e velocità, per implementare l'algoritmo, e permette di avere una minore dipendenza del sistema dal sensore servo visivo, ottenendo così vantaggi in termini di robustezza in caso di *fault* o nel caso in cui la telecamera non possa raggiungere una frequenza di campionamento sufficientemente elevata. Tuttavia questa tecnica causa problemi, in quanto la stima e il valore del sistema linearizzato, ricavati dalle simulazioni e mostrati nei grafici precedenti, non risultano essere perfettamente concordi, soprattutto nelle fasi di partenza e di arrivo dell'agente. In quei tratti infatti, a causa delle basse velocità, la stima si discosta molto dal sistema reale e col passare degli istanti di tempo questo fattore genera una somma di errori che causa di conseguenza un continuo allontanamento, a ogni acquisizione, del *robot* dalla posizione del suo *setpoint,* fino a mandare in *crash* l'algoritmo: ciò è dato dal fatto che avendo un continuo incremento dell'errore si viene a generare un comportamento instabile nel sistema. Si è quindi deciso di continuare ad acquisire tramite *webcam* la posizione e l'orientamento del *robot* a ogni istante di campionamento, visto che comunque il ritardo causato da questa trasmissione non risulta essere compromettente, utilizzando di fatto la stima dello stato dei soli valori di velocità e non di posizione.

3.4 Confronto sperimentale

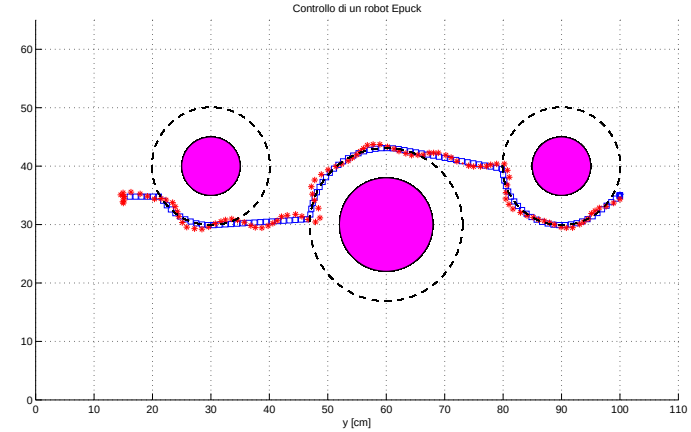
Si vogliono ora verificare sul sistema reale i miglioramenti ottenuti grazie alle varie innovazioni descritte nei precedenti paragrafi; a tale scopo si effettua un confronto sperimentale eseguendo inizialmente un *test* utilizzando un agente *e-puck* con struttura logica di controllo sequenziale, come proposto nella trattazione [13], e successivamente si ripete il *test*, mantenendo le stesse caratteristiche ambientali esterne, ma considerando ora il *robot* con la struttura innovativa che utilizza una logica in parallelo. Sottolineiamo che l'obiettivo di tale esperimento a questo livello non ha come scopo quello di soffermarsi sulla tecnica specifica di controllo o sugli algoritmi atti alla navigazione dei *robot* nel piano di lavoro, che saranno oggetto di future analisi nel prosieguo della tesi, ma vuole semplicemente dimostrare i forti vantaggi prestazionali, in termini di maggiore velocità e minori oscillazioni nell'inseguimento del segnale di riferimento, ottenuti grazie alle innovazioni proposte.

L'esperimento consiste nell'introdurre un agente *e-puck* nel piano di lavoro a disposizione, che ha dimensioni 115 x 66 cm^2, in cui sono inoltre presenti tre ostacoli, solo virtualmente implementati, posti in posizioni note a priori ovvero: $O1(60, 30, 8)$, $O2(90, 40, 5)$, $O3(30, 40, 5)$ dove i primi due termini dentro le parentesi tonde indicano le coordinate (x, y) del centro dell'ostacolo, che ricordiamo essere circolare, all'interno del piano di riferimento, mentre il terzo dato rappresenta il suo raggio. Il *robot*, in entrambi i casi, partirà dalla stessa posizione iniziale che presenta coordinate $(15, 34)$ e dovrà raggiungere il medesimo *goal* alle coordinate $(100, 35)$. Inoltre durante i *test* vengono utilizzati e mantenuti costanti i seguenti parametri:

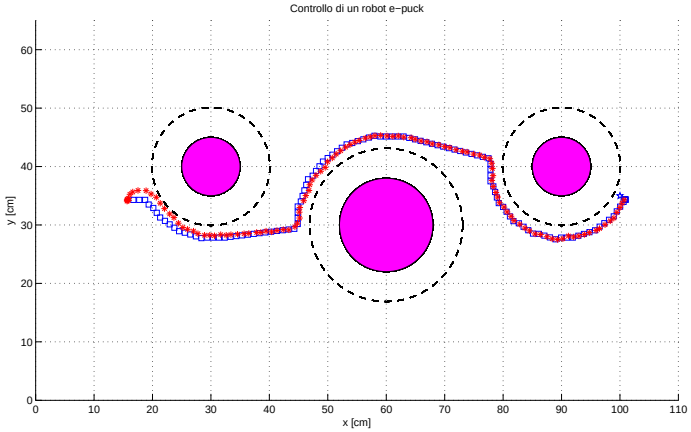
- struttura sequenziale: $\tau = 2.7$, Q e R matrici di peso dello stato e dell'ingresso utilizzate nel paragrafo 3.2 per la costruzione del predittore di *Kalman* sono state scelte in modo da essere matrici identità;
- struttura in parallelo: $\tau = 0.3$ (come detto è possibile abbassare di molto il tempo di computazione dell'anello esterno per risolvere il problema predittivo), Q e R matrici identità.

Utilizzando ora le seguenti figure andiamo a effettuare il confronto tra le due logiche, struttura sequenziale e in parallelo, e utilizziamo appunto esse per spiegare e dimostrare i vantaggi precedentemente enunciati; in primo luogo si analizza come la posizione reale dell'agente insegue il riferimento.

Nella Figura 3.12 viene rappresentato, attraverso quadratini blu, un intorno della traiettoria di riferimento in cui dovrebbe trovarsi l'agente reale. Avendo che i due algoritmi operano, come detto, a due tempi di campionamento differenti questo implica che il numero di campioni che rappresentano la posizione del *robot* acquisita dalla *webcam,* preso in considerazione un determinato spazio costante, risulta essere differente. Come si vede infatti dalla Figura 3.12 (a) l'algoritmo a logica sequenziale, che opera con un tempo di campionamento τ elevato, produce un segnale di posizione che presenta campioni leggermente distanti l'uno dall'altro e tale distanza è circa costante ed è appunto dettata dal tempo di campionamento con cui opera l'algoritmo; da notare che in questo caso la posizione dell'agente non resta sempre contenuta nell'intorno della traiettoria di riferimento. Nella Figura 3.12 (b), riferita a una struttura logica in parallelo, il tempo di campionamento τ utilizzato è minore e questo fa sì che i campioni risultino essere molto vicini tra loro dando origine a un segnale più fitto rispetto al caso precedente. Quanto detto porta a concludere



(a)



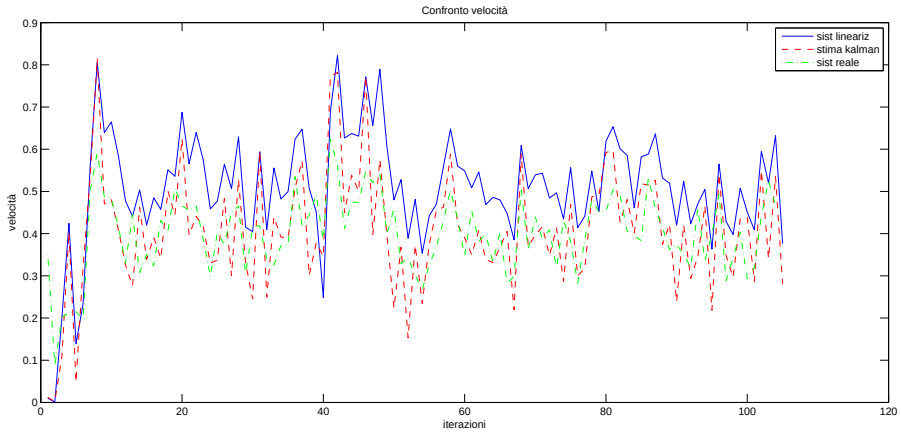
(b)

Figura 3.12: Posizione e riferimento del *robot* che utilizza una struttura a logica sequenziale (a) e una struttura a logica in parallelo (b)

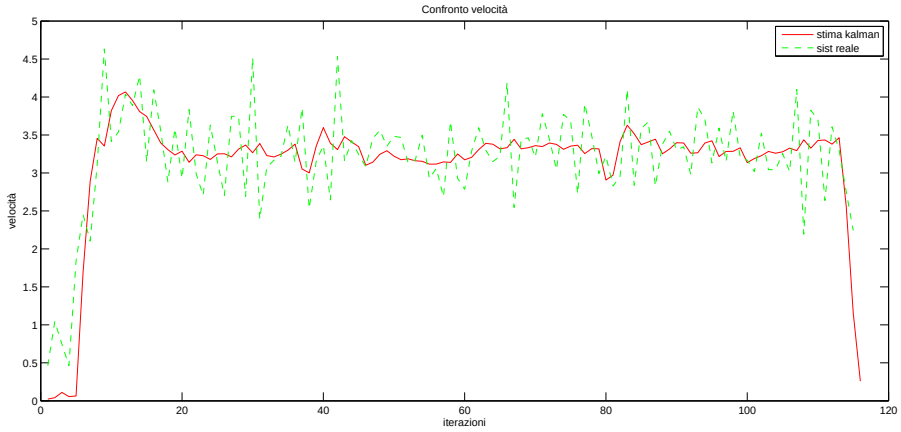
che, nel caso in cui è implementata una struttura logica sequenziale si ha un segnale meno continuo in quanto esso presenta un numero leggermente minore di campioni. Il controllore potrà quindi intervenire un numero limitato di volte, corrispondente al numero di campioni, per mantenere la traiettoria del *robot* all'interno del riferimento; di conseguenza l'azione di controllo applicata sarà di forte entità e quindi poco moderata. Nel caso in cui venga invece implementata la struttura logica in parallelo il segnale di posizione risulta essere molto più continuo e il controllore potrà in questo caso intervenire un numero di volte maggiore per indirizzare al meglio, e con una azione di controllo più moderata, la traiettoria del *robot* che infatti resta sempre confinata all'interno del suo riferimento. Quanto detto porta inoltre ad avere nel segnale di posizione della struttura logica sequenziale, Figura 3.12 (a), delle leggere oscillazioni il che si traduce semplicemente in una minor robustezza e prontezza del controllo nel guidare il *robot* verso la sua traiettoria di riferimento. Questo come spiegato non avviene invece nel caso in cui si utilizzi la struttura con logica in parallelo dove, grazie al fatto che si è riusciti ad abbassare notevolmente il tempo di campionamento τ con cui opera l'algoritmo, si è ottenuto un controllo molto più

performante in termini di tempi di reazione dell'azione di controllo.

Dalla Figura 3.13 è invece possibile analizzare i guadagni ottenuti in termini di



(a)



(b)

Figura 3.13: Andamento delle velocità per il *robot* che utilizza una struttura a logica sequenziale (a) e una struttura a logica in parallelo (b)

velocità da parte dell'agente; infatti è ben visibile, guardando il grafico in Figura 3.13 (b), riferito a una struttura logica in parallelo, come si abbiano sull'asse delle ordinate dei valori di velocità dell'agente maggiori, fino a circa cinque volte, rispetto al grafico in Figura 3.13 (a), riferito alla struttura sequenziale. Questo si traduce in un tempo ridotto nel raggiungimento dell'obiettivo finale da parte del *robot:* si passa infatti da 152 secondi, relativi alla struttura logica sequenziale, a 16 secondi per quella in parallelo. Inoltre la stima della velocità, portando internamente al *robot* l'anello linearizzante, risulta essere più assestata a un valore costante subendo così meno variazioni e avendo come conseguenza un migliore e più semplice controllo sull'agente, che risulta avere una dinamica meno aggressiva e più regolare. È fondamentale ribadire che, in realtà, la velocità del sistema reale non può essere ricavata in quanto, come detto precedentemente, essa non è misurabile; per velocità del sistema reale, ovvero quella mostrata nella Figura 3.13, si intende infatti la derivata discreta della

posizione misurata tramite *webcam* rispetto a due punti successivi, utilizzando la semplice formulazione (3.4).

In conclusione, dopo aver analizzato le Figure 3.12 e 3.13, è possibile comprendere appieno i grandi vantaggi in termini di robustezza e stabilità dell'agente ed è quindi corretto affermare che la struttura logica in parallelo con anello linearizzante interno al *robot* ci porta a ottenere grandi miglioramenti delle prestazioni del sistema controllato.

Capitolo 4

Controllo Predittivo

4.1 Introduzione

A partire dall'inizio degli anni '80 il controllo ottimo di tipo *model predictive control* (*MPC*) è diventato una delle metodologie più utilizzate per la soluzione di problemi di controllo multivariabile, soprattutto in ambito industriale, in quanto i processi analizzati presentano dinamiche relativamente lente. Esso consente di gestire vincoli rigidi sul sistema, forzando le variabili a rimanere all'interno di determinati *range* stabiliti. Inoltre è possibile definire molteplici e specifici obiettivi, che possono essere anche contrastanti tra loro. In particolare, la tecnica *MPC* consiste nel determinare il valore degli ingressi di controllo mediante l'ottimizzazione *on-line* di una cifra di costo costruita sulla predizione del comportamento del sistema in un certo orizzonte futuro, ottenuta sfruttando il modello del sistema stesso. Il risultato del problema di ottimizzazione è quindi una sequenza di controllo ottima per gli istanti successivi, della quale solo il primo elemento viene applicato realmente al sistema controllato. All'istante di campionamento successivo l'orizzonte predittivo è traslato in avanti, e il problema di controllo ottimo su orizzonte finito è risolto nuovamente sulla base di una nuova misura dello stato reale. Tale tecnica, denominata *Receding Horizon Control* e illustrata per un sistema *SISO* in Figura 4.1 , è descritta in modo specifico in [18] e viene solitamente considerata implicitamente quando si parla di controllo *MPC*. Il risultato dell'ottimizzazione è quindi la sequenza ottima di ingressi da applicare al sistema lungo tutto l'orizzonte.

All'istante k il problema di controllo ottimo viene risolto in base allo stato iniziale x_k lungo un orizzonte predittivo di lunghezza N fissata. Come visto viene quindi generata una sequenza di ingressi ottimi del tipo:

$$U_k^o = \{u_k, u_{k+1}, \ldots, u_{k+N-1}\} \tag{4.1}$$

il cui primo elemento viene applicato al sistema reale per tutta la durata del periodo di campionamento. All'istante successivo $(k+1)$ l'orizzonte predittivo viene traslato in avanti di un passo e, quando una nuova misura dello stato x_{k+1} è disponibile, il problema di controllo ottimo è risolto nuovamente con le informazioni aggiornate. Iterando il procedimento si ricava quindi a ogni istante k la sequenza di ingressi ottimi (4.1).

Il controllo *MPC* è caratterizzato da buone proprietà di robustezza a fronte di errori di modellazione e di disturbi. Questo è dovuto al fatto che tale tecnica introduce un meccanismo di *feedback* nel sistema, il cui controllo diventa quindi di tipo

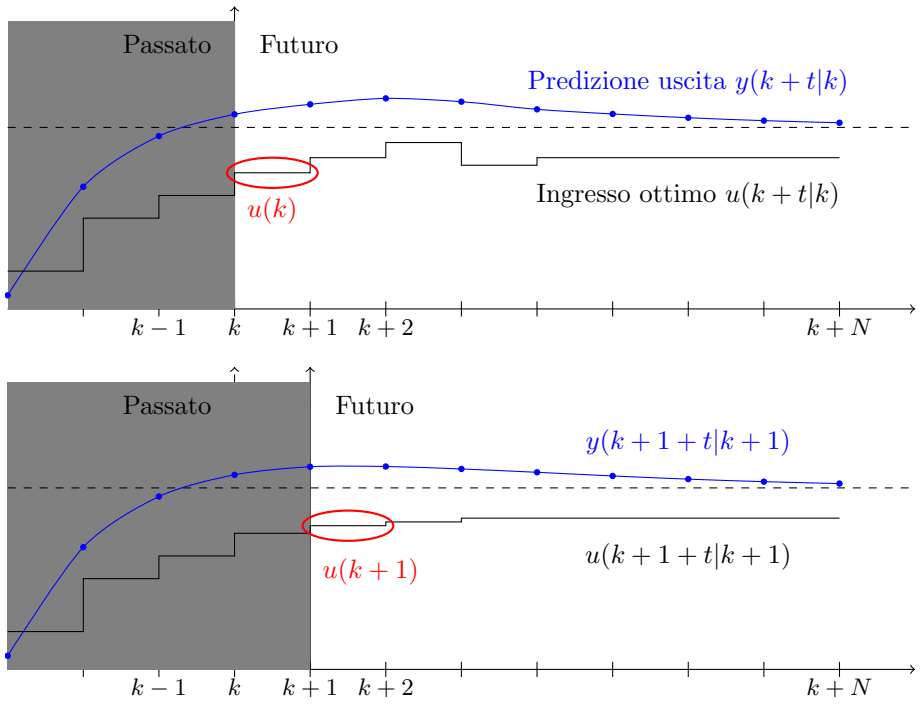


Figura 4.1: Tecnica *MPC* con *Receding Horizon*

closed-loop. Il motivo della popolarità di *MPC* è dato dal fatto che esso è in grado di garantire ottimalità rispettando contemporaneamente vincoli rigidi sul valore assunto dagli stati del sistema e dalle variabili di controllo. Una delle principali limitazioni di *MPC*, tuttavia, sta nella sua complessità computazionale, soprattutto se comparato con controllori di tipo classico, come per esempio i controllori *PID*; sarà quindi di fondamentale importanza che esso venga applicato a un sistema semplice e possibilmente lineare.

4.2 Tecnica MPC per sistemi lineari

In accordo con il nostro problema specifico si consideri ora un generico sistema lineare a tempo discreto:

$$\begin{cases} x_{k+1} = & Ax_k + Bu_k \\ y_k = & Cx_k \end{cases} \tag{4.2}$$

in cui $x_k \in \mathbb{R}^n, u_k \in \mathbb{R}^m, y_k \in \mathbb{R}^p$ rappresentano rispettivamente lo stato, l'ingresso e l'uscita. Al generico istante k si vuole quindi trovare la sequenza di ingressi ottimi (4.1), per un fissato orizzonte di predizione N, che minimizzi la cifra di merito:

$$V^N(x_k, U_k) = \sum_{i=0}^{N-1} l(x_{k+i}, u_{k+i}) + V^f(x_{k+N})$$

Tale funzione di costo è composta da due termini:

- $l(\cdot,\cdot)$ è una funzione dello stato e dell'ingresso definita positiva. La sua scelta solitamente ricade su una funzione quadratica del tipo:

$$l(x_k, u_k) = \|x_k\|_Q^2 + \|u_k\|_R^2 = x_k^T Q x_k + u_k^T R u_k$$

  dove le due componenti rappresentano il quadrato della norma euclidea di ciascun termine, pesata attraverso due matrici definite positive, Q e R, applicate rispettivamente sullo stato e sul controllo.

- V^f rappresenta invece un peso terminale sullo stato alla fine dell'orizzonte di predizione; essa solitamente è scelta nella forma:

$$V^f(x_{k+N}) = \|x_{k+N}\|_P^2 = x_{k+N}^T P x_{k+N}$$

  con P che rappresenta la matrice di peso dello stato finale anch'essa definita positiva.

Il problema di ottimizzazione può essere inoltre sottoposto a vincoli del tipo:

$$x_k \in \mathbb{X},\ u_k \in \mathbb{U} \tag{4.3}$$

in cui $\mathbb{X}$ e $\mathbb{U}$ sono due insiemi convessi e contenenti l'origine che rappresentano i domini, quindi tutti i possibili valori assumibili rispettivamente dalle variabili di stato e d'ingresso del *robot*. Inoltre si richiede che lo stato alla fine dell'orizzonte di predizione appartenga a un determinato *terminal set* $\mathbb{X}^f$, anch'esso centrato nell'origine e contenuto in $\mathbb{X}$:

$$x_{k+N} \in \mathbb{X}^f \subseteq \mathbb{X}$$

Descritte le varie componenti che definiscono il problema *MPC* e applicando ora, come visto in precedenza, la tecnica *Receding Horizon,* all'istante k si risolve il problema di ottimizzazione e si fornisce al sistema l'ingresso $u_{k|k}$ trovato che sarà il primo elemento della sequenza ottima U_k^o. Al passo successivo $k+1$, si risolve di nuovo il problema applicando, questa volta, l'ingresso $u_{k+1|k+1}$, primo elemento di U_{k+1}^o. Iterando il procedimento si ottiene così una legge di controllo implicita del tipo:

$$u_k = K_{MPC}(x_k)$$

ricavando così l'intera sequenza di ingressi ottimi U_k^o.

4.2.1 Condizioni per la stabilità MPC

Quello che ci si può chiedere a questo punto è se il sistema ottenuto sia stabile o meno, e se, al passo successivo, il problema di controllo predittivo risulti ancora ben posto. Per poter valutare le condizioni di stabilità del controllo *MPC* con *RH*, si introduce il concetto di *set* Positivamente Invariante. In particolare un *set* Ω è detto positivamente invariante per il sistema autonomo

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

se:

$$x_{\bar{k}} \in \Omega \Rightarrow x_k \in \Omega, \forall k \geq \bar{k}$$

Ovvero si richiede che se in un generico istante $\bar{k}$ lo stato è contenuto in Ω, allora la soluzione x_k resterà confinata in Ω anche per tutti i successivi istanti $k \geq \bar{k}$.

Si definisca la legge di controllo ausiliaria per il sistema (4.2) del tipo:

$$u_k = Kx_k$$

e sia $\mathbb{X}^f$ un insieme positivamente invariante per il sistema in anello chiuso

$$x_{k+1} = Ax_k + BKx_k$$

Si noti che, nel caso siano presenti vincoli (4.3) di ammissibilità delle variabili, è necessario che $\mathbb{X}^f \subseteq \mathbb{X}$ e che $Kx \in \mathbb{U} \forall x \in \mathbb{X}^f$. Infine, il peso terminale viene scelto in modo tale da essere una funzione di *Lyapunov* per il sistema controllato, cioè tale che:

$$V^f(Ax_k + BKx_k) \leq V^f(x_k) - l(x_k, Kx_k), \forall x_k \in \mathbb{X}^f \tag{4.4}$$

In questo modo, grazie al criterio di *Lyapunov,* è possibile garantire la stabilità del controllo *MPC*; per una trattazione più completa e approfondita si rimanda a [2].

4.2.2 Scelte dei parametri progettuali

Al fine di definire i parametri del nostro problema di controllo, si ha ora il compito di scegliere il peso terminale sullo stato impostando la funzione V^f e il *set* $\mathbb{X}^f$ a cui lo stato dovrà appartenere alla fine dell'orizzonte di predizione. Esistono molti metodi presentati in letteratura per la scelta di tali parametri progettuali, nella nostra trattazione la scelta ricade sulla tecnica *Quasi-infinite Horizon MPC*. Semplicemente si andrà a determinare il valore di V^f così da fare in modo che l'equazione (4.4) sia verificata; inoltre si ricava una soluzione P dell'equazione algebrica di Riccati, nota K scelta arbitrariamente e note le matrici simmetriche $Q \geq 0$ e $R > 0$, utilizzando l'equazione di *Lyapunov*:

$$(A + BK)^T P (A + BK) - P = -\left(Q + K^T R K\right) \tag{4.5}$$

In questo modo è possibile ricavare il *terminal set* $\mathbb{X}^f$ come:

$$\mathbb{X}^f = \left\{x \colon x^T P x \leq \alpha\right\} \subseteq \mathbb{X}, \alpha \geq 0$$

in cui il *set* viene preso in modo tale che $\mathbb{X}^f \subseteq \mathbb{X}$. Si può quindi concludere che questa scelta dei parametri garantisce la stabilità del sistema controllato; per maggiori dettagli si rimanda a [2].

4.3 Tube Based MPC

Dopo aver descritto la tecnica di controllo *MPC* torniamo a considerare il problema specifico del controllo di agenti *e-puck*. Fino a ora abbiamo visto come sia possibile descrivere il *robot* mobile come un modello a uniciclo e come si possa costruire una legge linearizzante in grado di semplificare, all'esterno, il problema di controllo. Abbiamo poi osservato come, in questa situazione, sia possibile scegliere gli ingressi di accelerazione a_x e a_y del sistema linearizzato a partire dai riferimenti di posizione x_r e y_r e poi applicare il tutto tramite un *feedback* dinamico al sistema reale. Tale approccio costringe tuttavia a progettare un ulteriore sistema di controllo

per la pianificazione della traiettoria da imporre al *robot* in quanto si ha bisogno di un'elevata robustezza per assicurare che l'agente segua in modo corretto il riferimento di posizione. In realtà sarebbe desiderabile unificare i due problemi e risolvere contemporaneamente il problema di stabilizzazione e quello di pianificazione, così da avere un sistema in grado di agire completamente in tempo reale senza bisogno di progettare alcun percorso, ma semplicemente avvicinandosi all'obiettivo. Una possibile soluzione è allora quella di applicare direttamente un controllo di tipo *MPC* che, come abbiamo visto, ha il vantaggio di poter gestire esplicitamente dei vincoli operativi sul problema. Inoltre, alla luce dell'obiettivo finale di coordinamento di una flotta di *robot*, la soluzione predittiva consente di avere a disposizione il vettore delle azioni future lungo l'orizzonte scelto, che potrà essere utilizzato in un contesto di tipo distribuito come verrà analizzato nel prossimo paragrafo.

Generalmente le classiche tecniche di controllo predittivo introdotte non sono esplicitamente progettate per garantire robustezza a fronte di incertezze del sistema controllato o di disturbi esterni agenti su di esso. Infatti, non è possibile trarre conclusioni riguardo alla stabilità nel caso in cui la dinamica del sistema reale si discosti da quella predetta. Tuttavia questo sarebbe proprio quello di cui si necessita per il problema in esame e per molti casi reali dove incertezze e disturbi sono sempre presenti a causa, ad esempio, di errori di modello, errori di misura ed errori di attuazione. Inoltre, nel caso specifico, la misura della posizione è affetta da imprecisioni dovute sia alla distorsione della *webcam* che all'algoritmo di elaborazione dell'immagine. Avendo quindi a che fare con un sistema su cui agiscono disturbi si vuole che esso risulti il più possibile insensibile a essi tramite una soluzione controllistica che garantisca elevata robustezza rispetto alle problematiche evidenziate.

A fronte dei problemi visti viene in nostro aiuto la tecnica conosciuta con il nome *Tube-based MPC* [19], la quale fa sì che lo stato del sistema controllato resti all'interno di un insieme confinato nell'intorno dello stato nominale anche a fronte di disturbi limitati. Il nome di questa tecnica deriva appunto dal fatto che la traiettoria reale dello stato si trova sempre all'interno di una regione tubolare limitata centrata sulla traiettoria nominale, come rappresentato nella Figura 4.2 per un sistema del secondo ordine. Inoltre nel nostro problema, si ricorda che, oltre a garantire robustezza, è anche necessario soddisfare eventuali vincoli su stato e ingressi e questa tecnica, facendo parte dell'insieme dei metodi di controllo *MPC*, ci permette anche di fare questo.

Nel controllo *MPC Tube-based* si considera quindi un sistema lineare:

$$x_{k+1} = Ax_k + Bu_k + w_k,\ w_k \in \mathbb{W} \tag{4.6}$$

in cui w_k è un disturbo agente sul sistema e $\mathbb{W}$ è un insieme compatto contenente l'origine, noto a priori. L'obiettivo è quindi quello di trovare una legge di controllo che garantisca convergenza e ottimalità delle variabili di stato, nel rispetto dei vincoli, indipendentemente dal valore assunto da tale disturbo, purché contenuto nel *set* $\mathbb{W}$. L'idea di questa soluzione di controllo robusto è di utilizzare un sistema nominale, come riferimento, in parallelo al sistema reale su cui verrà effettivamente applicato il controllo *MPC* classico, che garantirà la robustezza desiderata mediante vincoli più stringenti nel problema di ottimizzazione.

Considerando il sistema nominale associato a (4.6):

$$\hat{x}_{k+1} = A\hat{x}_k + B\hat{u}_k \tag{4.7}$$

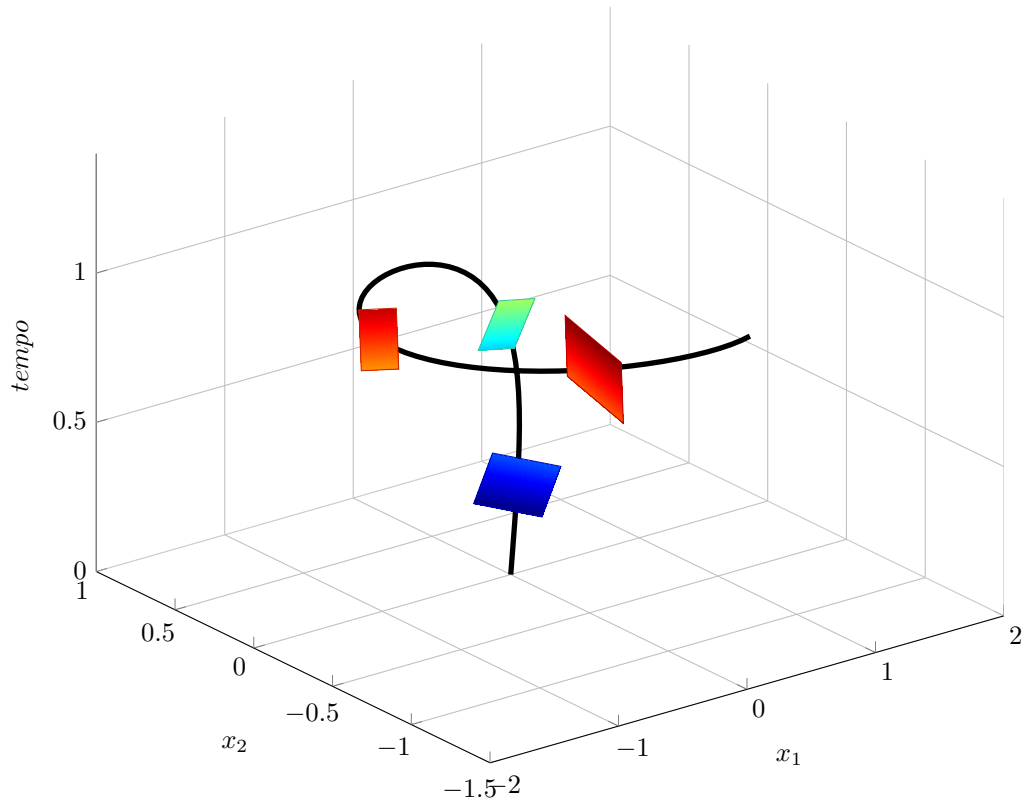


Figura 4.2: Evoluzione della traiettoria dello stato con *Tube-based MPC*

il problema può essere riformulato come quello di trovare un ingresso u_k per il sistema reale a partire dall'ingresso ottimo per il sistema nominale che mantenga lo stato reale x_k all'interno di un ben definito *set* $\mathbb{X}$, indipendentemente dal valore assunto dal disturbo w_k purché contenuto in $\mathbb{W}$. Nello specifico, l'ingresso robusto applicato al sistema reale viene ottenuto come mostrato nella seguente formulazione:

$$u_k = \hat{u}_k + K\left(x_k - \hat{x}_k\right) \tag{4.8}$$

in cui la quantità rappresentata tra parentesi nella (4.8) definisce un termine di *feedback* che tiene conto della dinamica dell'errore che esiste tra lo stato reale e quello nominale.

Sottraendo quindi l'equazione del sistema reale (4.6) a quella del sistema nominale (4.7) si ottiene:

$$x_{k+1} - \hat{x}_{k+1} = A\left(x_k - \hat{x}_k\right) + B\left(u_k - \hat{u}_k\right) + w_k$$

e sostituendo ora la (4.8) si ricava:

$$x_{k+1} - \hat{x}_{k+1} = A\left(x_k - \hat{x}_k\right) + BK\left(x_k - \hat{x}_k\right) + w_k$$

Definendo inoltre, come detto, la dinamica dell'errore $z_k = x_k - \hat{x}_k$ si ha:

$$z_{k+1} = \left(A + BK\right) z_k + w_k$$

Si può quindi affermare che, se $(A + BK)$ ha tutti gli autovalori confinati nella regione di stabilità, allora esiste un insieme robustamente positivamente invariante

$\mathbb{Z}$ tale per cui:

$$z_k \in \mathbb{Z},\, w_t \in \mathbb{W},\, \forall t \geq k \Rightarrow z_{k+h} \in \mathbb{Z},\, \forall h \geq 0$$

Con la terminologia "robustamente positivamente" invariante si indica che

$$z_{k+h} \in \mathbb{Z},\, h > 0$$

indipendentemente dall'entità del disturbo $w_k \in \mathbb{W}$, purché $z_k \in \mathbb{Z}$. In altre parole, se lo stato iniziale è tale che $z_0 \in \mathbb{Z}$, allora, in accordo con la definizione di *set* robustamente positivamente invariante, z_k rimarrà sempre all'interno di $\mathbb{Z}$ indipendentemente dall'entità del disturbo w_k. Tele proprietà si traduce formalmente come segue[1]:

$$(A + BK)\,\mathbb{Z} \oplus \mathbb{W} \subseteq \mathbb{Z}$$

Questo risultato garantisce che lo stato del sistema reale sia sempre contenuto in un intorno ben definito dello stato del sistema nominale.

Consideriamo ora i vincoli sullo stato e sull'ingresso del sistema reale:

$$x_k \in \mathbb{X},\, u_k \in \mathbb{U} \tag{4.9}$$

poiché il problema di ottimizzazione è risolto sul sistema nominale, è necessario trasformare questi vincoli sul sistema reale e quindi sulle relative variabili di stato e di ingresso. I nuovi vincoli dovranno essere tali da garantire che (4.9) siano rispettati indipendentemente dal valore assunto dal disturbo w_k risultando quindi più stringenti rispetto a quelli imposti precedentemente. I nuovi vincoli verranno infatti definiti nel seguente modo:

$$\begin{aligned} \hat{x}_k = x_k - z_k \quad &\Rightarrow \quad \hat{x}_k \in \hat{\mathbb{X}} = \mathbb{X} \ominus \mathbb{Z} \\ \hat{u}_k = u_k - K z_k \quad &\Rightarrow \quad \hat{u}_k \in \hat{\mathbb{U}} = \mathbb{U} \ominus K\mathbb{Z} \end{aligned}$$

Si noti che tali vincoli hanno senso solo se lo stato nominale all'istante iniziale k è tale che la dinamica dell'errore $z_k \in \mathbb{Z}$, altrimenti non c'è nessuna garanzia che z resti confinata in $\mathbb{Z}$. A tale proposito si potrebbe scegliere $\hat{x}_0 = x_0$, che chiaramente soddisfa quanto detto; successivamente si lascerà poi che $\hat{x}_k$ evolva autonomamente secondo la legge (4.6).

Per concludere, prima di definire il problema di ottimizzazione, si dovranno considerare i vincoli terminali. Si definisce il *set* terminale $\hat{\mathbb{X}}^f \subseteq \hat{\mathbb{X}}$ positivamente invariante per il sistema nominale in anello chiuso a cui è applicata la legge di controllo ausiliaria

$$\hat{u}_k = K\hat{x}_k$$

in modo tale che l'ingresso sia ammissibile nel rispetto dei vincoli precedentemente definiti, ovvero che:

$$\hat{x}_{\bar{k}} \in \hat{\mathbb{X}}^f \Rightarrow \begin{cases} \hat{x}_{k+1} = & (A + BK)\,\hat{x}_k \in \hat{\mathbb{X}}^f \\ \hat{u}_k = & K\hat{x}_k \in \hat{\mathbb{U}} \end{cases}, \forall k \geq \bar{k}$$

[1] Il simbolo $\oplus$ indica la somma di Minkowsi tra due insiemi, definita nel seguente modo: $A \oplus B = \bigcup_{b \in B} \{a + b \mid a \in A\}$. Analogamente, il simbolo $\ominus$ indica la differenza di Pontryagin definita come: $C = A \ominus B = \{c\colon c \oplus B \subseteq A\}$.

A questo punto si può definire il problema di ottimizzazione soggetto ai vincoli appena ricavati:

$$\begin{gathered}\min_{\hat{u}(k:\,k+N-1)} \sum_{h=0}^{N-1} \| \hat{x}_{k+h} \|_Q^2 + \| \hat{u}_{k+h} \|_R^2 + \| \hat{x}_{k+N} \|_P^2 \\ \hat{x}_{k+1} = A\hat{x}_k + B\hat{u}_k \\ \hat{x}_{k+h} \in \hat{\mathbb{X}}, h = 0, \ldots, N-1 \\ \hat{u}_{k+h} \in \hat{\mathbb{U}}, h = 0, \ldots, N-1 \\ \hat{x}_{k+N} \in \hat{\mathbb{X}}^f\end{gathered}$$

Tornando alla scelta dello stato iniziale del sistema nominale, si ricorda che è sufficiente garantire che $z_k \in \mathbb{Z}$ per ottenere una traiettoria dello stato reale tale che:

$$x_{k+h} \in \hat{x}_{k+h} \oplus \mathbb{Z}$$

Il problema di ottimizzazione può essere perciò riformulato come segue:

$$\begin{gathered}\min_{\hat{x}(k),\hat{u}(k:\,k+N-1)} \sum_{h=0}^{N-1} \| \hat{x}_{k+h} \|_Q^2 + \| \hat{u}_{k+h} \|_R^2 + \| \hat{x}_{k+N} \|_P^2 \\ \hat{x}_{k+1} = A\hat{x}_k + B\hat{u}_k \\ \hat{x}_{k+h} \in \hat{\mathbb{X}}, h = 0, \ldots, N-1 \\ \hat{u}_{k+h} \in \hat{\mathbb{U}}, h = 0, \ldots, N-1 \\ \hat{x}_{k+N} \in \hat{\mathbb{X}}^f \\ x_k - \hat{x}_k \in \mathbb{Z}\end{gathered} \tag{4.10}$$

Infine, utilizzando il principio *RH*, l'ingresso applicato al sistema reale all'istante k è:

$$u_k = \hat{u}_{k|k} + K\left(x_k - \hat{x}_{k|k}\right)$$

in cui $\hat{u}_{k|k}$ e $\hat{x}_{k|k}$ sono ottenuti grazie all'ottimizzazione del problema (4.10) a uno specifico istante k.

Si è quindi visto che è possibile, formulando il problema in modo corretto, mantenere lo stato del sistema reale sempre contenuto in un intorno prefissato dello stato del sistema nominale. Ciò ci consente di avere la certezza che, applicando tale controllo al nostro sistema, l'agente si trovi sempre all'interno di una determinata regione, centrata nella traiettoria nominale di riferimento e di dimensione scelta a priori, indipendentemente dai disturbi applicati al sistema purché contenuti all'interno del *set* $\mathbb{W}$, e questo inoltre indipendentemente dallo stato iniziale in cui è posizionato il *robot*. In questo modo si potrà quindi stabilire con precisione una regione di incertezza nella quale si troverà la traiettoria delle uscite dell'agente rispetto alla traiettoria di riferimento, dato che anche l'errore tra sistema reale e nominale sarà contenuto in un determinato intorno. Vedremo poi nel prosieguo della tesi che tale garanzia sarà molto utile per risolvere in maniera distribuita il problema di coordinamento di più agenti e inoltre, sfruttando sempre il concetto di predizione, costituirà un'efficace soluzione ai problemi di *obstacole* e *collision avoidance*. La tecnica appena descritta è implementata utilizzando lo schema di controllo mostrato nella Figura 4.3.

4.4 Controllo predittivo distribuito (DPC) per l'inseguimento di una data traiettoria

Si vuole ora considerare il problema di controllare una flotta di *robot e-puck* in modo tale che le loro uscite inseguano un dato riferimento, garantendo robustezza

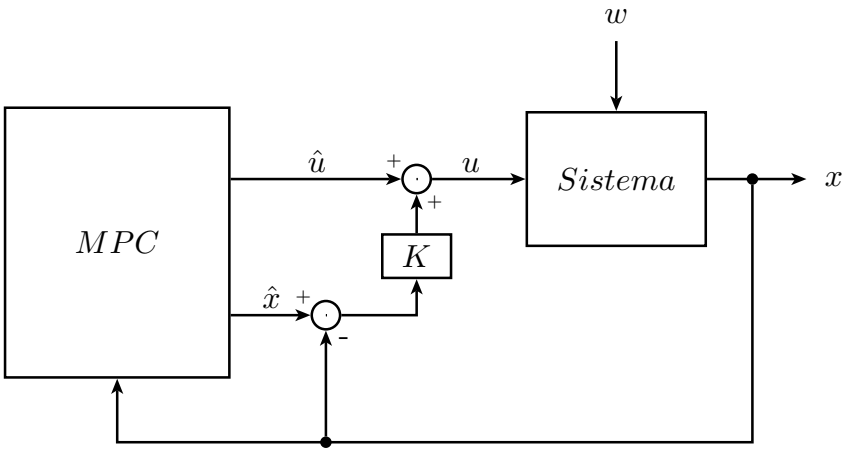


Figura 4.3: Schema di controllo *Tube-based MPC*

a fronte di incertezze e di disturbi che agiscono sul sistema reale usando tecniche di controllo predittivo. A tale proposito si considera un *set* composto da più *robot* come un unico sistema e si risolve il problema di controllo, non in modo centralizzato, ma attraverso un approccio di tipo distribuito. Questo metodo può essere usato in quanto ogni *robot* costituisce un sottosistema indipendente, interconnesso agli altri tramite dei vincoli, dove gli unici fattori rilevanti per il sistema complessivo, che costringono a considerare per intero il problema di controllo, sono la possibilità di collisione dei vari agenti durante il loro movimento e il problema di controllo di formazione. Quest'ultimo problema esula dagli scopi di questa tesi e quindi non verrà trattato; tuttavia per ulteriori informazioni è possibile consultare [13]. Rimane quindi da occuparsi del problema di collisione tra i vari agenti che, di fatto, viene risolto grazie al controllo *MPC* che fornisce, come visto, una predizione futura dell'uscita del sistema per i successivi N passi che, nel contesto corrente, corrisponde alla traiettoria di ciascun agente; in questo modo è possibile informare gli altri *robot* della propria futura posizione, riuscendo così a ottenere una generazione della traiettoria priva di collisioni. Il principale vantaggio dell'approccio distribuito è di tipo computazionale, esso infatti garantisce una maggior efficienza risolutiva. Di fatto un unico problema complesso viene semplificato e scomposto in tanti piccoli sotto-problemi che interagiscono dinamicamente tra loro. In questo modo viene ridotta la quantità complessiva di dati scambiati tra gli agenti, poiché ogni *robot* può richiedere informazioni solo circa i suoi vicini decentralizzando così l'intera struttura.

In questa trattazione per il controllo e il coordinamento dei nostri agenti facciamo riferimento a un'architettura gerarchica *DPC* (*Distributed Predictive Control)*, il cui schema di controllo, sviluppato dettagliatamente in [21], è in grado di unire la tecnica di controllo predittivo *Tube-based MPC*, presentata nel paragrafo 4.3, a una struttura distribuita, garantendo così un buon controllo del sistema complessivo anche a fronte di disturbi o di incertezze.

Considerando quindi un sistema di M agenti, dove ciascun agente è indicato con l'indice $i = 1, \ldots, M$, l'algoritmo *DPC*, presentato in Figura 4.4, opera su tre livelli distinti:

1. generazione della traiettoria di riferimento dell'uscita;

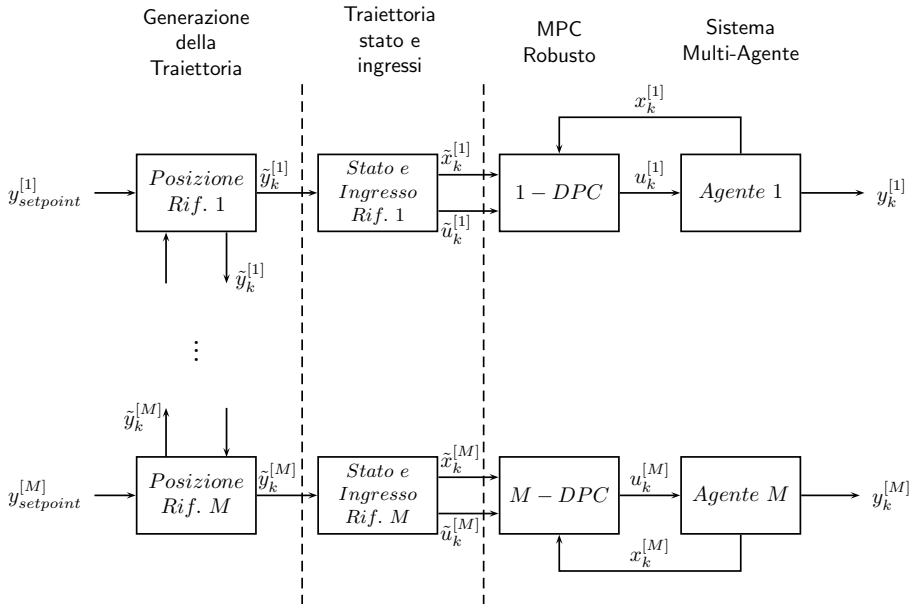


Figura 4.4: Schema di controllo distribuito per il coordinamento di più agenti

2. generazione delle traiettorie di riferimento per gli ingressi e per le variabili di stato;

3. inseguimento degli ingressi e delle variabili di stato (*MPC* robusto).

Andiamo ora a descrivere nel dettaglio i tre livelli enunciati sui quali agisce, come detto, la nostra struttura di controllo *DPC*, concentrandoci successivamente su come sono stati ricavati i *set* e i vari parametri utilizzati per il controllo e su cosa essi rappresentano.

4.4.1 Generazione della traiettoria di riferimento dell'uscita

Il compito di questo livello è quello di fornire una traiettoria di riferimento per l'uscita del nostro sistema, intesa come una successione di posizioni, che ogni *robot* i dovrà inseguire per ogni istante k detta $\left\{\tilde{y}^{[i]}_k, \ldots, \tilde{y}^{[i]}_{k+N-1}\right\}$. Questo livello adotta una struttura di tipo distribuito in quanto è necessaria una trasmissione di informazioni tra i controllori dei differenti agenti. Più nello specifico, ogni *robot*, a ogni istante k, una volta che è stata generata la propria traiettoria di riferimento dell'uscita, richiede anche la traiettoria di riferimento dell'uscita degli agenti vicini in modo da garantire la risoluzione di problemi di *collision avoidance* o di controllo di formazione tra i vari *robot*. Questo concetto verrà ulteriormente chiarito nel seguito della trattazione.

La principale proprietà che si vuole richiamare a questo punto, garantita da questo livello per ogni istante k, è descritta dalla seguente condizione[2]:

$$\tilde{y}^{[i]}_{k+1} \in \tilde{y}^{[i]}_k \oplus \beta^{(2)}_{q,\varepsilon_i}(0) \tag{4.11}$$

in questo modo noi richiediamo che la distanza tra due successivi punti della traiettoria di riferimento dell'uscita di ogni *robot* sia limitata, e che tale limite sia di dimensioni note.

[2]Una generica sfera di norma p centrata nell'origine nello spazio $\mathbb{R}^{dim}$ è definita come: $\beta^{(dim)}_{q,\varepsilon}(0) := \left\{x \in \mathbb{R}^{dim} : \| x \|_q \leq \varepsilon\right\}$.

4.4.2 Generazione delle traiettorie di riferimento per gli ingressi e le variabili di stato

Data la traiettoria di riferimento dell'uscita $\left\{\tilde{y}_k^{[i]}, \ldots, \tilde{y}_{k+N-1}^{[i]}\right\}$, calcolata e fornita dal livello superiore, che verrà ora utilizzata come ingresso, l'obiettivo di questo *layer* è quello di definire e generare le corrispondenti traiettorie di riferimento dello stato $\tilde{x}_k^{[i]}$ e dell'ingresso $\tilde{u}_k^{[i]}$ per ogni sottosistema $i = 1, \ldots, M$ che il corrispondente agente deve inseguire. Per far ciò si introduce una variabile di stato aggiuntiva corrispondente all'errore di inseguimento integrato che indicheremo con e_k. Al fine di definire le traiettorie di riferimento $\tilde{x}_k^{[i]}$ e $\tilde{u}_k^{[i]}$ per ogni sottosistema $i = 1, \ldots, M$, si considera quindi il seguente sistema dinamico:

$$\begin{cases} \tilde{x}_{k+1}^{[i]} = & A\tilde{x}_k^{[i]} + B\tilde{u}_k^{[i]} \\ \tilde{e}_{k+1}^{[i]} = & \tilde{e}_k^{[i]} + \tilde{y}_{k+1}^{[i]} - C\tilde{x}_k^{[i]} \\ \tilde{u}_k^{[i]} = & K_x\tilde{x}_k^{[i]} + K_e\tilde{e}_k^{[i]} \end{cases} \tag{4.12}$$

in cui le matrici A, B, C replicano la dinamica del sistema controllato, mentre K_x e K_e sono parametri di progetto che andranno tarati opportunamente in modo da garantire la stabilità del sistema. Il sistema (4.12), che tiene conto delle condizioni nominali degli agenti, rappresenta quindi un modo alternativo per descrivere il comportamento reale del sistema considerando un integratore aggiuntivo per ogni uscita e assumendo quindi $w_k^{[i]} = 0$. L'integratore risulta dunque essere necessario per risolvere il problema di *tracking* nel modo migliore.

4.4.3 Inseguimento degli ingressi e delle variabili di stato: MPC robusto

Si vuole ora applicare la tecnica di controllo robusta *Tube-Based MPC*, presentata nel paragrafo 4.3, con l'obiettivo di mantenere le traiettorie degli stati e degli ingressi più vicine possibili a quelle di riferimento $\tilde{x}_k^{[i]}$ e $\tilde{u}_k^{[i]}$, definite al livello precedente, rispettando allo stesso tempo i vincoli dati sulle variabili di stato. Utilizzando questa tecnica di controllo si è visto che anche a fronte di disturbi sulle variabili di processo le traiettorie dello stato e dell'ingresso di ogni agente rimangono vicine ai riferimenti generati dal livello precedente.

Il problema di ottimizzazione è basato sulla predizione dell'evoluzione del modello nominale, definito come:

$$\hat{x}_{k+1}^{[i]} = A\hat{x}_k^{[i]} + B\hat{u}_k^{[i]} \tag{4.13}$$

In accordo con quanto già accennato nel paragrafo 4.3 e come discusso dettagliatamente in [21], il problema di ottimizzazione complessivo da risolvere per questo livello risulta essere:

$$\min_{\hat{x}_k^{[i]}, \hat{u}_{k:\,k+N-1}^{[i]}} \sum_{h=0}^{N-1} \| \hat{x}_{k+h}^{[i]} - \tilde{x}_{k+h}^{[i]} \|_Q^2 + \| \hat{u}_{k+h}^{[i]} - \tilde{u}_{k+h}^{[i]} \|_R^2 + \| \hat{x}_{k+N}^{[i]} - \tilde{x}_{k+N}^{[i]} \|_P^2$$

$$\begin{gathered}\hat{x}_{k+1}^{[i]} = A\hat{x}_k^{[i]} + B\hat{u}_k^{[i]} \\ \hat{x}_{k+h}^{[i]} \in \hat{\mathbb{X}}_i, \forall h = 0, \dots, N-1 \\ \hat{u}_{k+h} \in \hat{\mathbb{U}}, \forall h = 0, \dots, N-1 \\ x_k^{[i]} - \hat{x}_k^{[i]} \in \mathcal{E}_i \\ C\left(\hat{x}_{k+h}^{[i]} - \tilde{x}_{k+h}^{[i]}\right) \in \Delta_i^z, \forall h = 0, \dots, N-1 \\ \hat{x}_{k+N}^{[i]} - \tilde{x}_{k+N}^{[i]} \in \mathcal{K}_i\mathcal{E}_i\end{gathered} \tag{4.14}$$

dove per ora ci basti sapere che $\mathcal{K}_i$ è un opportuno parametro, $\mathcal{E}_i$ è un *set* che contiene la dinamica dell'errore tra lo stato reale dell'agente e il suo corrispondente stato nominale, mentre Δ_i^z è un *set* arbitrario che consente di mantenere una certa flessibilità nelle operazioni di inseguimento della traiettoria di riferimento; i *set* trovati verranno poi descritti dettagliatamente nel successivo paragrafo. Inoltre le matrici simmetriche $Q \geq 0$ e $R > 0$ possono essere scelte arbitrariamente, mentre P deve soddisfare l'equazione di *Lyapunov* (4.5) dove K è scelto in modo che $(A + BK)$ sia una *matrice di Schur*[3]. Il risultato di (4.14) consente quindi di ricavare $\left(\hat{x}_{k|k}^{[i]}, \hat{u}_{k:\, k+N-1|k}^{[i]}\right)$. L'ingresso $u_k^{[i]}$ è ottenuto nel seguente modo:

$$u_k^{[i]} = \hat{u}_{k|k}^{[i]} + K\left(x_k^{[i]} - \hat{x}_{k|k}^{[i]}\right)$$

Il *set* $\hat{\mathbb{X}}_i$ viene invece ricavato come:

$$\hat{\mathbb{X}}_i = \mathbb{X}_i \ominus \mathcal{E}_i$$

al fine di garantire che $x_k^{[i]} \in \mathbb{X}_i$. Per maggiori dettagli e chiarimenti si rimanda a [21].

Un ulteriore importante risultato è il seguente: se la relazione (4.11) viene mantenuta, allora esiste un *set* limitato tempo invariante $\mathscr{Y}_i$ per cui:

$$y_k^{[i]} \in \tilde{y}_k^{[i]} \oplus \mathscr{Y}_i \tag{4.15}$$

per tutti gli istanti $k \geq 0$.

Questo significa che, nel problema considerato, se la velocità di crociera dell'agente è sufficientemente limitata, allora esiste a priori una zona conosciuta e a tempo invariante in cui risulterà essere contenuta la posizione dell'agente. Perciò il problema di pianificazione della traiettoria può essere direttamente risolto prendendo in considerazione questo *set* $\mathscr{Y}_i$ d'incertezza e quindi ricavando così anche una soluzione robusta ai problemi di *avoidance*, indipendentemente dai disturbi esterni agenti sul sistema purché contenuti nel *set* $\mathbb{W}$. Anche il *set* $\mathscr{Y}$ verrà ampiamente descritto nel prossimo paragrafo.

4.4.4 Calcolo e significato dei vari set

Andiamo ora ad analizzare come viene calcolato il *set* $\mathscr{Y}_i$ incontrato nel terzo livello della struttura gerarchica *DPC*. Il primo passo da fare è quello di trovare l'errore massimo tra la traiettoria di riferimento dell'uscita $\tilde{y}_k$, definita nel primo livello, e la corrispondente uscita per la traiettoria di riferimento dello stato $C\tilde{x}_k$. Tali valori

[3] Una matrice è detta *matrice di Schur* se tutti i suoi autovalori sono contenuti all'interno del cerchio di raggio unitario

risultano essere asintoticamente uguali solo nel caso in cui $\tilde{y}_k$ sia costante, a causa della presenza dell'integratore nel secondo livello. Il secondo livello si basa infatti sul sistema descritto in (4.12) che, tramite una semplice operazione di sostituzione della variabile $\tilde{u}_k$ nella sua prima equazione, può essere riscritto in modo compatto nella seguente forma:

$$\tilde{\mathcal{X}}_{k+1} = \mathscr{F}\tilde{\mathcal{X}}_k + \mathscr{G}\tilde{y}_{k+1} \tag{4.16}$$

dove

$$\tilde{\mathcal{X}}_k = \begin{pmatrix} \tilde{x}_k \\ \tilde{e}_k \end{pmatrix}, \mathscr{F} = \begin{bmatrix} A + BK_x & BK_e \\ -C & I \end{bmatrix}, \mathscr{G} = \begin{bmatrix} 0 \\ I \end{bmatrix}$$

Il sistema (4.16) risulta essere asintoticamente stabile solo se la matrice $\mathscr{F}$ è una *matrice di Schur*; si devono quindi trovare, come anticipato in precedenza, i valori di K_x e K_e che permettano di garantire tale condizione. Riscrivendo ora la matrice dinamica $\mathscr{F}$ nella seguente forma:

$$\mathscr{F} = \begin{bmatrix} A + BK_x & BK_e \\ -C & I \end{bmatrix} = \underbrace{\begin{bmatrix} A & 0 \\ -C & I \end{bmatrix}}_{\bar{A}} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{\bar{B}} \begin{bmatrix} K_x & K_e \end{bmatrix}$$

è possibile riportarsi a un classico problema di stabilizzazione per il quale esistono algoritmi di soluzione, purché la coppia $\left(\bar{A}, \bar{B}\right)$ sia stabilizzabile.

Come detto, l'utilizzo di questo sistema per il calcolo delle traiettorie di riferimento per lo stato e per l'ingresso, assume che il riferimento per le uscite $\tilde{y}$, a un istante generico k, sia costante. In altre parole, il sistema (4.12) garantisce un perfetto inseguimento nel caso in cui si mantenga il riferimento $\tilde{y}_k$ costante fintantoché l'uscita $C\tilde{x}_k$ non lo raggiunge, per passare poi al riferimento, $\tilde{y}_{k+1}$, successivo. In un caso generico, ciò non è pero sempre verificato, in quanto si hanno traiettorie tempo-varianti: è allora opportuno studiare quanto la traiettoria inseguita dal sistema (4.12), si discosti da quella di riferimento. A tale proposito, si consideri $\tilde{\mathcal{X}}_k^{SS}$ valore di equilibrio dello stato allargato $\tilde{\mathcal{X}}_k$, definito in (4.16), corrispondente al riferimento costante pari a $\tilde{y}_k$:

$$\tilde{\mathcal{X}}_k^{SS} = \mathscr{F}\tilde{\mathcal{X}}_k^{SS} + \mathscr{G}\tilde{y}_k$$

Si vuole quindi trovare una relazione tra $\tilde{\mathcal{X}}_k$ e il corrispondente valore di equilibrio $\tilde{\mathcal{X}}_k^{SS}$. Consideriamo a tale proposito la seguente differenza:

$$\tilde{\mathcal{X}}_{k+1}^{SS} - \tilde{\mathcal{X}}_k^{SS} = \mathscr{F}\left(\tilde{\mathcal{X}}_{k+1}^{SS} - \tilde{\mathcal{X}}_k^{SS}\right) + \mathscr{G}\left(\tilde{y}_{k+1} - \tilde{y}_k\right)$$

il che porta a scrivere:

$$\tilde{\mathcal{X}}_{k+1}^{SS} - \tilde{\mathcal{X}}_k^{SS} = \left(I - \mathscr{F}\right)^{-1}\mathscr{G}\left(\tilde{y}_{k+1} - \tilde{y}_k\right) \tag{4.17}$$

Possiamo ora calcolare l'equazione dinamica dell'errore tra lo stato $\tilde{\mathcal{X}}_k$ e il suo corrispondente valore di equilibrio $\tilde{\mathcal{X}}_k^{SS}$:

$$\begin{aligned} \tilde{\mathcal{X}}_{k+1} - \tilde{\mathcal{X}}_{k+1}^{SS} &= \mathscr{F}\left(\tilde{\mathcal{X}}_k - \tilde{\mathcal{X}}_{k+1}^{SS}\right) \\ &= \mathscr{F}\left(\tilde{\mathcal{X}}_k - \tilde{\mathcal{X}}_k^{SS}\right) - \mathscr{F}\left(\tilde{\mathcal{X}}_{k+1}^{SS} - \tilde{\mathcal{X}}_k^{SS}\right) \\ &= \mathscr{F}\left(\tilde{\mathcal{X}}_k - \tilde{\mathcal{X}}_k^{SS}\right) + \tilde{w}_k \end{aligned} \tag{4.18}$$

dove $\tilde{w}_k$ può essere visto come un disturbo agente sull'errore. Richiamando le equazioni (4.17) e (4.11), il dominio di $\tilde{w}_k$ può essere facilmente trovato in questo modo:

$$\begin{aligned} \tilde{w}_k &= -\mathscr{F}\left(\tilde{\mathcal{X}}_{k+1}^{SS} - \tilde{\mathcal{X}}_k^{SS}\right) \\ &= -\mathscr{F}\left(I - \mathscr{F}\right)^{-1}\mathscr{G}\left(\tilde{y}_{k+1} - \tilde{y}_k\right) \\ &\in -\mathscr{F}\left(I - \mathscr{F}\right)^{-1}\mathscr{G}\mathscr{B}_{p,\varepsilon}(0) = \tilde{\mathbb{W}} \end{aligned}$$

Così tenuto conto di (4.18) e in accordo con [21], se $\mathscr{F}$ è una *matrice di Schur*, esiste un *set* robusto positivamente invariante per l'errore $\tilde{\mathcal{X}}_k - \tilde{\mathcal{X}}_k^{SS}$ definito come:

$$\Delta x = \bigoplus_{h=0}^{\infty} \mathscr{F}^h \tilde{\mathbb{W}}$$

Ciò significa che l'evoluzione dell'errore

$$\tilde{\mathcal{X}}_{k+h} - \tilde{\mathcal{X}}_{k+h}^{SS},\ \forall h > 0$$

resta confinata in Δx indipendentemente dal valore assunto da w_k, posto che:

$$\tilde{\mathcal{X}}_k - \tilde{\mathcal{X}}_k^{SS} \in \Delta x$$

Il *set* Δx permette quindi di stabilire con precisione una regione di incertezza nella quale si troverà la traiettoria delle uscite dell'agente rispetto alla traiettoria di riferimento data. In particolare, per la definizione di *set* robusto positivamente invariante e richiamando la definizione di $\tilde{\mathcal{X}}_k$, possiamo scrivere:

$$\tilde{\mathcal{X}}_k \in \tilde{\mathcal{X}}_k^{SS} \oplus \Delta x \Rightarrow \tilde{x}_k \in \tilde{x}_k^{SS} \oplus \begin{bmatrix} I & 0 \end{bmatrix} \Delta x$$

In più, sapendo che $C\tilde{x}_k^{SS} = \tilde{y}_k$, tramite semplici operazioni otteniamo:

$$C\tilde{x}_k \in \tilde{y}_k \oplus \begin{bmatrix} I & 0 \end{bmatrix} \Delta x \tag{4.19}$$

risultante da un rapporto tra la traiettoria di riferimento dell'uscita e la variabile d'uscita associata con la traiettoria di riferimento per lo stato, quindi tra $\tilde{y}_k$ e $C\tilde{x}_k$. Tale informazione permetterà quindi di determinare a priori un intorno della traiettoria di riferimento in cui si avrà la certezza di trovare l'agente e tale concetto potrà essere successivamente utilizzato, ad esempio, per evitare la collisione dei *robot* con eventuali ostacoli.

Inoltre, come già spiegato in precedenza, l'ingresso robusto u_k è ottenuto attraverso un problema di ottimizzazione basato sul sistema nominale (4.13); è quindi possibile definire l'errore come:

$$\varepsilon_k = x_k - \hat{x}_k$$

la cui dinamica è descritta da:

$$\varepsilon_{k+1} = (A + BK)\,\varepsilon_k + w_k$$

dove il disturbo $w_k \in \mathbb{W}$ definito precedentemente. Quindi, se la matrice $(A + BK)$ è una *matrice di Schur,* allora esiste, in accordo con [21], un *set* robustamente positivamente invariante, definito come:

$$\mathcal{E} = \bigoplus_{h=0}^{\infty} (A + BK)^h\, \mathbb{W}$$

È possibile quindi affermare che lo stato x_k del sistema reale si trova sempre in un intorno limitato dello stato nominale $\hat{x}_k$, infatti:

$$\varepsilon_k \in \mathcal{E} \Rightarrow x_k \in \hat{x}_k \oplus \mathcal{E} \tag{4.20}$$

Come discusso in [21], si avrà poi che il problema di ottimizzazione è anche soggetto al seguente vincolo:

$$C\hat{x}_k \in C\tilde{x}_k \oplus \Delta^z,\ k = 0, \ldots, N-1 \tag{4.21}$$

in cui C è la trasformazione di uscita. Tale vincolo porta a definire un intorno dell'uscita corrispondente alla traiettoria di riferimento dello stato in cui si è certi di trovare l'uscita del sistema reale. Combinando i vincoli (4.20) e (4.21) si ottiene:

$$Cx_k \in C\tilde{x}_k \oplus C\mathcal{E} \oplus \Delta^z \in C\tilde{x}_k \oplus \mathscr{Z},\ \mathscr{Z} = C\mathcal{E} \oplus \Delta^z$$

Infine applicando il vincolo (4.19) ricaviamo:

$$y_k = Cx_k \in \tilde{y}_k \oplus \mathscr{Y}$$

dove

$$\mathscr{Y} = C \begin{bmatrix} I & 0 \end{bmatrix} \Delta x \oplus \mathscr{Z}$$

Siamo dunque riusciti a calcolare, come da obiettivo iniziale, il *set* $\mathscr{Y}$ che permetterà di determinare una regione di incertezza nel piano di lavoro, definita attorno alla traiettoria di riferimento, in cui è garantita la presenza dell'agente.

4.4.5 Problema di ottimizzazione per la generazione della traiettoria di riferimento dell'uscita

Il livello di generazione della traiettoria di riferimento dell'uscita è utilizzato, come visto, per definire i valori $\tilde{y}_k^{[i]}, \tilde{y}_{k+1}^{[i]}, \ldots$, che corrispondono a degli ingressi per il livello successivo, ovvero quello di generazione delle traiettorie di riferimento per gli ingressi e le variabili di stato. Per ogni agente i si assume quindi che, all'istante k, la traiettoria di riferimento futura dell'uscita $\left\{\tilde{y}_k^{[i]}, \ldots, \tilde{y}_{k+N-1}^{[i]}\right\}$ sia data. Ora che conosciamo dettagliatamente il significato di tutti i vari *set* del nostro problema di controllo non ci resta che andare ad analizzare il problema di ottimizzazione che viene risolto nel primo livello della struttura gerarchica *DPC*. La risoluzione di tale problema consente la corretta generazione della traiettoria di riferimento dell'uscita, che influirà, come visto, anche su tutti i livelli successivi. Definiamo dunque $\bar{y}_{k+N|k}^{[i]}$ come la soluzione del seguente problema di ottimizzazione:

$$\min_{\bar{y}_{k+N}^{[i]}} V_i^z \tag{4.22}$$

dove la funzione di costo V_i^z è dettagliata nel seguito. Il problema di minimizzazione è soggetto ai seguenti vincoli:

- vincoli di *obstacle avoidance;*
- vincoli di *collision avoidance*;
- $\bar{y}_{k+N}^{[i]} \in \tilde{y}_{k+N-1}^{[i]} \oplus \beta_{q,\bar{\varepsilon}_i}^{(2)}(0)$;

- $\bar{y}_{k+N}^{[i]} \in \mathbb{Z}_i$;

in cui il *set* $\mathbb{Z}_i$ è definito in modo tale che:

$$\begin{bmatrix} I & 0 \end{bmatrix} \left((I - \mathscr{F})^{-1} \mathscr{G} \mathbb{Z}_i \oplus \Delta_i^{\mathcal{X}} \right) \oplus \mathcal{K}_i \mathcal{E}_i \subseteq \hat{\mathbb{X}}_i$$

Nel problema (4.22), la minimizzazione della funzione di costo V_i^z permette la corretta gestione dei problemi di *collision avoidance* e di controllo di formazione. I vincoli per il problema di *collision avoidance* devono essere soddisfatti da tutti i *robot* in modo che risulti sempre garantita la (4.11).

La soluzione per il problema di ottimizzazione (4.22) è quindi data dal valore ottimo $\bar{y}_{k+N|k}^{[i]}$, e la traiettoria di riferimento dell'uscita verrà aggiornata in accordo con l'equazione:

$$\tilde{y}_{k+N}^{[i]} = \bar{y}_{k+N|k}^{[i]}$$

Il problema di *tracking* viene quindi risolto andando a imporre che, per ogni agente, la sua traiettoria raggiunga l'obiettivo finale rispettando i vari vincoli assegnati, ovvero si pone: $y_k \to y_{goal}$. Perciò la funzione di costo adottata sarà del tipo:

$$V_i^z = \gamma \parallel \bar{y}_{k+N}^{[i]} - \tilde{y}_{k+N-1}^{[i]} \parallel^2 + \parallel \bar{y}_{k+N}^{[i]} - y_{goal}^{[i]} \parallel_T^2$$

dove T e γ sono pesi che vanno scelti in modo che sia soddisfatto il vincolo $T > \gamma I$.

4.5 Caratteristiche generali dello schema studiato

In conclusione si può affermare che il problema di inseguimento delle traiettorie di riferimento delle uscite viene tradotto in un problema di inseguimento delle traiettorie dello stato e dell'ingresso del sistema; per ulteriori dettagli e dimostrazioni sui risultati ottenuti si rimanda a [2, 13]. È comunque importante sottolineare che, progettando la struttura di controllo robusto nel modo illustrato, è possibile disaccoppiare il problema di inseguimento da quello di generazione del riferimento, rendendo quest'ultimo indipendente dalla specifica tipologia di agente. Infatti, l'ingresso del sistema di controllo è la traiettoria di riferimento delle uscite, su cui non sono imposti vincoli riguardanti la tipologia e la dinamica degli agenti. Di conseguenza, la tecnica utilizzata per conseguire tale obiettivo può essere scelta in modo arbitrario, senza alcun vincolo rispetto alle tecniche implementate nei livelli di controllo dedicati all'inseguimento. Inoltre si osserva che il livello di generazione della traiettoria è indipendente dai livelli sottostanti. Ciò permette una più semplice soluzione del problema di coordinamento, poiché le interazioni fra gli agenti sono gestite in tale livello, ignorando il problema dell'inseguimento delle traiettorie generate. Per una descrizione più dettagliata di tale struttura di controllo si rimanda a [13].

Riassumendo, dunque, siamo in grado di controllare, con l'approccio visto, il movimento del singolo *robot* verso l'obiettivo finale formulando il tutto come un problema d'inseguimento di una traiettoria di riferimento. In particolare, la struttura dei vincoli presentata fa sì che si abbia la garanzia che la traiettoria percorsa realmente dall'agente sia costretta a stare in un intorno ben preciso di quella di riferimento, e con una incertezza nota a priori. In questo modo è possibile risolvere il problema di *collision avoidance* semplicemente ricorrendo a un'implementazione distribuita in cui ciascun agente trasmette ai suoi vicini la sequenza ideale dei suoi passi successivi

lungo l'orizzonte predittivo, insieme all'incertezza con cui questi saranno percorsi. Tale informazione andrà allora a costituire il *set* di ostacoli, con relativa dimensione, che ciascuno degli agenti restanti dovrà considerare nella soluzione della propria ottimizzazione; proprio questo problema sarà oggetto di analisi del prossimo capitolo.

Capitolo 5

Navigazione autonoma di agenti mobili

5.1 Introduzione al problema

Entriamo ora nel merito del problema della navigazione autonoma: come discusso in [9], l'obiettivo consiste nel guidare uno o più agenti verso dei punti noti dello spazio di lavoro chiamati *goal,* tenendo presente il fatto che nell'ambiente potrebbero esserci anche delle cause di impedimento che non permettono uno scontato raggiungimento dell'obiettivo finale. Solitamente questi impedimenti corrispondono a degli ostacoli che si suppone abbiano una posizione nota e prefissata; in questo caso si parla di problema di navigazione *obstacle avoidance,* come mostrato nella Figura 5.1. In alternativa si potrebbe trattare di ostacoli dinamici costituiti da altri *robot* nella

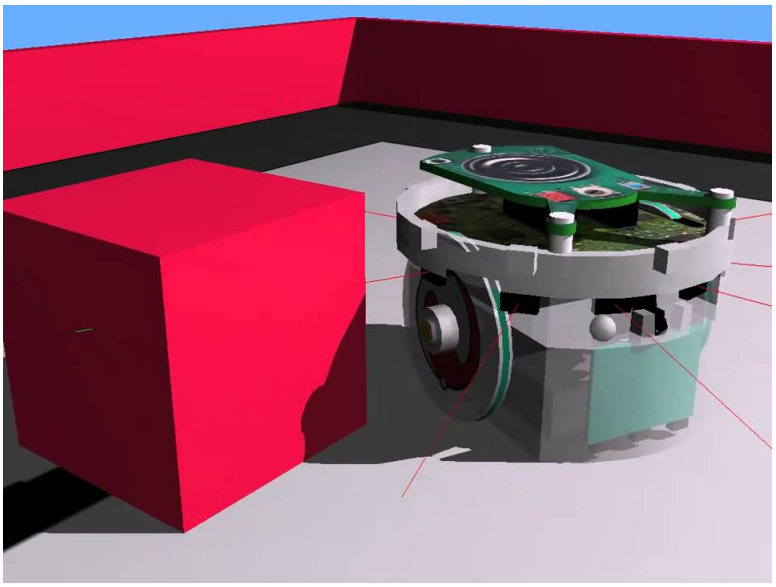


Figura 5.1: Problema dell'o*bstacle avoidance*

medesima superficie di lavoro e in questo caso si parla di problema di *collision avoidance.* Si tratta quindi di trovare un algoritmo che permetta di generare un percorso nello spazio operativo privo di collisioni (*path planning*) nel rispetto della geometria dell'agente e dell'ambiente circostante, seguito successivamente dalla generazione di

una traiettoria (*trajectory planning*) che attribuisce al percorso una legge temporale costruita sulla base della dinamica dell'agente sotto controllo [8].

Occupiamoci inizialmente del problema relativo a un solo agente per poi passare al caso di coordinamento e gestione di più *robot*. Normalmente quando controlliamo un *robot* mobile imponendogli l'inseguimento di una traiettoria desiderata fino al raggiungimento dell'obiettivo finale, espresso sotto forma di coordinate (x, y), si vuole che esso riesca a rispettare il suo compito operando in uno spazio di lavoro solamente parzialmente noto a priori, ovvero anche se sono presenti ostacoli di qualsiasi tipo e dimensione lungo la sua traiettoria. Grazie ai suoi sensori di bordo, l'agente infatti dovrebbe comunque essere in grado di raggiungere il *goal*. Tale problema di navigazione autonoma può quindi essere scomposto in due sotto-problemi: il primo riguarda la capacità dell'agente di adattarsi e reagire agli ostacoli che può incontrare lungo il suo cammino trovando sempre in tempi brevi e *on-line* la soluzione migliore per affrontare il problema. Il secondo, già analizzato precedentemente, consiste nel *path following* della traiettoria desiderata. Si può quindi inizialmente ragionare pensando che durante l'esecuzione dei suoi *task* il *robot* risolva il secondo problema, ma in caso di variazioni dello scenario, quali per esempio ostacoli o altri agenti, il *robot* passi a risolvere anche il primo problema pur mantenendo la priorità del raggiungimento dell'obiettivo finale. Avremo quindi a che fare in primo luogo con il problema di *path plannig* e parallelamente, in alcune occasioni, avremo bisogno di risolvere problemi dinamici di *obstacle* e di *collision avoidance*.

Per chiarezza si effettua ora una distinzione tra gli algoritmi di risoluzione per i due problemi di *avoidance* che si potrebbero incontrare i quali risultano essere fortemente correlati poiché entrambi tendono a creare deformazioni intelligenti della traiettoria ottimale, per permettere all'agente il raggiungimento del *goal* senza subire collisioni:

- *obstacle avoidance* si intende un algoritmo che permetta agli agenti di evitare ostacoli fissi e noti a priori che costituiscono un blocco verso il raggiungimento dell'obiettivo finale;

- *collision avoidance* si intende un algoritmo che permetta ai *robot* di evitare altri agenti durante il loro movimento e per far questo si sfrutta il fatto che, grazie al controllo *MPC*, si conoscono le predizioni future delle traiettorie dei *robot*. Ciò permette di vedere gli altri agenti, operanti nello spazio di lavoro, come se fossero ostacoli mobili.

Si noti che i problemi di *collision* e *obstacle avoidance* sono del tutto analoghi se un ostacolo viene considerato come un *robot*.

Nel caso trattato l'idea sfruttata è stata quella di utilizzare una tecnica di *switch*, gestita tramite cicli *for*, dove un anello di controllo esterno permette di garantire la generazione della traiettoria migliore per far raggiungere agli agenti i rispettivi *goal* tramite tecniche predittive e, nel caso in cui un *robot* entri nella regione di attivazione data da un ostacolo o da un altro agente, viene applicata una speciale legge di controllo di *obstacle avoidance* o di *collision avoidance* che manterrà tale *robot* a una distanza voluta dall'ostacolo, evitandone così la collisione. Il tutto agisce quindi in modo simultaneo e una volta superato l'ostacolo si ritorna a eseguire singolarmente il solo problema di inseguimento della traiettoria (*path following),* come mostrato nella Figura 5.2. Nel lavoro illustrato si conoscono nel dettaglio le misure del piano operativo, avendo quindi piena capacità dello spazio di lavoro degli agenti, tuttavia la

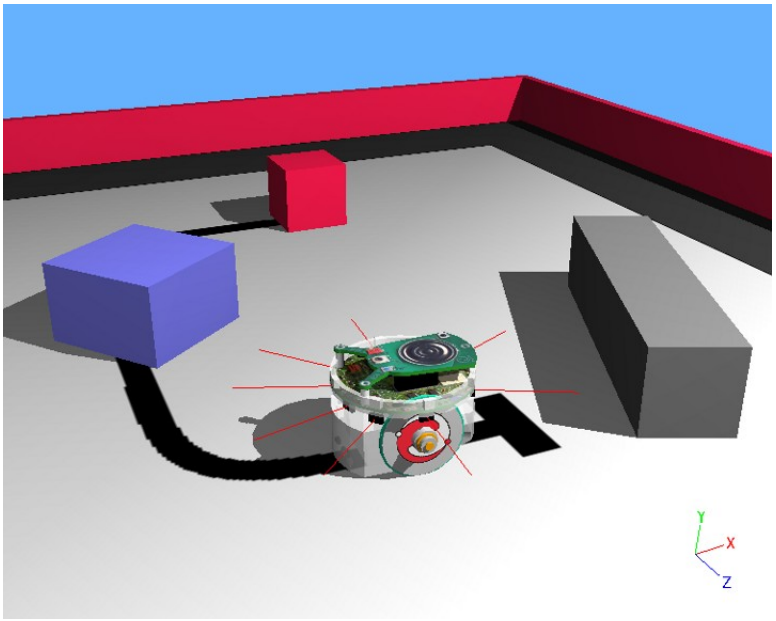


Figura 5.2: Inseguimento del riferimento unito al problema di *obstacle avoidance*

disposizione degli ostacoli e dei *goal* dei *robot* stessi può variare in maniera dinamica e a nostro piacimento. Il nostro algoritmo dovrà essere quindi robusto, in modo da riuscire a tollerare tutte queste possibili variazioni, le quali possono essere viste come disturbi e inoltre dovrà anche essere robusto rispetto a generici disturbi sul sistema, come ad esempio disturbi di attuazione.

5.2 Obstacle Avoidance

Il problema dell'*obstacle avoidance* può essere risolto inserendo un opportuno vincolo aggiuntivo nel problema di ottimizzazione formulato nel controllo MPC; nello specifico tale vincolo dovrà essere adeguatamente considerato nel livello di generazione delle traiettorie di riferimento dell'uscita. Tramite il corretto coinvolgimento delle variabili di riferimento dell'uscita $\tilde{y}_{k+N}^{[i]}$, analizzate nel paragrafo 4.4.1, con $i = 1, \ldots, M$ numero degli agenti utilizzati, e riprendendo la formulazione (4.15) che introduceva il *set* $\mathscr{Y}_i$, il quale ribadiamo rappresenta la regione di incertezza in cui si troverà l'agente, sarà possibile risolvere, come vedremo nel seguito, il problema preso in esame. Si può infatti richiedere che l'agente venga mantenuto a una distanza minima da ciascun ostacolo, opportunamente scelta in base all'incertezza della posizione effettiva del *robot*, il tutto sempre supponendo di conoscere sia la dimensione che la posizione di tutti gli ostacoli presenti nello spazio operativo; per ulteriori chiarimenti si rimanda a [10].

Si vuole ora trattare il problema di controllare un agente che eviti degli ostacoli assunti circolari, le cui posizioni e dimensioni sono note a priori. Si consideri dunque per ogni *robot* i, con $i = 1, \ldots, M$, un *set* di n_i^o ostacoli prossimi all'agente chiamato $\mathscr{O}_i \subseteq \{1, \ldots, n_i^o\}$. Ogni ostacolo viene specificato utilizzando l'indice h e sarà definito tramite le coordinate del suo centro $y_h^o = (x_c, y_c)$, riferite al piano di lavoro, e il suo raggio R_h^o.

Ricaviamo ora, tramite una rappresentazione politopica del vincolo di minima distanza precedentemente enunciato, vincoli convessi che potranno essere usati per risolvere il problema di ottimizzazione (4.22); essi infatti sono necessari per riformulare la distanza minima richiesta, tra *robot* e ostacolo, usando appunto disequazioni lineari. A tale proposito consideriamo un generico vincolo di distanza minima:

$$\| y - \bar{y} \|_2 \geq \bar{d} \tag{5.1}$$

dove $\bar{y}$ e $\bar{d}$ sono date. Definiamo ora un politopo simmetrico $\mathscr{P}$, circoscritto all'ostacolo circolare, che è definito usando un numero r di disequazioni lineari, cioè:

$$\mathscr{P}: = \left\{ y : h_t^T (y - \bar{y}) \leq \bar{d} \right\} \tag{5.2}$$

Osserviamo che la violazione di ognuna delle disequazioni

$$h_t^T (y - \bar{y}) \leq \bar{d}$$

(cioè esiste $\bar{t} \in \{1, \ldots, r\}$ così che $h_{\bar{t}}^T (y - \bar{y}) > \bar{d}$) implica che (5.1) è verificata. Questa proprietà sarà meglio spiegata nel seguito della trattazione.

La proprietà (4.15), come detto in precedenza, risulterà essere particolarmente adatta per risolvere il problema di *obstacle avoidance* definendo infatti la massima regione di incertezza tramite il *set* $\mathscr{Y}_i$ come:

$$\delta_i = \max_{\delta y \in \mathscr{Y}_i} \| \delta y \| \tag{5.3}$$

Inoltre sapendo che $R_{robot} = 2.1\,cm$, che sarà costante per ogni agente *e-puck*, possiamo verificare che:

$$\| \tilde{y}_{k+N}^{[i]} - y_h^o \|_2 \geq R_{robot} + R_h^o + \delta_i$$

garantendo dunque che:

$$\| y_{k+N}^{[i]} - y_h^o \|_2 \geq R_{robot} + R_h^o$$

e quindi prevenendo così la collisione tra il *robot* i e l'ostacolo h considerato. Riprendendo ora il problema di ottimizzazione (4.22) all'istante k corrispondente all'uscita $\bar{y}_{k+N}^{[i]}$, il seguente vincolo ricavato dovrà essere incluso in (4.22) per garantire la corretta risoluzione del problema di *obstacle avoidance* rispetto all'ostacolo h considerato:

$$\| \bar{y}_{k+N}^{[i]} - y_h^o \|_2 \geq R_{robot} + R_h^o + \delta_i \tag{5.4}$$

Si riesce in questo modo a far sì che la posizione del *robot* rispetto al centro dell'ostacolo y_h^o non sia mai minore della quantità $R_{robot} + R + \delta_i$ garantendo così che l'agente non collida con l'ostacolo considerato appartenente al *set* $\mathscr{O}_i$ precedentemente definito. Tuttavia, il vincolo (5.4) risulta essere non convesso; per preservare la convessità del problema di ottimizzazione, è possibile convertire (5.4) in un vincolo lineare. Per far ciò definiamo ora un politopo $\mathscr{P}_{hi}^o$ circoscritto all'ostacolo centrato in y_h^o, con raggio $R_{robot} + R_h^o + \delta_i$, definito per un *set* r_{hi}^o di disequazioni lineari:

$$\left(h_t^{[o,hi]} \right)^T \left(\bar{y}_{k+N}^{[i]} - y_h^o \right) \leq d_{hi}^o$$

con $t = 1, \ldots, r^o_{hi}$. Inoltre definiamo

$$t^{[o,hi]}_{max}(k+N-1) = \underset{t \in \{1,\ldots,r^o_{hi}\}}{argmax} \left(h^{[o,hi]}_t\right)^T \left(\tilde{y}^{[i]}_{k+N-1} - y^o_h\right) - d^o_{hi}$$

All'istante k potrà quindi essere utilizzato, per il *robot* i considerato, il seguente vincolo lineare rimpiazzando (5.4) e garantendo nuovamente la corretta risoluzione del problema di *obstacle avoidance*:

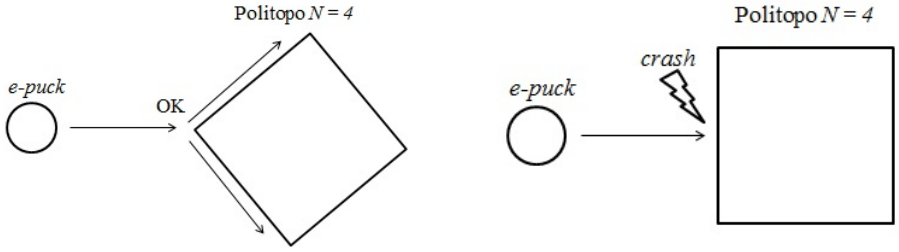
$$\left(h^{[i,j]}_{t^{[o,hi]}_{max}(k+N-1)}\right)^T \left(\bar{y}^{[i]}_{k+N} - y^o_h\right) \geq d^o_{hi} \tag{5.5}$$

In [13] si è utilizzato un algoritmo per risolvere il problema di *obstacle avoidance* che gestiva vincoli (5.4) di tipo non lineare, il che ne aumentava la complessità dando come risultato un problema di ottimizzazione non convesso, con conseguenti elevati tempi di risoluzione e quindi di non semplice soluzione. Tuttavia, tramite prove sperimentali si giunge alla conclusione che tale algoritmo non presenta tempi computazionali eccessivi, come invece accadrà per il problema di *collision avoidance* che verrà analizzato nel prosieguo, e sarà quindi sempre in grado di soddisfare le specifiche richieste permettendo al *robot* di superare con successo l'ostacolo. In ogni caso, per ridurre la complessità computazionale si è optato per modificare l'algoritmo in modo da permettere, tramite l'utilizzo di vincoli questa volta lineari (5.5), una più rapida risoluzione del problema e una minor potenza di calcolo richiesta.

5.2.1 Descrizione dell'algoritmo specifico

Viene descritto ora dettagliatamente il nuovo algoritmo implementato che utilizza, come detto, vincoli lineari, ma presenta gli stessi parametri dell'algoritmo analizzato in precedenza. Si ha infatti $y^o_h = (x_c, y_c)$ che rappresenta le coordinate del centro dell'ostacolo, d che rappresenta la distanza minima desiderata da y^o_h la quale dipenderà, anche in questo caso, dal raggio dell'ostacolo R^o_h, dal raggio del *robot e-puck* R_{robot} e da una certa distanza δ_i calcolata come visto in precedenza; indichiamo inoltre con (x_a, y_a) le coordinate riferite alla posizione attuale dell'agente. Come fatto precedentemente, si definisce ora un politopo simmetrico $\mathscr{P}$, circoscritto all'ostacolo circolare, usando un numero r di disequazioni lineari. Il politopo quindi presenterà N lati con $N = r$; in particolare lavorando su un ambiente definito in $\mathbb{R}^2$, ovvero una superficie, si potrà parlare nello specifico di un poligono. Osserviamo inoltre che l'approssimazione del poligono a una circonferenza sarà tanto più precisa quanto più è alto il numero N di lati del poligono stesso. Il numero di lati del poligono deve essere almeno uguale a tre in modo da poter definire così una regione chiusa e limitata, ma possibilmente $N \geq 8$ così da ottenere anche una buona approssimazione di una circonferenza. Prendendo ad esempio $N = 4$ si ottiene un quadrilatero; questo potrebbe creare problemi in quanto, se l'agente si muovesse perpendicolarmente a un lato del quadrilatero non riuscirebbe a trovare una via di fuga per evitare l'ostacolo e quindi l'intero algoritmo non sarebbe più in grado di fornire una soluzione ammissibile; questo viene appunto rappresentato nella Figura 5.3 (b). Dalla Figura 5.3 (a) si vede bene che l'algoritmo funziona invece correttamente quando il *robot* si avvicina all'angolo del poligono, riuscendo così ad aggirarlo.

Nell'algoritmo implementato si utilizza $N = 20$, che rappresenta un ottimo compromesso tra la buona approssimazione del poligono a una circonferenza e la bassa



(a) (b)

Figura 5.3: Problemi derivanti da un limitato valore di N: (a) l'agente riesce a evitare l'ostacolo; (b) l'agente muovendosi perpendicolarmente al lato dell'ostacolo si blocca

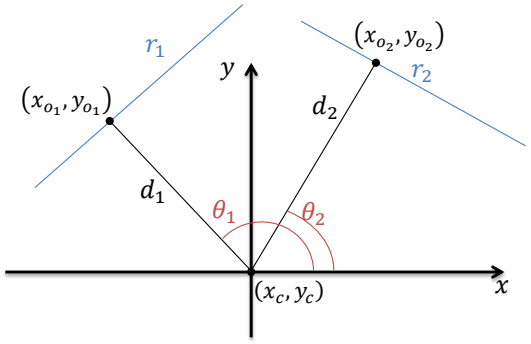
probabilità che il *robot*, in presenza di un ostacolo, vada a sbattere lungo un lato N del poligono in modo perpendicolare, incorrendo quindi nel problema mostrato nella Figura 5.3 (b). Inizialmente vengono dunque calcolate le equazioni lineari che descrivono le rette su cui giacciono i lati del poligono regolare, distanti d dal centro del poligono stesso. In questo modo si hanno equazioni del tipo:

$$d = (x - x_0)\cos\theta + (y - y_0)\sin\theta$$

ovvero in forma matriciale:

$$d = \begin{bmatrix} \cos\theta & \sin\theta \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix}$$

dove θ è l'angolo compreso tra l'asse x del piano cartesiano, avente origine in (x_c, y_c), e il segmento di distanza minima tra la retta stessa e l'origine, come mostrato nella Figura 5.4. Inoltre come si vede dalla figura (x_o, y_o) rappresentano le coordinate del punto di intersezione tra la retta r_i e il segmento d_i corrispondente con $i = 1, \ldots, N$. Successivamente si definisce il poligono cercato attraverso la disuguaglianza:



Per ogni retta r_i viene selezionato il corrispondente angolo θ e il corrispondente punto di coordinate (x_o, y_o)

Figura 5.4: Rappresentazione di θ e del punto (x_o, y_o) nel piano cartesiano

$$H\,(x - x_0) \leq d$$

, dove la matrice dei coefficienti H è definita come segue:

$$H = \begin{bmatrix} \cos\theta_1 & \cdots & \cos\theta_N \\ \sin\theta_1 & \cdots & \sin\theta_N \end{bmatrix}$$

e

$$\theta_n = \frac{2\pi(k-1)}{N}$$

con $0 < k \leq N$ e $0 < n \leq N$.

Successivamente si definisce il vettore colonna S di N righe:

$$S = \begin{bmatrix} d + \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \end{bmatrix} \\ \vdots \\ d + \begin{bmatrix} \cos\theta_N & \sin\theta_N \end{bmatrix} \begin{bmatrix} x_c \\ y_c \end{bmatrix} \end{bmatrix}$$

e il vettore J, che dipende direttamente dalla posizione attuale del *robot*, nel seguente modo:

$$J = \left[\begin{bmatrix} \cos\theta_1 & \sin\theta_1 \end{bmatrix} \begin{bmatrix} x_a \\ y_a \end{bmatrix} - S_1 \cdots \begin{bmatrix} \cos\theta_N & \sin\theta_N \end{bmatrix} \begin{bmatrix} x_a \\ y_a \end{bmatrix} - S_N \right]$$

Gli elementi del vettore J rappresentano le distanze del punto (x_a, y_a) dalle rette r_i individuate da

$$\begin{bmatrix} \cos\theta_i & \sin\theta_i \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} d = 0$$

e in particolare, se esiste un elemento di $J > 0$, ciò significa che (x_a, y_a) si trova al di fuori del poligono individuato. Supponendo quindi che, all'istante k, l'elemento massimo di J sia il t-esimo, all'istante $k+1$ il seguente vincolo verrà imposto:

$$A_{obst} x_{k+1} \leq b_{obst}$$

, dove

$$\begin{cases} A_{obst} = & -\begin{bmatrix} \cos\theta_k & \sin\theta_k \end{bmatrix} \\ b_{obst} = & -S_k \end{cases}$$

L'algoritmo presentato si basa quindi sulla seguente idea:

- si individua, data la posizione del *robot* all'istante k, il lato del poligono da cui l'agente, avente posizione (x_{a_k}, y_{a_k}), si trova a maggiore distanza, cioè l'elemento di J (positivo) avente valore maggiore;
- all'istante successivo $k+1$, per garantire *obstacle avoidance,* la violazione del vincolo individuato in precedenza verrà imposta nel problema di ottimizzazione.

In questo modo, ricorsivamente, si garantisce che l'agente non entri mai nella regione interna al poligono individuato, e quindi all'ostacolo circolare, risolvendo così il problema di *obstacle avoidance*. Nella Figura 5.5 viene mostrato in maniera grafica e intuitiva come opera l'algoritmo appena presentato.

L'algoritmo dovrà poi essere applicato per ogni ostacolo inserito nello spazio di lavoro, ottenendo in questo modo l'insieme finito delle varie componenti A_{obst}, di tante righe quanti sono gli ostacoli inseriti e di due colonne, e l'insieme delle componenti b_{obst} di tante righe quanti sono gli ostacoli inseriti, ma di una colonna; queste due componenti saranno quindi ricavate per ogni agente utilizzato. Successivamente

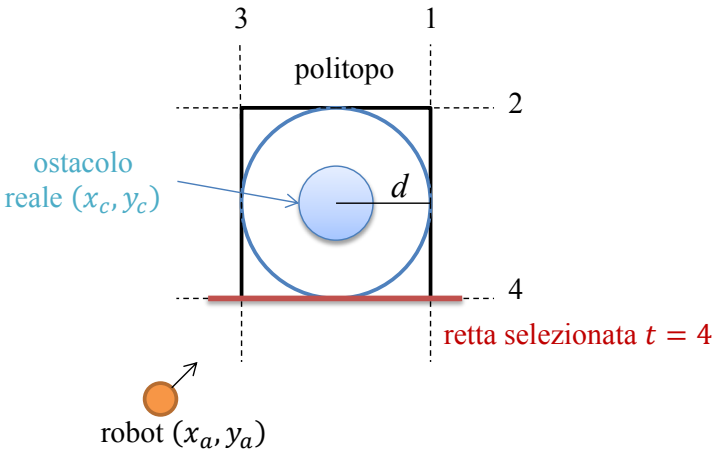


Figura 5.5: Algoritmo per l'*obstacle avoidance* che utilizza vincoli lineari (caso $N = 4$)

si andrà poi a inserire negli algoritmi di controllo, oltre che il vincolo sulla posizione attuale del *robot* espressa in coordinate (x, y) acquisita dalla *webcam* e sulle velocità v_x e v_y stimate tramite predittore di *Kalman*, le due componenti A_{obst}, b_{obst} relative ai vari ostacoli contenuti in $\mathcal{O}_i$, $i = 1, \ldots, M$, in modo da permettere così a ogni *robot* di evitarli.

5.2.2 Risultati sperimentali

Si effettua ora un *test* in cui vengono confrontati i risultati ottenuti tramite l'algoritmo che utilizza vincoli non lineari con quello appena descritto che lavora invece con vincoli lineari; le condizioni di operatività sono le seguenti: si ha un unico agente che si muove nello spazio operativo seguendo la traiettoria ottima per il raggiungimento del suo *goal* e si inseriscono nel medesimo ambiente quattro ostacoli con posizioni note a priori.

Si analizza in primo luogo l'algoritmo che utilizza la funzione che restituisce vincoli non lineari (5.4); dall'esperimento ricaviamo i risultati mostrati in Figura 5.6 in termini di posizione, angoli e velocità. Dai grafici rappresentati in Figura 5.6 si vede come anche utilizzando questo algoritmo per il problema di *obstacle avoidance* non si ha nessun tipo di inconveniente, se non il fatto di ottenere un problema non convesso. Infatti è garantito il corretto funzionamento dell'agente in quanto la traiettoria di riferimento è ben inseguita dal sistema reale. Ciò è ben visibile dal primo grafico in alto a sinistra di Figura 5.6, dove i quadratini blu rappresentano il segnale di riferimento, mentre la posizione reale dell'agente è indicata tramite asterischi rossi che restano sempre all'interno del riferimento garantendo così un ottimo inseguimento della traiettoria da parte dell'agente reale. Anche il secondo grafico di Figura 5.6, quello in alto a destra, fornisce informazioni circa la posizione dell'agente; in esso infatti si confronta la posizione reale del *robot*, ovvero quella acquisita dalla *webcam* che sarà effettivamente utilizzata per implementare l'algoritmo di controllo, con la stima, rappresentata in rosso, che segue abbastanza fedelmente la posizione reale, risultando quindi essere molto buona. Nel grafico in basso a sinistra viene invece

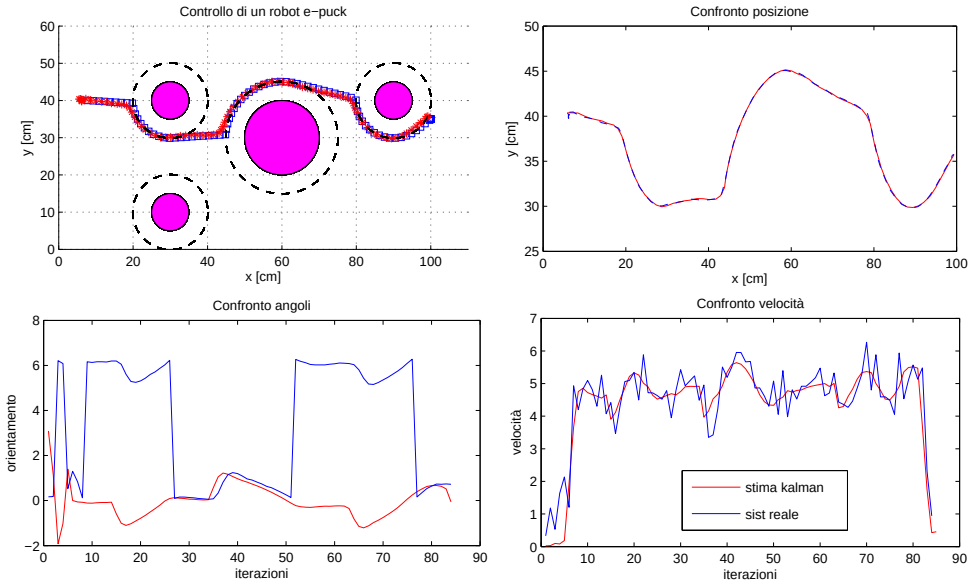


Figura 5.6: *Obstacle avoidance* con vincoli non lineari

rappresentato il confronto tra l'angolo stimato e l'angolo θ acquisito direttamente dalla telecamera; come già discusso precedentemente, in questo caso si avranno problemi per quanto riguarda la stima dell'angolo che rappresenta l'orientamento del *robot*. Infatti il sistema a partire dal segnale stimato non è in grado di riconoscere se l'agente è orientato correttamente verso il *goal* o ruotato nel verso opposto di $\pm 180°$. Questo rappresenta una criticità, ma dimostra che quanto detto in precedenza è veritiero, ovvero che l'algoritmo di controllo deve essere implementato utilizzando obbligatoriamente l'angolo θ reale, ovvero quello direttamente misurato dalla *webcam*. Infine nel grafico in basso a destra viene confrontata la velocità del sistema reale, che ricordiamo essere calcolata con derivata discreta della posizione misurata tramite *webcam* rispetto a due punti successivi, con la sua stima; si conferma che la stima risulta essere buona, e infatti essa è utilizzata come visto per implementare l'azione di controllo. In conclusione, dai dati ricavati, come era presumibile, è possibile affermare che il problema dell'*ostacle avoidance* trova una buona soluzione anche nel caso venga utilizzato l'algoritmo basato su vincoli non lineari (5.4), infatti esso non porta alla generazione di nessun tipo particolare di problema.

Si passa ora ad analizzare l'algoritmo che utilizza vincoli lineari (5.5) per evitare la collisione tra *robot* e ostacoli noti cercando di fare un confronto con l'algoritmo precedentemente analizzato; esso permette di ottenere i risultati mostrati in Figura 5.7. Anche utilizzando questo algoritmo le osservazione che si possono fare sono analoghe al caso precedente; fattore importante è che anche con questo algoritmo, che restituisce vincoli di tipo lineare generando così un problema convesso e che quindi ci consente di avere tutta una serie di vantaggi computazionali, è possibile garantire un ottimo inseguimento della traiettoria di riferimento conducendo in modo corretto l'agente al suo *goal*.

Infine si confrontano nella Figura 5.8 i tempi computazionali richiesti da ogni iterazione del ciclo di controllo dei due diversi algoritmi. Dalla figura si osserva, come già anticipato, che le iterazioni dell'algoritmo per l'*obstacle avoidance,* che utilizza vincoli lineari, impiegano mediamente un tempo computazionale leggermente minore

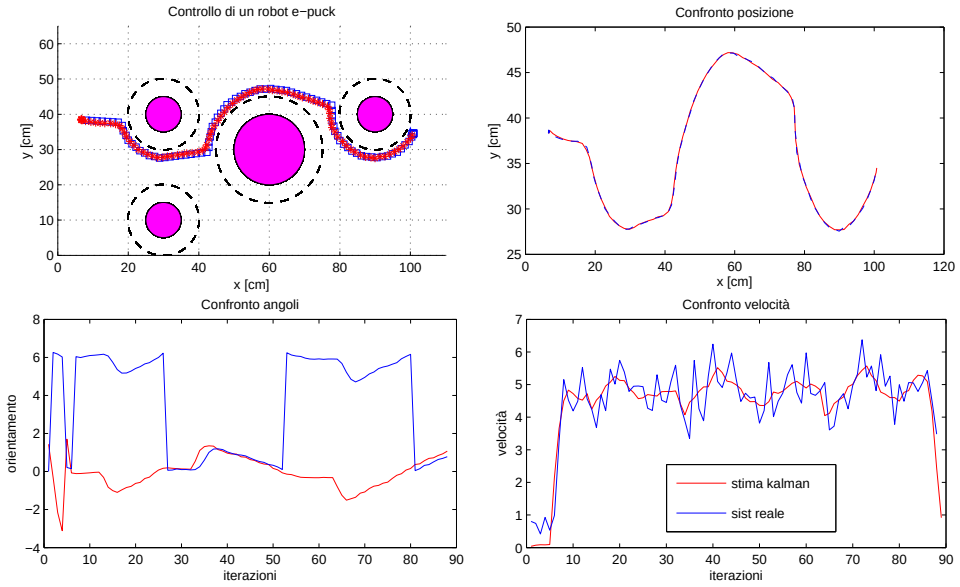


Figura 5.7: *Obstacle avoidance* con vincoli lineari

rispetto a quelle dell'algoritmo che lavora con vincoli non lineari; tuttavia tali tempi sono comparabili e ciò, come visto, permette di ottenere una soluzione ammissibile al problema in ambedue i casi. Da notare inoltre che nella Figura 5.8 è stata trascurata per entrambi gli algoritmi, che impiegano lo stesso numero di iterazioni se viene mantenuta per l'agente la medesima posizione iniziale e lo stesso *goal*, l'iterazione iniziale in cui il sistema effettua l'acquisizione dell'immagine e la connessione con l'agente, in quanto essa occupa un tempo computazionale molto maggiore e quindi non comparabile con le altre iterazioni.

5.3 Collision Avoidance

Si consideri ora il caso della *collision avoidance*, che consiste nel fare in modo che diversi *robot* presenti nello stesso spazio operativo non si scontrino tra loro durante l'inseguimento delle proprie traiettorie; per ulteriori e più approfondite analisi si rimanda a [11]. Un grande aiuto nell'applicazione di un algoritmo che risolva questo tipo di problema viene dato, come anticipato in precedenza, dal controllo predittivo; infatti grazie a esso è possibile prevedere le traiettorie che avranno i vari agenti negli istanti futuri, consentendo in questo modo ai diversi *robot*, all'istante attuale, di prendere le dovute precauzioni per far sì di non collidere tra loro e allo stesso tempo di continuare comunque il loro inseguimento della traiettoria ottimale verso il *goal*. In sintesi, grazie a questo concetto datoci dall'approccio predittivo, è possibile vedere i *robot* come degli ostacoli mobili, inscritti in una circonferenza che costituisce la regione di sicurezza che gli altri agenti non possono invadere, permettendo in questo modo di far sì che essi non collidano tra loro. Dopo tale osservazione si capisce che è possibile quindi applicare lo stesso ragionamento utilizzato dall'algoritmo per l'*obstacle avoidance* anche per questo tipo di problema che può quindi essere risolto imponendo un vincolo aggiuntivo di distanza minima fra i vari agenti presenti nello spazio operativo al problema di ottimizzazione (4.22).

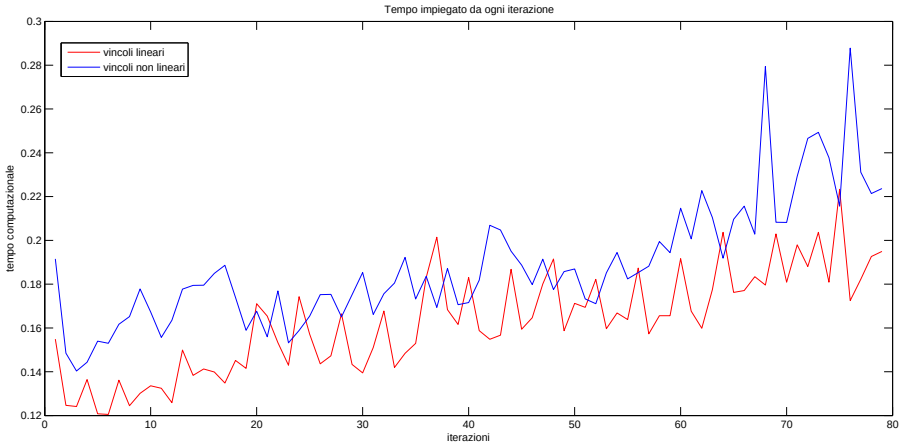


Figura 5.8: Tempo impiegato da ogni singola iterazione del ciclo operativo per risolvere il problema di *obstacle avoidance*

Data una coppia di *robot* i e j diremo infatti che essi riusciranno a risolvere il problema di *collision avoidance* se, durante il loro movimento, essi saranno in grado di prevenire ed evitare collisioni tra di loro. A tale scopo, per ogni agente $i = 1, \ldots, M$ considerato operante nel piano di lavoro, definiamo il suo *set* di *robot* vicini come:

$$\mathscr{C}_i \subseteq \mathscr{N} \setminus \{i\}$$

con i quali la *collision avoidance* deve essere evitata. Per coerenza assumiamo che $j \in \mathscr{C}_i$ se e solo se $i \in \mathscr{C}_j$. Anche se il problema di *collision avoidance* risulta essere, come detto, simile a quello dell'*obstacle avoidance* esso presenterà una maggiore difficoltà risolutiva. Infatti, le principali criticità risiederanno nel fatto che le varie coppie di agenti dovranno riuscire a coordinare i propri movimenti. In questo modo ogni *robot* dovrebbe essere in grado di gestire, in modo parallelo e non sequenziale, i propri vincoli così da evitare tutte le possibili collisioni.

Assumiamo ora che tramite una rete di comunicazione formata tra i vari *robot* vicini, il valore $\tilde{y}^{[j]}_{k+N-1}$ sia disponibile per l'agente i e che $\tilde{y}^{[i]}_{k+N-1}$ sia disponibile per il *robot* j; definiamo a questo punto:

$$\delta_{ij} = \max_{\delta y_i \in \mathscr{Y}_i} \| \delta y_i \| + \max_{\delta y_j \in \mathscr{Y}_j} \| \delta y_j \| = \delta_i + \delta_j$$

che sarà una distanza opportunamente calcolata. Per garantire *collision avoidance* è necessario dunque implementare un algoritmo che, a ogni istante k, garantisca:

$$\| y^{[i]}_{k+N} - y^{[j]}_{k+N} \|_2 \geq 2 \cdot R_{robot}$$

Ciò, in somiglianza per il problema di *obstacle avoidance*, è raggiunto se

$$\| \tilde{y}^{[i]}_{k+N} - \tilde{y}^{[j]}_{k+N} \|_2 \geq 2 \cdot R_{robot} + \delta_{ij} \tag{5.6}$$

Tuttavia, richiedere (5.6) non è né possibile né sufficiente a garantire la fattibilità ricorsiva di tale vincolo poiché sia $\tilde{y}^{[i]}_{k+N}$ che $\tilde{y}^{[j]}_{k+N}$ sono variabili di ottimizzazione, gestite da due differenti *robot* allo stesso istante. In questo caso infatti, rispetto

all'*obstacle avoidance,* la situazione è nettamente più delicata in quanto l'algoritmo che utilizza vincoli non lineari (5.6), oltre a fornire un problema non convesso, genera innumerevoli criticità sia in termini di logica che di computazione. Infatti il metodo richiede un tempo di computazione elevato che spesso risulta essere addirittura maggiore del tempo di campionamento dell'anello per il controllo predittivo. Ciò può quindi frequentemente generare criticità nell'implementazione dell'intero algoritmo di controllo. Si è quindi dovuta cercare una nuova soluzione che permetta di evitare tali problemi e che allo stesso tempo consenta di aumentare anche l'efficienza del sistema. La soluzione trovata si basa sulla riformulazione dei vincoli (5.6) in termini lineari; per far ciò dovremo nuovamente ricorrere alla rappresentazione del *set* politopico già utilizzata in precedenza nel paragrafo relativo all'*obstacle avoidance.* Grazie a essa riusciamo infatti, anche in questo caso, ad approssimare la circonferenza definita da (5.6) attraverso un insieme di r_{ij} disequazioni lineari:

$$\left(h_t^{[ij]}\right)^T \left(\tilde{y}_{k+N}^{[i]} - \tilde{y}_{k+N}^{[j]}\right) \leq d_{ij}$$

con $t = 1, \ldots, r_{ij}$. Definiamo inoltre a questo punto

$$t_{max}^{[ij]}(k+N-1) = \underset{t \in \{1,\ldots,r_{ij}\}}{argmax} \left(h_t^{[ij]}\right)^T \left(\tilde{y}_{k+N-1}^{[i]} - \tilde{y}_{k+N-1}^{[j]}\right) - d_{ij} \tag{5.7}$$

e

$$\rho_{max}^{[ij]}(k+N-1) = \max_{t \in \{1,\ldots,r_{ij}\}} \left(h_t^{[ij]}\right)^T \left(\tilde{y}_{k+N-1}^{[i]} - \tilde{y}_{k+N-1}^{[j]}\right) - d_{ij} \tag{5.8}$$

Ricordando ora che le variabili di ottimizzazione per i *robot* i e j sono rispettivamente $\bar{y}_{k+N}^{[i]}$ e $\bar{y}_{k+N}^{[j]}$ ed è quindi possibile imporre, all'istante k, i seguenti vincoli per i due agenti:

$$\left(h_{t_{max}^{[ij](k+N-1)}}^{[ij]}\right)^T \left(\bar{y}_{k+N}^{[i]} - \tilde{y}_{k+N-1}^{[j]}\right) \geq d_{ij} + \rho_{max}^{[ij]}(k+N-1)/2 \tag{5.9}$$

$$\left(h_{t_{max}^{[ij](k+N-1)}}^{[ij]}\right)^T \left(\tilde{y}_{k+N-1}^{[i]} - \bar{y}_{k+N}^{[j]}\right) \geq d_{ij} + \rho_{max}^{[ij]}(k+N-1)/2 \tag{5.10}$$

Anche in questo la soluzione adottata in [13] si basa sulla formulazione di vincoli di tipo non lineare (5.6) e essa non viene descritta in quanto risulta essere analoga a quella vista in precedenza relativa all'*obstacle avoidance.*

5.3.1 Descrizione dell'algoritmo specifico

Si presenta ora dettagliatamente l'algoritmo utilizzato per risolvere il problema di *collision avoidance* tra coppie di *robot* operanti nello stesso piano di lavoro riferito al caso di vincoli lineari (5.9) e (5.10). Si definiscano con (x_1, y_1) e (x_2, y_2) le coordinate delle posizioni dei due *robot* i e j, e sia d_{ij} la distanza minima desiderata tra i due agenti, che dipenderà, anche in questo caso, dal raggio R_{robot} dei due *robot e-puck* e da una certa distanza limite δ_{ij} opportunamente calcolata come visto in precedenza. Sia inoltre $N = r_{ij}$ il numero di lati di un politopo per il quale valgono le stesse osservazioni fatte precedentemente per il caso dell'*obstacle avoidance.* Il metodo realizzato permette quindi di riformulare i vincoli di distanza massima come due vincoli lineari (5.9) e (5.10), uno per ogni *robot.* Si può infatti pensare che ogni

agente sia inscritto in una circonferenza di raggio d la quale, a sua volta, è inscritta in un politopo formato da un certo numero di rette N, ognuna delle quali rappresentata da un vincolo lineare r_{ij}. Anche in questo caso si utilizza per l'algoritmo implementato un valore di $N = 20$ avendo così un ottimo compromesso tra la buona approssimazione del poligono a una circonferenza e la bassa probabilità che il *robot*, in presenza di un ostacolo, vada a sbattere lungo un lato N del poligono in modo perpendicolare, incorrendo nel problema mostrato nella Figura 5.3 (b).

Inizialmente l'algoritmo calcola le equazioni lineari che descrivono le rette su cui giacciono i lati del poligono regolare, distanti d dal centro del poligono stesso. In questo modo si hanno equazioni del tipo:

$$d = (x - x_o)\cos\theta + (y - y_o)\sin\theta$$

ovvero in forma matriciale:

$$d = \begin{bmatrix} \cos\theta & \sin\theta \end{bmatrix} \begin{bmatrix} x - x_o \\ y - y_o \end{bmatrix}$$

dove θ e il punto di coordinate (x_o, y_o) assumono la stessa definizione enunciata in precedenza per il caso di *obstacle avoidance* come mostrato in Figura 5.4. Successivamente si definisce il poligono cercato attraverso la disuguaglianza:

$$H\,(x - x_o) \leq d$$

, dove la matrice H è definita come segue:

$$H = \begin{bmatrix} \cos\theta_1 & \cdots & \cos\theta_N \\ \sin\theta_1 & \cdots & \sin\theta_N \end{bmatrix}$$

e

$$\theta_n = \frac{2\pi(k-1)}{N}$$

con $0 < k \leq N$ ed $0 < n \leq N$. A questo punto non resta che isolare i vincoli più appropriati relativamente ai due agenti; andando infatti ad analizzare tutte le rette appartenenti al politopo si sceglie quella che risulta essere più vicina alle posizioni attuali dei due *robot,* rappresentate, come detto, dalle coordinate (x_1, y_1) e (x_2, y_2). Questo viene fatto, dal punto di vista analitico, definendo un vettore riga J di N colonne, che dipende direttamente dalla posizione attuale dei *robot*, nel seguente modo:

$$J = \Bigg[\begin{bmatrix} \cos\theta_1 & \sin\theta_1 \end{bmatrix} \begin{bmatrix} x_1 - x_2 \\ y_1 - y_2 \end{bmatrix} - d \cdots$$

$$\cdots \begin{bmatrix} \cos\theta_N & \sin\theta_N \end{bmatrix} \begin{bmatrix} x_1 - x_2 \\ y_1 - y_2 \end{bmatrix} - d \Bigg]$$

andando appunto a scegliere, tra le varie componenti del vettore riga J, quella che presenta parte positiva maggiore, indicata con t_{max}, dove $1 \leq t \leq N$, e corrispondente valore in modulo ρ_{max}, come definito in (5.7) e (5.8). Supponendo quindi che, all'istante k, l'elemento massimo di J si il t-esimo, all'istante $k+1$ i seguenti vincoli verranno imposti:

$$\begin{cases} Aca_{12}x_{k+1} \leq & bca_{12} \\ Aca_{21}x_{k+1} \leq & bca_{21} \end{cases}$$

, dove

$$\begin{cases} Aca_{12} = & -\begin{bmatrix} \cos\theta_k & \sin\theta_k \end{bmatrix} \\ bca_{12} = & -\begin{bmatrix} \cos\theta_k & \sin\theta_k \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} - d - \left(\frac{\rho}{2}\right) \end{cases}$$

e

$$\begin{cases} Aca_{21} = & \begin{bmatrix} \cos\theta_k & \sin\theta_k \end{bmatrix} \\ bca_{21} = & \begin{bmatrix} \cos\theta_k & \sin\theta_k \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} - d - \left(\frac{\rho}{2}\right) \end{cases}$$

L'algoritmo presentato si basa quindi sulla seguente idea:

- si individua, data la posizione del *robot* all'istante k, il lato del poligono da cui l'agente, avente posizione (x_{a_k}, y_{a_k}), si trova a maggiore distanza, cioè l'elemento di J (positivo) avente valore maggiore;
- all'istante successivo $k+1$, per garantire *collision avoidance,* la violazione del vincolo individuato in precedenza verrà imposta nel problema di ottimizzazione.

Facendo ciò è possibile garantire, ricorsivamente, che i due agenti considerati non entrino mai in collisione tra loro, risolvendo così il problema di *collision avoidance*. Nella Figura 5.9 viene mostrato in maniera grafica e intuitiva come opera l'algoritmo appena presentato.

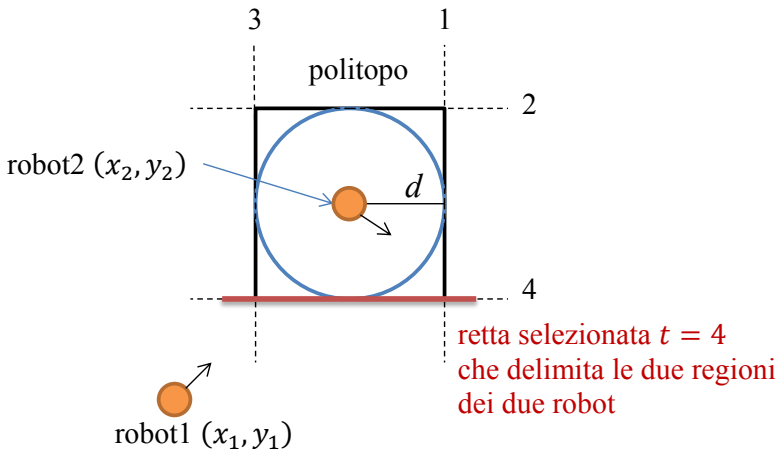


Figura 5.9: Algoritmo per il *collision avoidance* che utilizza vincoli lineari (caso $N = 4$)

L'algoritmo dovrà poi essere applicato per tutte le coppie di agenti inserite nello spazio di lavoro in modo così da garantire una corretta risoluzione del problema di *collision avoidance* complessivo. Ricaviamo così le componenti Aca_{RiRj}, Aca_{RjRi}, bca_{RiRj}, bca_{RjRi} per ogni coppia di agenti (i, j) presa in considerazione. È interessante inoltre notare che l'algoritmo così sviluppato assicura una corretta risoluzione del problema di *collision avoidance* in modo biunivoco per la singola coppia di *robot*

considerata. Anche in questo caso andremo poi a inserire negli algoritmi di controllo, oltre che il vincolo sulla posizione attuale del *robot* espressa in coordinate (x, y) acquisita dalla *webcam* e sulle velocità v_x e v_y stimate tramite predittore di *Kalman*, le componenti ricavate relative ai vari accoppiamenti tra gli agenti in modo da permettere così che la coppia di *robot* (i, j) considerata non entri mai in collisione.

5.3.2 Risultati sperimentali

Si presenta ora un *test* in cui vengono confrontati i risultati ottenuti tramite l'algoritmo che utilizza vincoli non lineari con quello che si basa invece su vincoli lineari. Le condizioni di lavoro sono le seguenti: si hanno tre agenti che operano nello stesso spazio di lavoro seguendo la traiettoria ottima per il raggiungimento dei loro *goal*; essi sfruttano il concetto offerto dalla teoria predittiva per fare in modo di non scontrarsi anche se dovessero intersecarsi le loro traiettorie.

Impiegando in primo luogo l'algoritmo che si basa su vincoli non lineari per far sì che il *robot* non collida con gli altri agenti si ricava quanto già spiegato in precedenza, ovvero che a causa dei problemi enunciati, dati principalmente dall'utilizzo di vincoli non lineari, non si riesce a ottenere una soluzione ammissibile in un tempo sufficientemente breve per una efficace implementazione *on-board*. A prova di ciò si mostra ora, nella Figura 5.10 il tempo computazionale impiegato da ogni iterazione per i due algoritmi. Dalla figura si vede che il tempo computazionale per il problema

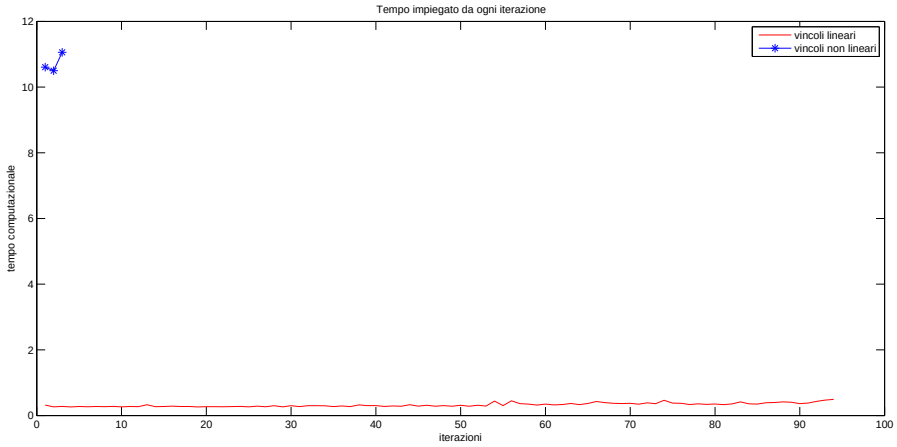


Figura 5.10: Tempo impiegato da ogni singola iterazione del ciclo operativo per risolvere il problema di *collision avoidance*

che utilizza vincoli non lineari risulta molto più alto di quello che utilizza vincoli lineari. Inoltre tale valore risulta essere non ammissibile per il ciclo di controllo, infatti appena la posizione reale dei *robot* si discosta eccessivamente dalla traiettoria di riferimento predetta dal controllo *DPC*, cosa che avviene dopo tre iterazioni, l'algoritmo va in *crash*. Questo causa notevoli pericoli in quanto si perde completamente il controllo degli agenti, che potrebbero anche collidere tra loro all'interno del piano di lavoro. È interessante inoltre notare che il tempo computazionale di ogni iterazione, nel caso di utilizzo di vincoli non lineari, si dimezza o raddoppia a seconda che venga rispettivamente tolto o inserito un agente. Come quindi annunciato si può conclu-

dere che questo algoritmo per il problema di *collision avoidance* non è utilizzabile in quanto non in grado di risolvere e fornire una soluzione al problema.

Si passa ora ad analizzare l'algoritmo che sfrutta vincoli lineari per evitare la collisione tra i diversi *robot*; questo nuovo algoritmo implementato permette di ottenere i risultati in termini di posizione, angoli e velocità mostrati in Figura 5.11. Dai primi

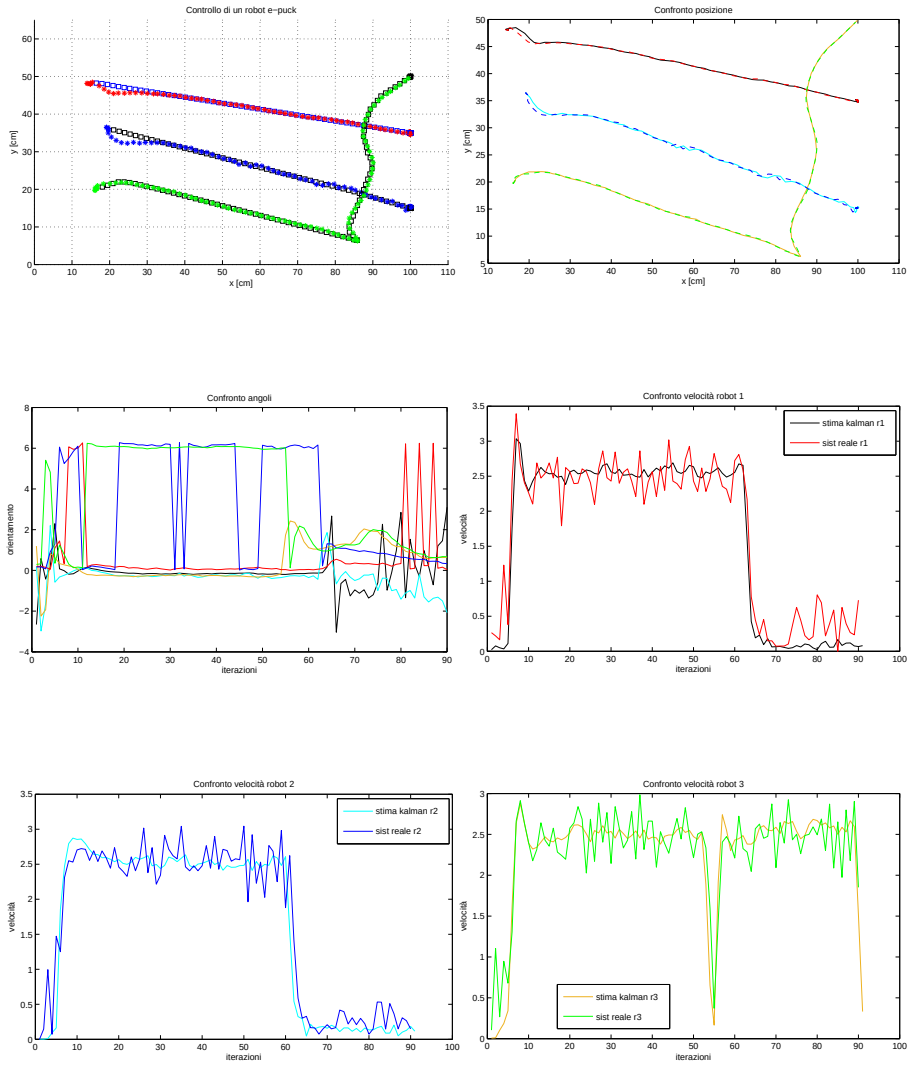


Figura 5.11: *Collision avoidance* con vincoli lineari

due grafici della Figura 5.11 si osserva che i tre agenti riescono a raggiungere i propri *goal* seguendo in modo fedele il riferimento, a parte qualche discrepanza alla partenza dovuta ai problemi analizzati in precedenza. Inoltre, grazie all'algoritmo appena descritto i *robot* sono in grado di effettuare, senza collisioni, incroci di traiettoria. Andando infatti a regolare a ogni istante le velocità dei tre agenti si fa in modo che i vincoli di distanza tra i vari *robot* siano rispettati risolvendo così il problema di *collision avoidance*; dagli ultimi tre grafici della Figura 5.11 è infatti possibile vedere gli andamenti delle velocità indipendenti che assumono i tre agenti. Da notare come il

robot 3 (il verde) per non collidere con gli altri due agenti decrementi la sua velocità all'istante 56 per poi raggiungere successivamente il suo obiettivo; dai grafici delle velocità è inoltre visibile che i primi due *robot* (il rosso e il blu) arrivano ai propri *goal* entrambi circa all'iterazione 62 e permangono in esso con velocità circa nulla in attesa che anche il terzo agente, il *robot* verde, raggiunga il proprio obiettivo. Si può quindi essere soddisfatti dei risultati ottenuti in quanto si è riusciti a fornire, per il problema affrontato, una soluzione affidabile, scalabile ed efficiente.

5.4 Utilizzo congiunto dei due algoritmi

Nel caso in cui nel piano di lavoro siano presenti contemporaneamente sia più agenti sia un certo numero di ostacoli noti, verranno simultaneamente utilizzati entrambi gli algoritmi appena descritti per l'*obstacle* e per il *collision avoidance.* Tale fatto non costituisce un problema; infatti i due algoritmi presentati possono essere applicati contemporaneamente. Tuttavia, per tenere in considerazione entrambi gli effetti, si dovranno combinare/unire i vari vincoli. Per ogni agente si andranno quindi a ricavare i vincoli complessi formati dalle componenti già analizzate nei precedenti due paragrafi, che saranno questa volta dei vettori colonna:

$$A_{Ri} = \begin{bmatrix} A_{obst} \\ Aca_{RiRj} \end{bmatrix} \quad , \quad b_{Ri} = \begin{bmatrix} b_{obst} \\ bca_{RiRj} \end{bmatrix}$$

e

$$A_{Rj} = \begin{bmatrix} A_{obst} \\ Aca_{RjRi} \end{bmatrix} \quad , \quad b_{Rj} = \begin{bmatrix} b_{obst} \\ bca_{RjRi} \end{bmatrix}$$

Si ricavano così tutti i vincoli complessivi di cui si ha bisogno per garantire una soluzione sia al problema dell'*obstacle* che del *collision avoidance;* una volta fatto ciò li si andranno poi a implementare nei corretti algoritmi di controllo come visto in precedenza.

L'algoritmo proposto per risolvere entrambi i problemi di *avoidance* ha dunque un piccolo impatto sulla complessità del problema di ottimizzazione (4.22) di ogni *robot.* Infatti, a ogni istante k i vincoli lineari verificati sono n_i^o, per quanto riguarda il problema di *obstacle avoidance*, e $| \mathscr{C}_i | \leq M - 1$, per quanto riguarda il problema di *collision avoidance* considerato. Inoltre notiamo che il numero di vincoli in (4.22) non dipende dal numero di disequazioni utilizzate per definire i politopi.

Assumiamo ora che, all'istante k, il problema di ottimizzazione (4.22) sia ammissibile e abbia soluzione $\bar{y}_{k+N|k}^{[i]}$ per tutti i *robot* $i = 1, \ldots, M$ così che (5.4) (o, alternativamente, (5.5)) sia verificato per tutti gli ostacoli $h \in \mathscr{O}_i$, (5.9) sia verificato per tutti $j \in \mathscr{C}_i$ e $\bar{y}_{k+N|k}^{[i]} \in \tilde{y}_{k+N-1}^{[i]} \oplus \mathscr{B}_{q,\varepsilon_i}^{(2)}(0)$; allora è possibile dimostrare che, se all'istante k vengono garantite le non collisioni sia con ostacoli $h \in \mathscr{O}_i$ sia con altri agenti vicini $j \in \mathscr{C}_i$, all'istante $k+1$ il problema di ottimizzazione (4.22) è ammissibile per tutti gli agenti $i = 1, \ldots, M$ considerati.

Questa proposizione appena enunciata risulta essere molto importante in quanto ci consente di garantire la ricorsiva ammissibilità del problema di ottimizzazione (4.22). Comunque, possibili situazioni di *deadlock*, cioè soluzioni dalle quali non è possibile minimizzare ulteriormente la funzione di costo a causa della rigidezza dei vincoli, possono essere incontrate e, a meno di esse, come discusso dettagliatamente in [21], lo schema di controllo *DPC* presentato garantisce convergenza degli agenti verso i rispettivi *goal.*

5.4.1 Risultati sperimentali

Si effettua ora un *test* in cui si dimostra che i risultati ottenuti tramite la combinazione simultanea dei due algoritmi appena visti porta ad avere un sistema efficace nella risoluzione del problema d'interesse. Avendo già discusso ampiamente i vantaggi derivanti dall'uso dei vincoli lineari, verranno implementati solamente i corrispondenti algoritmi. Le condizioni di lavoro sono le seguenti: si hanno tre agenti che operano nel medesimo spazio di lavoro seguendo la traiettoria ottima per il raggiungimento dei loro *goal*. Inoltre sempre in tale spazio operativo sono inseriti due ostacoli circolari con posizione nota fissata a priori; questo problema presenta quindi ostacoli sia fissi che mobili.

Analizzando i risultati ottenuti sempre in termini di posizione, angoli e velocità, come mostrato in Figura 5.12 , è evidente che combinando entrambe le tecniche di

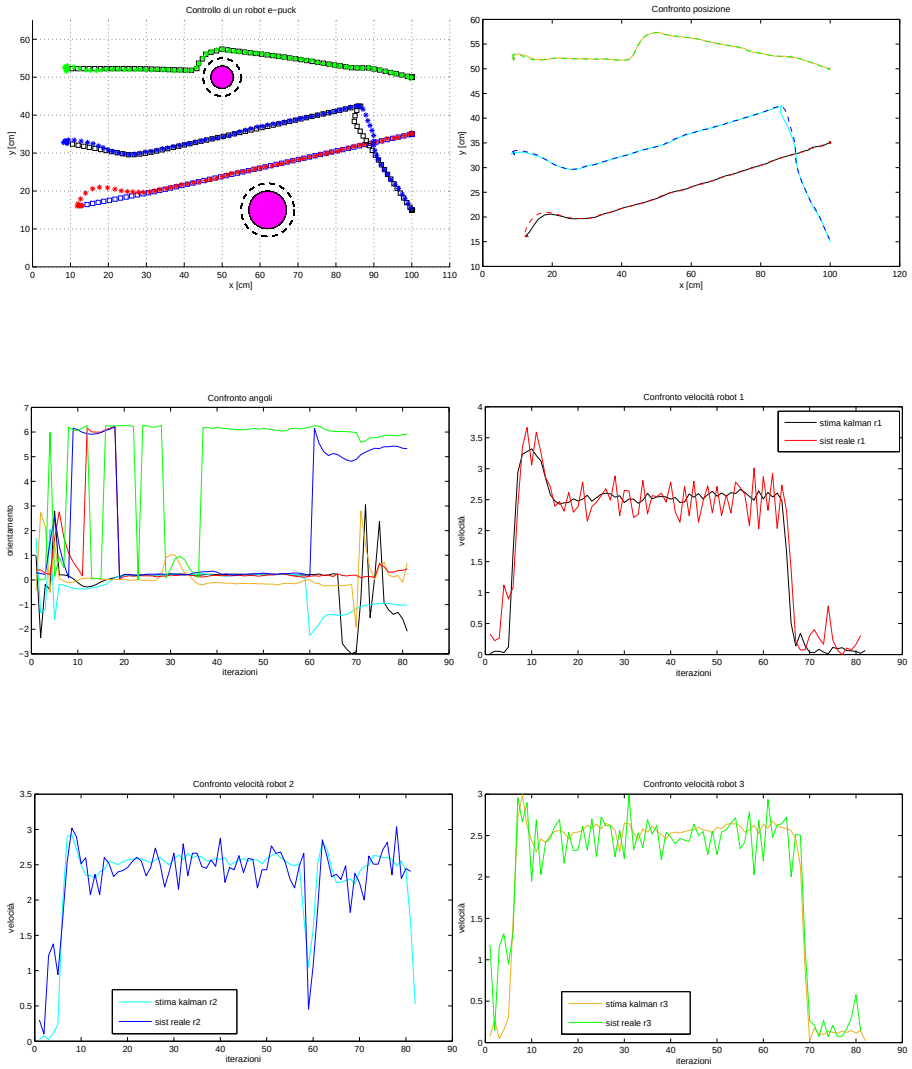


Figura 5.12: Combinazione di *obstacle & collision avoidance*

obstacle e *collision avoidance* nello stesso algoritmo si ottengono dei risultati molto soddisfacenti e, cosa più importante, è garantita la corretta risoluzione del problema, infatti i tre agenti arrivano ai propri obiettivi finali senza collidere né tra di essi né con gli ostacoli fissati. Dai primi tre grafici di Figura 5.12 si possono ricavare le solite osservazioni sulla capacità dei *robot* di inseguire il loro corrispondente *setpoint* e sul fatto che la stima degli angoli non rispecchi il corretto orientamento dei tre agenti. L'unica particolarità degna di nota è data dal fatto che, analizzando il primo grafico di Figura 5.12, l'agente blu (*robot* 2) alle coordinate di $(84, 42)$ si discosta dal suo riferimento in quanto deve effettuare una brusca decelerazione. Questo fenomeno è visibile molto bene dal grafico della velocità, sempre mostrato in Figura 5.12, relativo al secondo agente, in cui si osserva che circa all'iterazione 60 si ha una netta diminuzione della velocità dovuta al fatto che il *robot* 2 (il blu) deve aspettare il passaggio del *robot* 1 (il rosso) per far sì che i due agenti non collidano. Dagli ultimi tre grafici si può osservare come l'algoritmo agisca appunto sulle velocità, facendo in modo che esse vengano opportunamente regolate quando gli agenti sono al cospetto di un ostacolo, sia fisso che mobile ed è proprio grazie alla corretta modulazione di tali segnali che tutto funziona senza problemi e in maniera ottimale.

Capitolo 6

Metodo dei Potenziali Artificiali

Un capitolo a parte riguardante i problemi di navigazione autonoma di agenti mobili viene riservato alla tecnica dei potenziali artificiali, che costituisce un metodo alternativo alla risoluzione dei problemi di *obstacle* e *collision avoidance* rispetto a quelli presentati al Capitolo 5. La scelta di non inserire tale metodologia all'interno del Capitolo 5 è dettata dal fatto che, per lo sviluppo di questa tecnica, non viene utilizzato un controllo basato su teoria predittiva, bensì si andrà a sviluppare un algoritmo indipendente da tutti i concetti visti fino ad ora e descritto in [20].

6.1 Introduzione

Diamo ora una visione generale di una strategia, molto diffusa in letteratura, che utilizza appunto il concetto di potenziale artificiale *APF* (*Artificial Potential Field)* per risolvere problemi di *avoidance* nell'ambito della navigazione autonoma di agenti mobili. Con tale tecnica si va a considerare il *robot* mobile come una particella sottoposta all'influsso di un campo potenziale artificiale, le cui variazioni rispecchiano la conformazione e la struttura del piano di lavoro a seconda del posizionamento dell'obiettivo finale e dei vari ostacoli presenti nell'ambiente circostante. Nello specifico avremo che tale campo è costituito da un potenziale attrattivo, generato dall'obiettivo che il *robot* deve raggiungere, mentre ciascun ostacolo, sempre considerato di forma circolare e localizzato nel piano di lavoro, conferisce un potenziale repulsivo di ampiezza data dal proprio raggio.

Sulla base della definizione data si nota come, grazie al concetto di potenziale artificiale definito sul piano di lavoro, si riesca a garantire sia la risoluzione dei problemi di *avoidance*, sia, allo stesso tempo, venga fornita una valida strategia di controllo per la movimentazione del *robot* verso l'obiettivo impostato (problema di *tracking*). In particolare, il controllo dell'agente è dato dal fatto che il potenziale artificiale, presente nello spazio operativo, genera e definisce un campo vettoriale formato da gradienti di velocità, che consentono di guidare il *robot,* evitando gli ostacoli, fino al raggiungimento del proprio *goal.*

Un modo analogo per vedere il problema consiste nell'approccio di tipo energetico: è infatti facile notare come il gradiente del campo vettoriale possa essere visto, in ciascun punto, come una forza. Supponendo quindi che il *robot* sia sensibile al potenziale imposto nello spazio di lavoro, avremo che, una volta immerso nel campo, esso risulta soggetto all'azione di varie forze che lo guidano verso l'obiettivo per

effetto della componente attrattiva generata dal *goal*, mentre lo spingono lontano dagli ostacoli per effetto della componente repulsiva dovuta al campo da essi generato.

Analiticamente bisognerà dunque definire una funzione potenziale come la somma di due funzioni, una caratterizzata da un potenziale attrattivo, data dal *goal,* e l'altra descritta da un potenziale repulsivo, data dai vari ostacoli presenti sulla superficie di lavoro. Il potenziale attrattivo solitamente è rappresentato tramite una funzione euclidea di tipo quadratico che rappresenta la distanza euclidea, in linea d'aria, tra tutti i punti e l'obiettivo. Essa è definita in modo tale che generi una forza attrattiva che va ad attenuarsi man mano ci si avvicina al *goal,* all'obiettivo infatti essa sarà nulla. Il potenziale repulsivo, che viene calcolato per ogni ostacolo presente, sarà invece costruito in modo da assumere un valore elevato quando il *robot* è vicino agli ostacoli mentre decrescerà fino ad annullarsi man mano che questo si allontana da essi. Il potenziale repulsivo così definito consente quindi di produrre sul piano di lavoro delle superfici repulsive, che dovranno essere opportunamente pesate, in base al raggio dell'ostacolo, in modo da non andare ad alterare più del dovuto la traiettoria del *robot* verso l'obiettivo. Il campo potenziale, così definito, potrà dunque agire da funzione di navigazione per la guida dei vari agenti all'interno del proprio spazio operativo; a tal proposito viene riportato in Figura 6.1 un esempio di campo potenziale definito su una superficie.

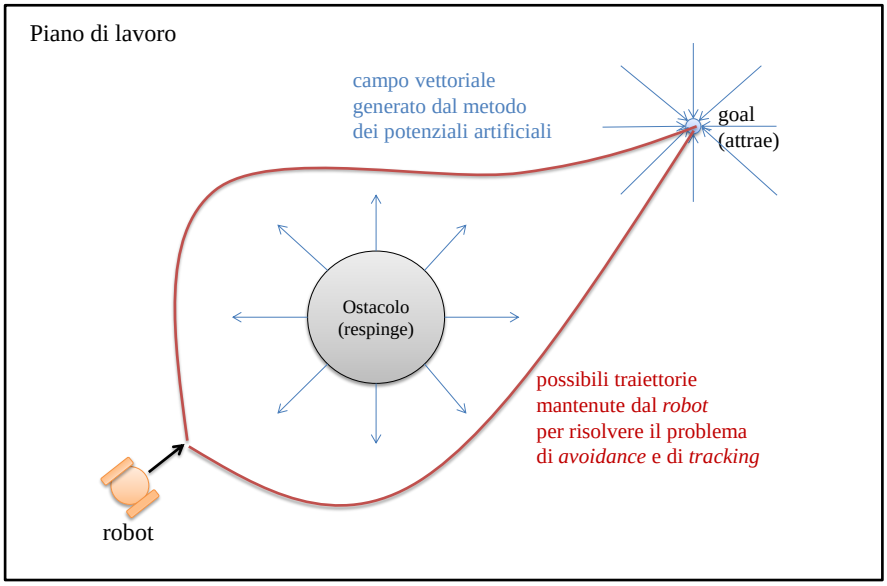


Figura 6.1: Campo vettoriale generato sulla superficie di lavoro dal metodo dei potenziali artificiali

Un ulteriore modo per vedere il problema analizzato è quello di considerare il potenziale distribuito sulla superficie di lavoro come un profilo costituito da diverse curve di livello. Nel dettaglio si ha che i potenziali repulsivi, dati dagli ostacoli, sono rappresentati come dei rilievi ovvero con delle curve di livello che presentano un'altezza positiva, mentre il *goal* è visto come una depressione della superficie operativa

nella quale si andrà a ricadere ed è quindi descritto da curve di livello con altezza negativa. In Figura 6.2 viene appunto riportato un esempio dell'approccio appena descritto.

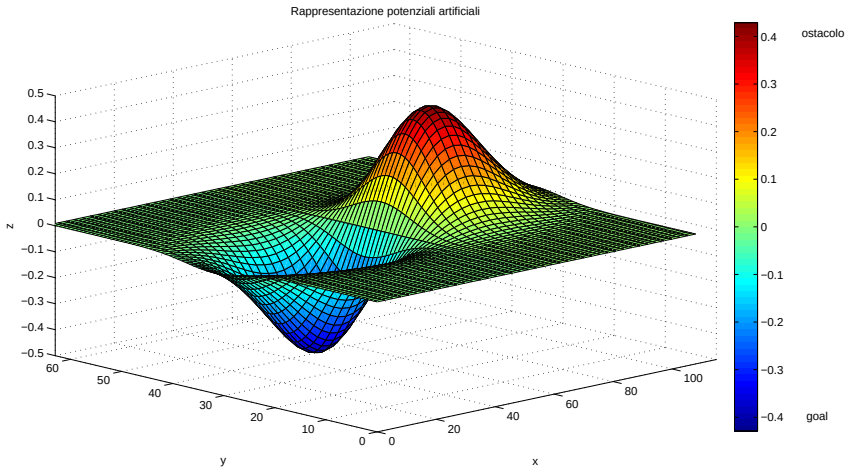


Figura 6.2: Esempio di una funzione potenziale definita sulla superficie in esame con approccio basato su curve di livello

Ipotizziamo ora che il *robot* operi, come nel nostro caso, in uno spazio 2-D, ovvero in un piano su cui è fissato un determinato sistema di riferimento che permette di individuare sempre la posizione dell'agente tramite coordinate (x, y) definite in un vettore q. Il campo potenziale all'interno del quale si muove l'agente sarà, come detto, una funzione scalare $U(q)$ da $\mathbb{R}^2$ in $\mathbb{R}$ generata dalla sovrapposizione di potenziali repulsivi e attrattivi, dovuti rispettivamente agli ostacoli e all'obiettivo:

$$U(q) = U_a(q) + U_r(q)$$

A sua volta il potenziale repulsivo può essere generato dalla sovrapposizione dei singoli effetti repulsivi dovuti a ciascun ostacolo presente nell'area di lavoro e pertanto può essere scritto come:

$$U_r(q) = \sum_{i=1}^{N_{obst}} U_{r_i}(q)$$

dove indichiamo con U_{r_i} il potenziale generato dall'i-esimo ostacolo. In modo completo la funzione potenziale potrà quindi essere scritta come:

$$U(q) = U_a(q) + \sum_{i=1}^{N_{obst}} U_{r_i}(q) \tag{6.1}$$

Supponendo ora che la funzione potenziale (6.1) costruita sia differenziabile, a ogni valore della sua posizione q sarà possibile calcolare il gradiente di tale funzione, che verrà visto come una forza artificiale del tipo:

$$F(q) = -\nabla(U_a(q) + U_r(q)) = F_a(q) + \sum_{i=1}^{N_{obst}} F_{r_i}(q)$$

Tale forza avrà un dato modulo e una data direzione a seconda del peso che avranno le due funzioni U_a e U_r nello specifico punto q di coordinate (x, y) preso in esame. In particolare il vettore che rappresenta tale forza in un punto q assume la direzione nella quale viene localmente minimizzata la funzione potenziale. Tenendo presente tale considerazione otteniamo allora che il *robot* dovrà muoversi sempre in modo da diminuire il più possibile il valore assunto dalla funzione potenziale evitando così gli ostacoli, che come detto presentano un valore di potenziale elevato, fino a raggiungere appunto potenziale uguale a zero nel suo obiettivo.

Tale strategia presentata impone implicitamente anche un vincolo sulla velocità dell'agente che avrà valori elevati lontano dal *goal* e diminuirà progressivamente man mano ci si avvicina a esso. Ovviamente, quindi, più lontano risulta il *robot* dal punto desiderato più grande è l'ampiezza del campo attrattivo a cui esso è soggetto e dunque più elevata sarà la velocità con cui esso viaggia. Viene così imposto che il vettore di velocità sia proporzionale al vettore del campo di forze e ciò fa sì che il *robot* venga guidato in modo autonomo verso l'obiettivo rallentando man mano si avvicina a esso. Il potenziale repulsivo, invece, ha il compito di mantenere l'agente lontano dagli ostacoli garantendo così un percorso privo di collisioni. A tale proposito abbiamo visto che siamo in grado di ottenere questo effetto facendo sì che ogni ostacolo produca localmente un aumento, idealmente infinito, della funzione potenziale dal momento che la navigazione ha come obiettivo la sua minimizzazione.

La tecnica presentata risulta quindi essere di semplice implementazione e costituisce un metodo efficace per risolvere i problemi proposti. Tuttavia il metodo dei potenziali artificiali presenta anche diversi problemi e svantaggi di cui il principale è dovuto alla possibile presenza di minimi locali nella funzione del potenziale $U(\cdot)$ costruita. Un minimo locale si potrebbe presentare in occasione di un perfetto bilanciamento tra il potenziale attrattivo e quello repulsivo, generando così un annullamento del gradiente che potrebbe essere visto dal *robot* come un "falso" *goal*. Se infatti un cammino pianificato entra nel bacino di attrazione di un minimo locale di $U(q_m)$, la pianificazione si blocca e la forza risulta

$$F(q_m) = -\nabla U(q_m) = 0$$

In particolari situazioni, quindi, la presenza di minimi locali diversi dall'obiettivo, ma comunque caratterizzati da un gradiente nullo, possano far si che il *robot* resti intrappolato, senza poter completare il compito assegnato; questo può dunque costituire una forte criticità legata all'uso di questa tecnica.

6.2 Algoritmo di controllo basato su APF

Si vuole ora analizzare e presentare nel dettaglio l'algoritmo sviluppato che utilizza il metodo dei potenziali artificiali, appena descritto, in accordo con quanto riportato in [20] e successivamente implementarlo sui *robot e-puck*. Grazie alla *webcam* assumiamo che ogni agente conosca in modo aggiornato la sua posizione e il suo orientamento; inoltre si ipotizza che, anche in questo caso, gli ostacoli presenti sulla superficie di lavoro abbiano posizione e raggio noti a priori. In questo algoritmo viene utilizzata un'architettura decentralizzata in cui i controllori sono applicati su ogni agente, che risulterà quindi indipendente dagli altri *robot*. Solamente nella risoluzione del problema di *collision avoidance,* che analizzeremo nel prosieguo, si vedrà che gli agenti dovranno interagire tra loro avendo appunto la necessità di comunicare le

proprie posizioni per evitare collisioni. Nello specifico, il problema di *collision avoidance* che, come visto in precedenza, opera in modo *real time* utilizzerà una funzione potenziale, definita localmente, che potrà assumere forme diverse permettendo così a ogni agente di rilevare oggetti nelle sue vicinanze. In sostanza l'algoritmo costruito permetterà quindi al *robot* di inseguire una data traiettoria di riferimento evitando al tempo stesso collisioni con ostacoli noti o con altri agenti che potrebbero presentarsi lungo di essa.

Consideriamo ora un *robot* soggetto alle dinamiche già introdotte in precedenza nel Capitolo 1:

$$\begin{cases} \dot{x} = & v\cos(\theta) \\ \dot{y} = & v\sin(\theta) \\ \dot{\theta} = & \omega \end{cases} \tag{6.2}$$

Data una traiettoria di riferimento (x_d, y_d), definiamo gli errori di posizione come:

$$\begin{cases} e_x = & x - x_d \\ e_y = & y - y_d \end{cases}$$

e una funzione distanza:

$$d_a = \sqrt{\left(\frac{x - x_c}{\alpha}\right)^2 + \left(\frac{y - y_c}{\beta}\right)^2}$$

dove (x_c, y_c) descrivono le coordinate del centro dell'ostacolo, che come detto ha posizione nota, mentre α e β rappresentano due parametri che avranno come unico vincolo quello di essere strettamente positivi.

In accordo con [20] si risolve il problema di *obstacle avoidance* grazie alla seguente funzione, definita per $\alpha = \beta = 1$:

$$V_a = \left(\min\left\{0, \frac{d_a^2 - R^2}{d_a^2 - r^2}\right\}\right)^2 \tag{6.3}$$

in cui $r > 0$ e $R > 0$, con $R > r$, sono rispettivamente il raggio dell'ostacolo effettivo e quello della regione di attivazione per il problema di *avoidance*. Tali due regioni, appena enunciate sono definite nel seguente modo:

$$\Omega = \left\{ \begin{bmatrix} x & y \end{bmatrix} : (x,y) \in \mathbb{R}^2, \| \begin{bmatrix} x & y \end{bmatrix}^T - \begin{bmatrix} x_a & y_a \end{bmatrix}^T \| \leq r \right\}$$

$$\Gamma = \left\{ \begin{bmatrix} x & y \end{bmatrix} : \begin{bmatrix} x & y \end{bmatrix} \notin \Omega, r < \| \begin{bmatrix} x & y \end{bmatrix}^T - \begin{bmatrix} x_a & y_a \end{bmatrix}^T \| \leq R \right\}$$

e sono rappresentate in Figura 6.3.

La funzione (6.3), per come è definita, presenta un valore pari a zero quando si è al di fuori della regione di attivazione Γ mentre assume un valore finito, diverso da zero, quando si è al suo interno. Considerando regioni di tipo circolare, come mostrato in Figura 6.3, è possibile effettuare la seguente derivazione parziale della funzione V_a rispetto alle sue due coordinate del piano x e y ottenendo così:

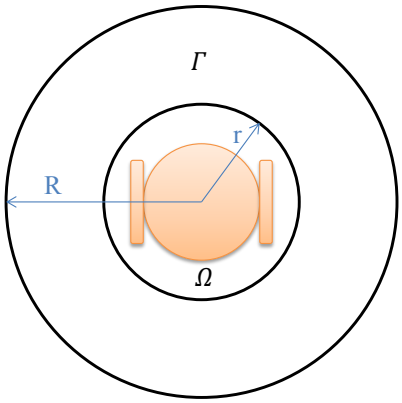


Figura 6.3: Raggio dell'ostacolo effettivo da evitare (r); raggio della regione di attivazione del problema di *avoidance* (R)

$$\frac{\partial V_a}{\partial x} = \begin{cases} 0, & d_a \geq R \\ 4\frac{(R^2-r^2)(d_a^2-R^2)}{(d_a^2-r^2)^3}(x-x_c), & R > d_a > r \\ 0, & d_a < r \end{cases} \tag{6.4}$$

e

$$\frac{\partial V_a}{\partial y} = \begin{cases} 0, & d_a \geq R \\ 4\frac{(R^2-r^2)(d_a^2-R^2)}{(d_a^2-r^2)^3}(y-y_c), & R > d_a > r \\ 0, & d_a < r \end{cases} \tag{6.5}$$

Definiamo ora i due potenziali artificiali dell'algoritmo relativi alle due coordinate che verranno aggiornati a ogni ciclo iterativo:

$$\begin{cases} E_x = & e_x + \frac{\partial V_a}{\partial x} \\ E_y = & e_y + \frac{\partial V_a}{\partial y} \end{cases} \tag{6.6}$$

Questi, come visto al paragrafo 6.1, consentono di generare sulla superficie di lavoro il campo vettoriale che guiderà i *robot* al raggiungimento dei propri obiettivi; a tale proposito viene definito, per $(E_x, E_y) \neq 0$, l'orientamento desiderato dell'agente come:

$$\theta_d = atan2(-E_y, -E_x) \tag{6.7}$$

trovando così l'errore d'orientamento dato da:

$$e_\theta = \theta - \theta_d$$

Si ribadisce che le due componenti definite dalle formule (6.4) e (6.5) sono nulle al di fuori della regione di attivazione Γ, non andando così a influenzare né i potenziali descritti da (6.6) né l'angolo θ_d ottenuto dalla (6.7). È importante inoltre osservare che θ_d definisce l'orientamento desiderato con cui il *robot* andrà a spostarsi nel piano di lavoro, in accordo con quanto detto sempre nel paragrafo 6.1; infatti, come si vede dalla (6.7), θ_d dipende dalle due funzioni potenziali ricavate rispetto a x e y e quindi dipenderà anche dalla posizione dell'ostacolo e dalla posizione attuale dell'agente.

Nell'implementazione di questo algoritmo si assumerà sempre che all'interno della regione di attivazione Γ la traiettoria di riferimento resti costante, quindi $\dot{x}_d = \dot{y}_d = 0$, per $r \leq d_a \leq R$. Questo significa che quando l'agente individua un ostacolo il suo obiettivo di riferimento viene momentaneamente congelato e rimane tale fino a quando il *robot* non ha risolto il problema di *avoidance*, che momentaneamente assume quindi la massima priorità. Successivamente, al di fuori della regione di attivazione, si avrà l'aggiornamento del riferimento e il *robot* tornerà a risolvere il suo problema di *tracking*. La ragione di questa scelta è dettata dal fatto che in presenza di un ostacolo il problema di *avoidance* assume, come detto, una priorità maggiore rispetto al problema di *tracking,* in quanto la collisione dell'agente con l'ostacolo andrebbe a compromettere l'intero algoritmo. Per maggiori dettagli riguardanti tale procedimento si rimanda a [20].

6.2.1 Obstacle Avoidance

Studiamo ora il caso in cui un ostacolo statico circolare di raggio r viene posto nel piano di lavoro in una posizione nota. Si vuole progettare una legge di controllo, basata sul metodo dei potenziali artificiali, che riesca a gestire e risolvere sia il problema di *tracking* per una data traiettoria di riferimento sia, allo stesso tempo, quello di *obstacle avoidance*. Per fare ciò viene in nostro aiuto il seguente teorema: considerato il sistema descritto in (6.2), una traiettoria di riferimento che porta a un dato *goal* definito dalle coordinate (x_d, y_d) e un ostacolo statico localizzato sulla superficie operativa tramite coordinate (x_c, y_c) è possibile definire un orientamento desiderato θ_d, come in (6.7), che permette di garantire una soluzione sia al problema di *tracking,* con banda d'errore limitata, sia al problema di *obstacole avoidance* nel caso in cui venga applicata la seguente legge di controllo algebrica:

$$\begin{cases} \omega = & -K_\theta e_\theta + \hat{\dot{\theta}}_d \\ v = & -K\cos(e_\theta)D \end{cases} \tag{6.8}$$

per tutti i guadagni $K < 0$, $K_\theta > 0$ e per $D = \sqrt{E_x^2 + E_y^2} > 0$. Naturalmente il teorema può essere tranquillamente esteso al caso di ostacoli multipli definendo per ognuno di essi una specifica funzione di *obstacle avoidance*; per la dimostrazione di tale teorema si rimanda a [20].

Riportiamo ora i risultati sperimentali ottenuti compiendo un *test* in cui un agente *e-puck* partendo da coordinate:

$$\begin{cases} x = & 15 \\ y = & 35 \end{cases}$$

deve raggiungere il suo *goal* di coordinate:

$$\begin{cases} x_d = & 100 \\ y_d = & 35 \end{cases}$$

evitando un ostacolo, definito come (x_c, y_c, r), con parametri $(50, 40, 6)$ e regione di attivazione Γ con raggio $R = 10$; inoltre si utilizzano guadagni così definiti:

$$\begin{cases} K = & -0.04 \\ K_\theta = & 0.001 \end{cases} \tag{6.9}$$

ricavati tramite metodo empirico.

I risultati ottenuti sono riportati in Figura 6.4 dove è mostrato dal grafico in

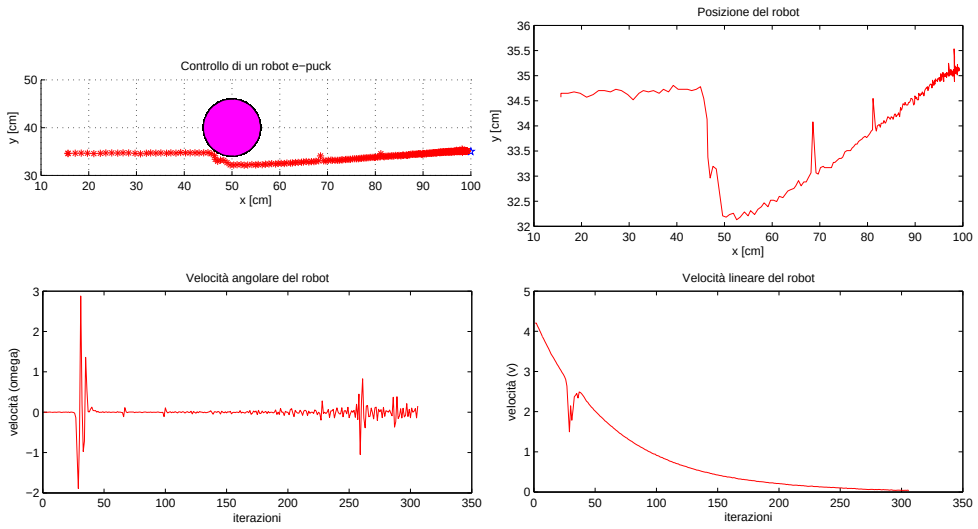


Figura 6.4: *Obstacle avoidance* con potenziali artificiali

alto a sinistra come l'agente riesca a risolvere sia il problema di *tracking* che quello di *obstacle avoidance* ottenendo dei buoni risultati; il grafico in alto a destra riporta la medesima informazione con l'unica differenza che in esso non viene rappresentata la superficie dell'ostacolo ma ci si concentra solo sulla traiettoria mantenuta dal *robot* reale nel piano di lavoro. Nei due grafici in basso di Figura 6.4, è possibile osservare gli andamenti della velocità, angolare e lineare; è importante notare che, in relazione ai grafici mostrati, il *robot* impiega un tempo di circa 44 secondi per giungere al proprio *goal*. Tale tempo risulta essere elevato se generalmente confrontato con quello ottenuto per risolvere i problemi di *obstacle avoidance* tramite algoritmo di controllo predittivo *DPC* analizzato nel paragrafo 5.2.

Per quanto riguarda la velocità angolare si nota, come ci si aspettava, che essa rimane sempre prossima al valore nullo, in quando l'obiettivo dell'agente è sostanzialmente posto su una linea retta rispetto alla sua posizione di partenza. Fanno eccezione a ciò la regione di attivazione del problema di *obstacle avoidance,* in cui il *robot* dovrà necessariamente ruotare per evitare l'ostacolo, e quella in prossimità dell'obiettivo finale in cui, a causa delle basse velocità dovute a un basso valore del potenziale attrattivo, l'agente dovrà applicare frequentemente una correzione al suo orientamento per raggiungere il proprio *goal*. Dal grafico della velocità lineare si osserva infatti che il suo andamento subisce, come ci si aspettava, un continuo decremento fino al raggiungimento del *goal* dove essa risulta essere molto bassa, quasi nulla. Questo spiega il perché di un tempo di raggiungimento dell'obiettivo finale così elevato, che è appunto dato dal fatto che la velocità continua a decrescere, in modo esponenziale, man mano che ci avviciniamo a esso. In particolare, come già spiegato, se il *robot* è lontano dal *goal*, la forza attrattiva sarà elevata producendo un'elevata velocità di spostamento, mentre man mano che il *robot* si avvicina al suo obiettivo la forza attrattiva diminuisce, generando così delle velocità di spostamento sempre più limitate in modulo.

Un ruolo fondamentale è assunto poi dai valori dati ai due guadagni, che ricordiamo dovranno essere scelti in modo che $K < 0$ e $K_\theta > 0$ per non avere instabilità

nel sistema. Nello specifico, il valore del guadagno K_θ non dovrà assumere valori troppo elevati che potrebbero causare pericolosi incrementi della velocità angolare con conseguente introduzione di instabilità nel sistema. A tale proposito, variando il valore di tale guadagno, entro determinati limiti, atti a non portare il sistema all'instabilità, è possibile controllare la sovraelongazione nella traiettoria dell'agente verso l'obiettivo andando così a rendere il sistema più performante in termini di velocità nel raggiungere il *goal.* Dalla Figura 6.5 che presenta valori dei guadagni pari a:

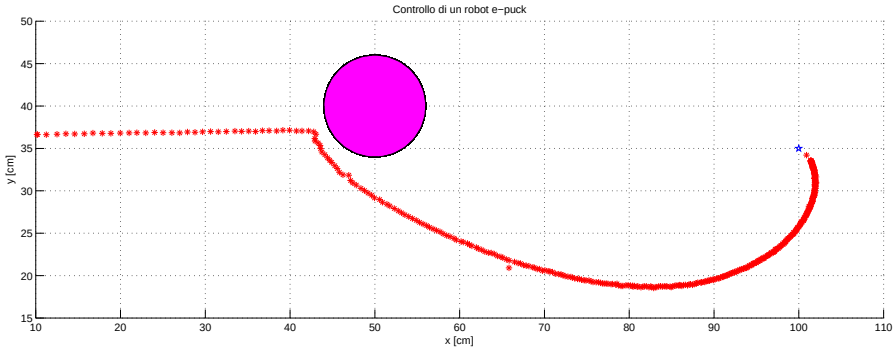


Figura 6.5: Errata taratura del guadagno K_θ con conseguente sovraelongazione nella traiettoria del *robot*

$$\begin{cases} K = & -0.04 \\ K_\theta = & 0.01 \end{cases}$$

è infatti ben visibile che avendo aumentato il valore di K_θ di un fattore dieci rispetto a quello utilizzato in Figura 6.4, si ottiene una sovraelongazione dell'agente nel raggiungere il proprio *goal*, il che si traduce in una perdita prestazionale. Alzando invece il valore di K si andrà ad aumentare la velocità dell'agente durante la risoluzione dei suoi *task*; anche tale guadagno non dovrà crescere troppo se non si vogliono avere problemi nel controllare il sistema.

6.2.2 Collision Avoidance

Si consideri ora lo scenario in cui due agenti devono risolvere in modo indipendente un problema di *tracking* verso il loro obiettivo noto a priori e allo stesso tempo devono riuscire a gestire in *real time* il problema di *collision avoidance* in modo che essi non collidano tra loro nel raggiungere i rispettivi *goal.* Come al solito la posizione (x, y) e l'orientamento θ degli agenti sono ricavati tramite *webcam;* ogni *robot* quindi vede l'altro agente, operante nello stesso piano di lavoro, come un ostacolo mobile che presenta coordinate note e che sarà circoscritto in una circonferenza di raggio r. Tale agente sarà inoltre caratterizzato da una regione di attivazione Γ per il problema di *avoidance* che avrà raggio R, dove anche in questo caso si avrà che $R > r$. Come per il caso precedente, riferito al problema di *obstacle avoidance,* è possibile ricavare, tramite uno specifico risultato, una legge di controllo che permetta a ogni *robot* di risolvere il problema di *tracking* e allo stesso tempo quello relativo

al *collision avoidance* tra i due agenti. Il teorema considerato non è altro che un'estensione di quello citato al paragrafo precedente per il caso si utilizzino più *robot* e afferma che, considerando $i = 1, \dots M$ sistemi di tipo (6.2) con le rispettive traiettorie di riferimento e con i corrispondenti *goal* definiti dalle coordinate (x_{di}, y_{di}), è possibile definire un dato orientamento θ_{di} come da (6.7) in modo da garantire che gli agenti inseguano la data traiettoria di riferimento, con una banda d'errore limitata, al di fuori della regione di attivazione, mentre al suo interno essi risolvano in modo dinamico il problema di *collision avoidance.* Questo è possibile applicando il seguente controllo per ogni *robot*:

$$\begin{cases} \omega = & -K_{\theta i} e_{\theta i} + \hat{\dot{\theta}}_{di} \\ v = & -K_i \cos(e_{\theta i}) D_i \end{cases} \tag{6.10}$$

per tutti i guadagni $K_i < 0$, $K_{\theta i} > 0$ e per $D_i = \sqrt{E_{xi}^2 + E_{yi}^2} > 0$. Naturalmente il risultato può essere esteso al caso di più di due agenti operanti nella medesima superficie di lavoro; nel nostro caso dovendo controllare due *robot* avremo che $i = 1, 2$. Per la dimostrazione di tale teorema si rimanda a [20].

Riportiamo ora i risultati sperimentali trovati compiendo un *test* in cui due agenti *e-puck* partendo da coordinate:

$$\begin{cases} x_1 = & 10 \\ y_1 = & 40 \end{cases} \qquad \begin{cases} x_2 = & 10 \\ y_2 = & 50 \end{cases}$$

devono raggiungere i loro corrispondenti *goal* di coordinate:

$$\begin{cases} x_{1d} = & 100 \\ y_{1d} = & 35 \end{cases} \qquad \begin{cases} x_{2d} = & 100 \\ y_{2d} = & 15 \end{cases}$$

risolvendo al tempo stesso il problema di *collision avoidance* e quindi evitando di collidere tra loro. L'ostacolo dinamico assumerà quindi, in base all'agente considerato, le coordinate della posizione dell'altro *robot* che, come detto, saranno note in quanto ricavate tramite *webcam.* L'ostacolo viene quindi definito come (x_i, y_i, r), dove viene scelto un valore del raggio $r = 3$ tenendo conto di un certo margine di sicurezza in eccesso dato dal fatto che $R_{robot} = 2.1$. Inoltre si sceglie un raggio di attivazione per il problema di *collision avoidance* pari a $R = 6$ e vengono utilizzati, per entrambi i *robot,* gli stessi valori dei guadagni definiti anche per il caso precedente di *obstacle avoidance* (6.9).

I risultati ottenuti sono quelli riportati in Figura 6.6 dove è mostrato dai due grafici in alto come gli agente riescano a risolvere sia il problema di *tracking* sia, allo stesso tempo, quello di *collision avoidance* ottenendo dei buoni risultati. Da notare che, anche in questo caso, come per la Figura 6.5 riferita al problema di *obstacle avoidance,* i campioni di posizione, rappresentati tramite asterischi, risultano essere leggermente distanti tra loro alla partenza, mentre man mano ci si avvicina al *goal* dei due agenti essi si infittiscono sempre di più. Questo fenomeno è ben spiegato dal grafico, in basso a destra di Figura 6.6, nel quale vengono rappresentate le velocità lineari dei due *robot* che inizialmente risultano essere elevate, ma tendono poi a decrescere esponenzialmente verso valori prossimi a zero più gli agenti si avvicinano ai rispettivi *goal.* È importante osservare che, in relazione ai grafici mostrati, i *robot* impiegano lo stesso tempo, circa 48 secondi, per giungere ai propri *goal*; data infatti

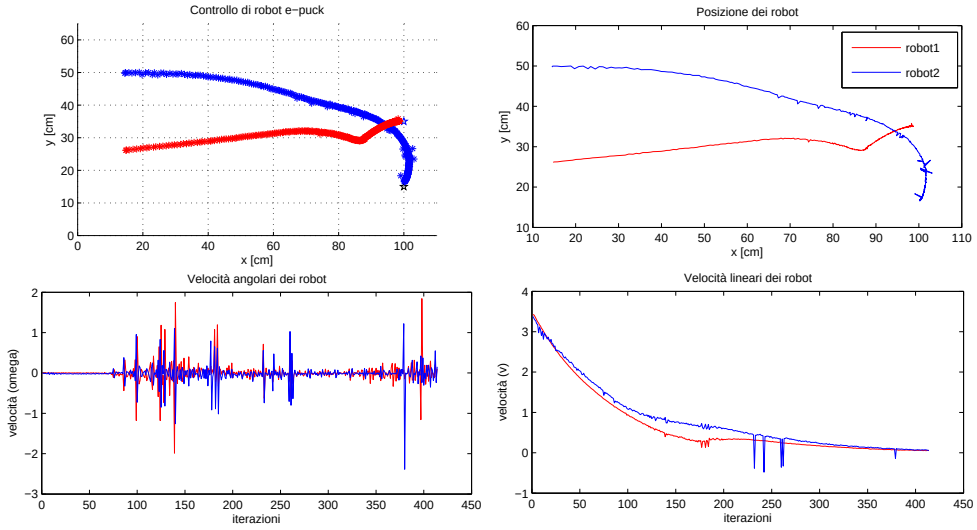


Figura 6.6: *Collision avoidance* con potenziali artificiali

la legge di controllo algebrica (6.10) questo algoritmo garantisce che, qualunque sia la distanza dei *robot* dai propri obiettivi, essi arrivino ai *goal* circa nel medesimo istante e questo è possibile grazie a una corretta modulazione delle loro velocità lineari. Anche in questo caso il tempo di raggiungimento dell'obiettivo risulta essere più elevato se comparato con quello generalmente ottenuto per risolvere i problemi di *collision avoidance* tramite algoritmo di controllo predittivo *DPC* analizzato nel paragrafo 5.3.

Per quanto riguarda le velocità angolari si nota dal grafico in basso a sinistra di Figura 6.6 che, come ci si aspettava, questa volta esse variano in quanto i rispettivi *goal* dei due *robot* non sono posti sulla medesima linea orizzontale rispetto alla loro posizione iniziale; in aggiunta nella regione di attivazione del problema di *collision avoidance* e in prossimità del raggiungimento dell'obiettivo esse varieranno ulteriormente, per le stesse motivazioni spiegate nel paragrafo precedente. Il problema di collisione tra i due agenti verrà quindi risolto tramite una corretta modulazione delle velocità, angolari e lineari, dei due *robot*.

Un ruolo fondamentale assumono poi i valori dati ai guadagni che ricordiamo dovranno essere scelti in modo che $K_i < 0$ e $K_{\theta i} > 0$ per non avere instabilità nel sistema. Inoltre il valore del guadagno $K_{\theta i}$ non dovrà assumere valori troppo elevati che andrebbero a causare pericolosi incrementi della velocità angolare, che a loro volta potrebbero introdurre elementi di instabilità nel sistema. Inoltre, variando il valore di tale guadagno entro determinati limiti, che nel caso di più agenti nello stesso piano lavorativo risultano essere ancora più stringenti per non portare il sistema all'instabilità, è possibile controllare le sovraelongazioni delle traiettorie dei due *robot* nel raggiungimento dei loro obiettivi, andando così a rendere il sistema più performante in termini di velocità. Aumentando invece il valore di K_i si andrà, anche questa volta, ad aumentare la velocità dell'agente durante la risoluzione dei suoi *task*, quindi anche tale guadagno non dovrà crescere eccessivamente se non si vogliono avere problemi di controllo del sistema in esame.

6.3 Confronto sperimentale

In questo paragrafo si vuole effettuare un confronto tra i vari algoritmi descritti nella tesi e utilizzati per risolvere i problemi di *avoidance*. In particolare, tramite prove sperimentali, si analizzano le differenze e le analogie ottenute implementando su *robot e-puck* l'algoritmo che utilizza tecniche di controllo predittive con relative funzioni di risoluzione per i problemi di *avoidance*, descritte nei precedenti capitoli, con quello appena illustrato nel paragrafo antecedente che adotta il metodo dei potenziali artificiali.

Si effettua quindi un *test* andando a implementare sugli agenti *e-puck* l'algoritmo che utilizza una tecnica di controllo predittivo *DPC*, che presenta la struttura innovativa con logica di controllo in parallelo, quindi con anello di *Feedback Linearization* posto internamente all'agente, come trattato al paragrafo 3.4. Successivamente si ripete l'esperimento, mantenendo le stesse caratteristiche ambientali esterne, ma applicando ora ai *robot* l'algoritmo basato sulla tecnica dei potenziali artificiali per risolvere simultaneamente problemi di *tracking* e di *avoidance*. Per far ciò non si farà altro che andare a unire i due procedimenti illustrati nei paragrafi 6.2.1 e 6.2.2 in un unico algoritmo che permetta la gestione sia di problemi di *obstacole* che di *collision avoidance* oltre che a garantire al *robot* il raggiungimento del proprio *goal*; in questo modo si riescono a evitare inutili ripetizioni concettuali nelle quali sicuramente si incorrerebbe confrontando singolarmente sul sistema reale i vari algoritmi.

In sintesi, con entrambe le due metodologie di controllo presentate i *robot* dovranno riuscire a risolvere sia il problema di *tracking* che, al tempo stesso, quello di *obstacle* e *collision avoidance*. Durante i *test* vengono utilizzati e mantenuti costanti i seguenti parametri:

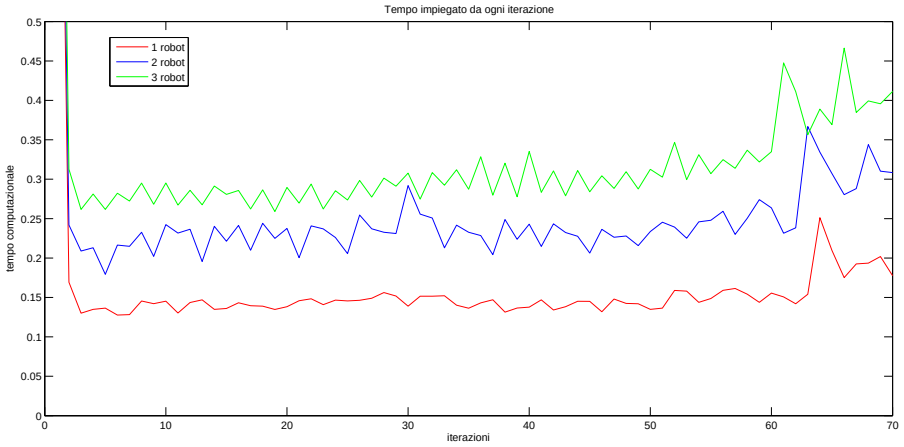
- tecnica di controllo MPC: tempo di campionamento $\tau = 0.4$ e distanza minima desiderata dal centro dell'ostacolo $d = R_h^o + R_{robot} + \delta_i$, dove R_h^o rappresenta il raggio dell'ostacolo e δ_i una distanza limite opportunamente calcolata, in accordo con quanto trattato nel Capitolo 5;

- tecnica potenziali artificiali: tempo di campionamento $\tau = 0.3$ che è circa lo stesso del problema di controllo predittivo, questa costatazione rende infatti ancor più interessante l'effettivo confronto tra i due algoritmi implementati. Inoltre viene scelto un raggio della regione di attivazione per il problema di *collision avoidance* pari a $R_1 = 8$, sapendo che il raggio del *robot* è pari a $R_{robot} = 2.1$, mentre viene presa una regione di attivazione per il problema di *obstacle avoidance* con un raggio pari a $R_2 = 16$, sapendo che il raggio dell'ostacolo è pari a $r = 10$.

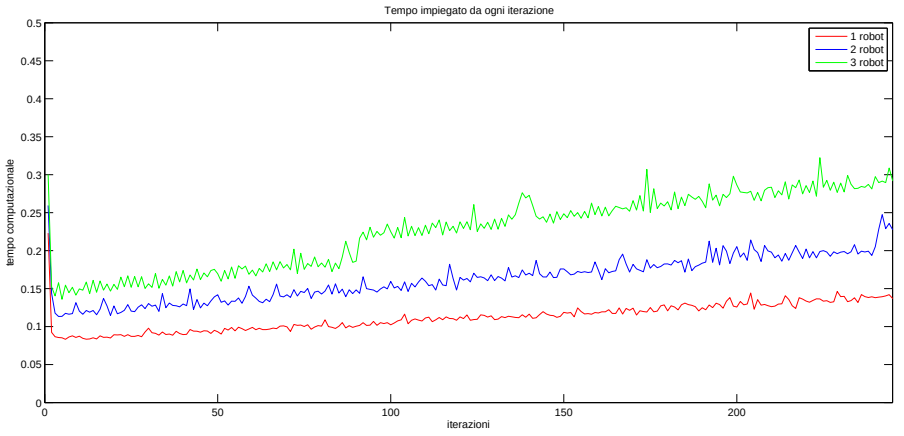
Come anticipato in precedenza, l'algoritmo che opera con i potenziali artificiali risulta essere particolarmente efficiente quando i parametri della superficie di lavoro, ostacoli e *goal* dei *robot,* vengono definiti e impostati *off-line;* dopodiché in base alla posizione dell'agente nel piano lavorativo viene calcolato *on-line* il potenziale in quel punto specifico della superficie e tramite la semplice legge algebrica (6.10) si va a guidare il *robot* verso il suo *goal* risolvendo in contemporanea eventuali problemi di *avoidance* che si potrebbero presentare lungo il suo cammino. Questo algoritmo non effettua quindi nessuna ottimizzazione e analizzando semplicemente i parametri utilizzati, appena descritti, si può osservare che esso adotta un tempo di campionamento pari a $\tau = 0.3$.

Tramite la tecnica predittiva si esegue invece, oltre che la linearizzazione internamente all'HW-SW del *robot,* anche l'ottimizzazione della traiettoria dell'agente in modo da permettere a esso di evitare eventuali altri *robot* o ostacoli noti presenti nel piano di lavoro nel miglior modo possibile, raggiungendo così il proprio *goal* in modo performante. Tutto questo viene fatto utilizzando un tempo di campionamento pari a $\tau = 0.4$, che risulta essere leggermente maggiore, ma sicuramente comparabile con quello utilizzato per la tecnica di controllo basata sui potenziali artificiali. Nelle prove mostrate in questo capitolo si dimostrerà che la tecnica predittiva può risultare più prestazionale in termini di velocità dei *robot* nel raggiungere i rispettivi riferimenti finali, oltre che più adatta a risolvere problemi dinamici che comportano variazioni dell'ambiente operativo in cui agiscono gli agenti.

Analizzando ora i diversi tempi computazionali utilizzati dai due algoritmi risulta molto utile la Figura 6.7, dove viene mostrata la relazione che esiste tra il numero di *robot* operanti nel piano di lavoro e il tempo impiegato da ogni singolo ciclo iterativo dell'algoritmo per consentire agli agenti di portare a termine i propri *task*. In



(a)



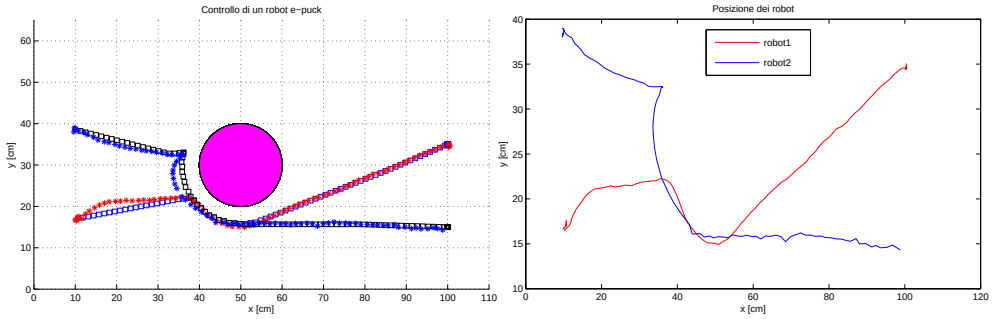
(b)

Figura 6.7: Tempo impiegato da ogni iterazione per svolgere un singolo ciclo operativo dell'algoritmo che utilizza tecnica predittiva (a) e tecnica basata sui potenziali artificiali (b) in funzione del numero di agenti operanti nel piano di lavoro

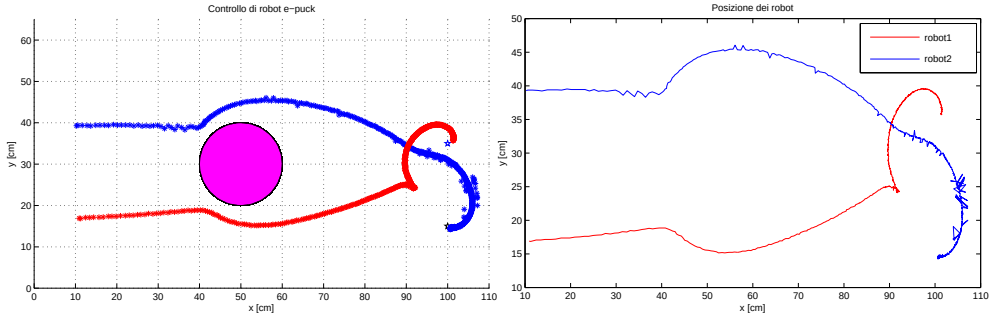
entrambi i casi, all'aumentare del numero di *robot,* il tempo computazionale totale richiesto per lo svolgimento di tutte le operazioni, aumenta in modo sostanzialmente analogo. È inoltre importante notare che l'algoritmo *MPC*, nonostante svolga come detto un'operazione di ottimizzazione, non presenta un carico computazionale significativamente maggiore rispetto al metodo dei potenziali artificiali. Quello che invece varia in modo rilevante è il numero di passi che i due algoritmi devono compiere per portare gli agenti ai propri obiettivi. Si vede infatti che l'algoritmo che si basa sul metodo dei potenziali artificiali, Figura 6.7 (b), richiede un numero di passi molto maggiore rispetto a quello che utilizza la tecnica di controllo predittiva, Figura 6.7 (a). Si noti che tali grafici sono stati ricavati risolvendo un semplice problema di *tracking* per ogni agente utilizzato senza inserire nessun problema di *avoidance.* A tale proposito si intuisce che nel caso di inserimento di eventuali problemi di *collision* o *obstacle avoidance,* per entrambi gli algoritmi, si avrà un incremento di iterazioni nella risoluzione dello specifico problema, dato che vengono introdotti degli elementi che tendono a complicare il raggiungimento degli obiettivi finali da parte dei vari agenti. Nel dettaglio, come si mostrerà nel seguito, introducendo anche problemi di *avoidance* per la tecnica di controllo predittivo il numero di passi richiesto per raggiungere gli obiettivi aumenterà di un ordine molto minore rispetto al metodo basato sui potenziali artificiali, dove invece le iterazioni andranno ad aumentare quasi esponenzialmente. Inoltre avendo che ogni iterazione impiega circa, come visto, 0.3 secondi questo andrà a incrementare il tempo di navigazione con cui gli agenti arriveranno ai rispettivi *goal* e ciò motiverà appunto la forte discrepanza che si ha nei due algoritmi in termini di tempo per raggiungere l'obiettivo. In aggiunta, sempre rispetto ai tempi computazionali richiesti per svolgere le varie operazioni, che influenzano direttamente sul tempo di campionamento τ, sono state fatte alcune osservazioni sperimentali analoghe per ambedue gli algoritmi confrontati. L'inserimento di ostacoli nello spazio di lavoro porta a un aumento del tempo di risoluzione della singola iterazione in quanto incrementa la complessità dell'algoritmo stesso. Dopo aver ripetuto numerose prove sperimentali, si ricava che la presenza di cinque ostacoli provoca un aumento di circa 0.02 al tempo di computazione di quasi ogni iterazione del ciclo operativo. Tuttavia tale aumento risulta essere inferiore di circa un fattore 10 rispetto all'incremento prodotto dall'aggiunta di un ulteriore *robot* nella superficie operativa. A fronte di questa considerazione si è potuto affermare che l'elemento critico risulta essere quindi il numero di agenti utilizzati rispetto al numero di ostacoli inseriti e per tale motivo si è deciso di concentrare proprio su questo fattore la nostra analisi. Realizzando ora un confronto tra i grafici riportati in Figura 6.7 si nota che per ogni *robot* aggiunto nel piano di lavoro si ha un aumento del tempo di computazione delle singole iterazioni del ciclo operativo. Questo, considerato che le due tecniche, in un contesto reale, prevedono un'implementazione di tipo distribuito, consente di verificare la scalabilità degli algoritmi studiati.

Un ulteriore esperimento svolto consiste nell'introdurre due agenti *e-puck* nel piano di lavoro a disposizione, di dimensioni 115 x $66\,cm^2$, in cui è inoltre presente un ostacolo, solo virtualmente implementato, posto in una posizione nota a priori e definito come segue: $(50, 30, 10)$ dove, come al solito, i primi due termini dentro le parentesi indicano le coordinate (x, y) del centro dell'ostacolo, che ricordiamo essere circolare, mentre il terzo dato rappresenta il suo raggio. I *robot*, in entrambi i casi, partiranno dalle stesse posizioni iniziali che presentano coordinate pari a $(10, 17)$ per il primo agente e $(10, 39)$ per il secondo e dovranno raggiungere i medesimi

goal alle coordinate $(100, 35)$ per il primo *robot* e $(100, 15)$ per il secondo. Ponendo posizioni iniziali e obiettivi degli agenti in questo modo ci si aspetta che i *robot* siano costretti a risolvere anche un problema di *collision avoidance* in quanto le loro traiettorie dovranno incrociarsi per permettere il raggiungimento dei propri *goal*. Utilizzando ora le seguenti figure andiamo a effettuare il confronto tra i due algoritmi implementati, avvalorando attraverso un'analisi numerica quanto appena affermato; in primo luogo si analizza la traiettoria degli agenti in ambedue i casi, concentrandoci appunto sulla posizione dei due *robot*. Dalla Figura 6.8 si osserva facilmente come



(a)



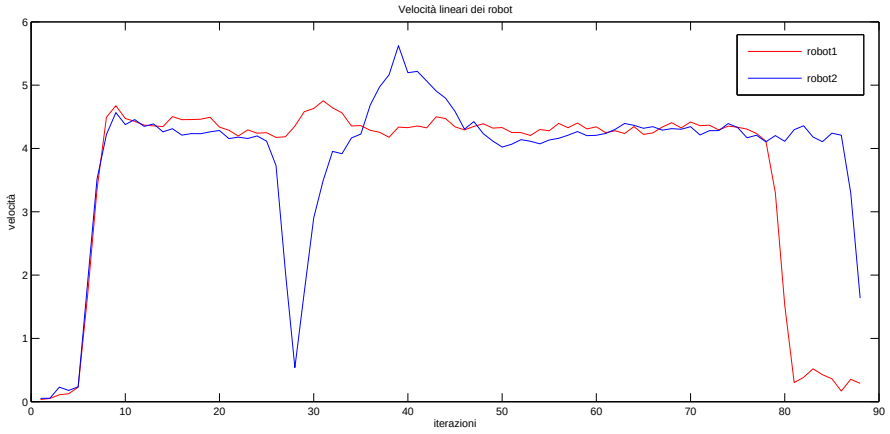
(b)

Figura 6.8: Traiettoria mantenuta dai *robot* che utilizzano l'algoritmo con tecnica di controllo predittivo (a) e l'algoritmo basato sul metodo dei potenziali artificiali (b)

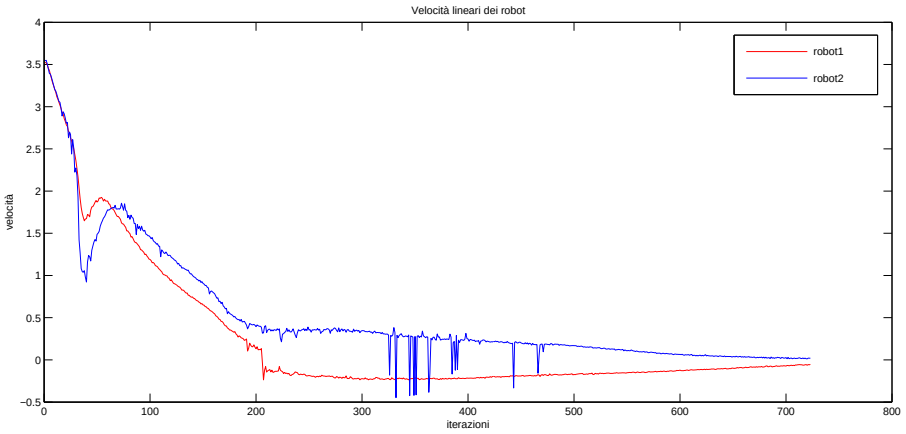
tutti e due gli algoritmi implementati portino alla corretta risoluzione del problema di *obstacle* e *collision avoidance* conducendo infatti le traiettorie degli agenti agli obiettivi desiderati. Nel caso in esame si osserva inoltre come la traiettoria generata dalla tecnica predittiva, Figura 48 (a), sia più breve grazie al fatto che l'ostacolo non ha un effetto repulsivo sulla traiettoria, ma costituisce semplicemente un vincolo fisso inviolabile.

Un'ulteriore difformità tra i due algoritmi è data, come anticipato in precedenza, dal tempo di navigazione impiegato dagli agenti per raggiungere il proprio obiettivo. Infatti, nell'algoritmo che utilizza la tecnica di controllo predittivo *DPC*, Figura 6.8 (a), gli agenti impiegano circa 11 secondi per percorrere la traiettoria generata

dal punto iniziale fino al loro *goal,* mentre nell'algoritmo che utilizza la tecnica dei potenziali artificiali *APF*, Figura 6.8 (b), i *robot* impiegano circa 96 secondi per raggiungere i loro riferimenti finali. Questa differenza è giustificata dal fatto che, per quanto riguarda l'algoritmo che lavora con i potenziali artificiali, man mano i *robot* si avvicinano ai loro obiettivi più il potenziale attrattivo del corrispondente *goal* decresce, provocando una proporzionale diminuzione della forza di attrazione e della velocità di navigazione dell'agente stesso. Il *robot* quindi si avvicinerà al *goal* sempre con una velocità minore avendo così dei campioni, che indicano la posizione dell'agente, sempre più vicini l'uno dall'altro in uno spazio sempre più limitato; ciò non avviene invece nel caso di tecnica di controllo predittivo dove i campioni rimangono a una distanza circa costante lungo tutta la traiettoria. Quest'ultima osservazione, oltre che mostrata nella Figura 6.8, è inoltre verificata dagli andamenti delle velocità lineari tenute dagli agenti nei due algoritmi e mostrate nella Figura 6.9. Si ribadisce che il tempo di campionamento τ utilizzato per entrambi gli algoritmi è



(a)



(b)

Figura 6.9: Andamento delle velocità lineari dei *robot* che utilizzano l'algoritmo con tecnica di controllo predittivo (a) e l'algoritmo basato sul metodo dei potenziali artificiali (b)

circa lo stesso ed è inoltre importante sottolineare che il ciclo operativo di controllo

termina, per entrambi i metodi, quando ambedue i *robot* raggiungono i propri *goal*. Dalla Figura 6.9 si nota, ancora una volta, che il numero di passi fatti per giungere all'obiettivo da parte dei due algoritmi risulta essere molto diverso tra loro. In Figura 6.9 (b), riferita all'algoritmo che utilizza i potenziali artificiali, si osserva infatti che il *robot* raggiunge il proprio obiettivo dopo circa 720 iterazioni, mentre nel grafico 6.9 (a), riferito all'algoritmo che utilizza controllo predittivo, le iterazioni per giungere al *goal* sono circa 90. Questo prova quanto detto in precedenza, ovvero che inserendo problemi di *avoidance,* oltre al classico problema di *tracking,* i passi impiegati dai due algoritmi per giungere all'obiettivo crescono in maniera molto differente. Si ha infatti, come anticipato, un leggero aumento nell'algoritmo che utilizza la tecnica di controllo predittivo passando dalle 70 iterazioni della Figura 6.7 (a), alle 90 iterazioni della Figura 6.9 (a), per permettere la corretta risoluzione del problema in esame. Per quanto riguarda la tecnica di controllo basata sui potenziali artificiali, si passa invece dalle 250 iterazioni della Figura 6.7 (b), alle 720 iterazioni della Figura 6.9 (b), avendo di fatto un incremento significativo che provoca, come detto, un conseguente aumento del tempo di navigazione degli agenti per raggiungere i propri *goal*.

Il fatto di avere un differente numero di iterazioni tra i due algoritmi, pur utilizzando circa lo stesso tempo di campionamento, la stessa posizione iniziale e i *goal* posti alle medesime coordinate, è dovuto come visto, alle differenti velocità che essi andranno a impartire alle ruote dei *robot*. Mentre, come visto in Figura 6.9 (a), e già osservato in precedenza, le velocità lineari dei *robot* nel caso del metodo dei potenziali artificiali diminuiscono all'approssimarsi degli agente ai *goal,* nel caso dell'algoritmo di controllo predittivo gli agenti inizialmente accelerano fino al raggiungimento di una velocità di crociera costante, che verrà poi mantenuta tale fino all'approssimarsi ai loro obiettivi finali, questo a meno di dover risolvere problemi di *avoidance,* che verranno appunto gestiti al meglio proprio effettuando una corretta modulazione di tali velocità lineari. Ciò dimostra quindi l'osservazione fatta inizialmente, ovvero che la tecnica di controllo predittivo risulta avere delle prestazioni migliori in termini di velocità nel raggiungere i propri obiettivi e in termini di robustezza del sistema ad adattarsi alle possibili dinamiche che si potrebbero presentare nello spazio operativo.

6.4 Caratteristiche generali dell'algoritmo studiato

In questo capitolo è stata quindi illustrata una possibile tecnica di controllo per la navigazione di agenti mobili basata sull'utilizzo di potenziali artificiali; essa risulta essere molto diffusa in letteratura e costituisce un efficace metodo di controllo per risolvere sia il problema di *tracking* sia, al tempo stesso, eventuali problemi di *avoidance* incontrati lungo la traiettoria che porta l'agente all'obiettivo finale. Si è infatti dimostrato, attraverso prove sperimentali, che questa tecnica, creando un campo di forze vettoriale opportunamente pesato all'interno della nostra superficie di lavoro, permette di guidare i *robot* verso i loro obiettivi risolvendo durante la navigazione sia eventuali problemi di *obstacle* che di *collision avoidance*. Si è poi effettuato un confronto tra tale algoritmo e quello descritto nei precedenti capitoli che sfrutta tecniche predittive per il controllo di agenti mobili. Dalle prove sperimentali è emerso che grazie alla teoria predittiva e alla struttura innovativa con logica in parallelo implementata e presentata nella trattazione al Capitolo 3, è possibile ottenere risultati nettamente più performanti se confrontati con la tecnica sviluppata in questo capitolo. Si può quindi affermare, dopo le analisi effettuate, che sia dal punto di vista

delle *performance* sia dal punto di vista della robustezza del sistema a disturbi o a variazioni dinamiche dell'ambiente operativo, il controllo predittivo appositamente progettato risulta essere nettamente migliore e più efficiente per il controllo dei *robot e-puck* rispetto alla sola legge di controllo algebrica fornita dal metodo che sfrutta l'utilizzo di potenziali artificiali.

Capitolo 7

Il Problema del Coverage

Nel seguente capitolo viene ora presentato un problema che occupa un ruolo di rilievo nell'ambito della gestione e del coordinamento di flotte di *robot.* La sua risoluzione permette infatti di risolvere in modo efficiente problemi di ambito comune quali operazioni di ricerca e recupero, svolgimento di differenti *task* in ambienti ostili o pericolosi per l'uomo, esplorazione, videosorveglianza e molti altri compiti. Stiamo parlando del problema che in letteratura viene definito con il termine *coverage* e che sostanzialmente consiste nel disporre tutti gli agenti operativi in una determinata area di lavoro garantendone la sua massima copertura.

7.1 Introduzione

Grazie al continuo sviluppo tecnologico avvenuto in questi ultimi anni è divenuto possibile coordinare le azioni di un elevato numero di agenti attraverso reti di comunicazione costruite *ad-hoc.* In questo modo i *robot* possono eseguire compiti complessi e grazie alla loro rete di comunicazione, non agiranno più in maniera indipendente, ma potranno distribuirsi le varie operazioni ottenendo così un risultato al problema considerato molto più efficiente. I potenziali vantaggi derivanti dall'utilizzo di una flotta di agenti sono quindi numerosi, tra questi vi sono il miglioramento delle prestazioni ottenibili nell'esecuzione del compito (sia esso di rilevamento o altro) e la maggiore robustezza, garantita dalla maggiore ridondanza. Considerazione fondamentale per permettere di operare con un *team* cooperativo di agenti è che la rete di rilevamento mobile tra i *robot* sia dinamica, permettendo così di far fronte a variazioni date dalle movimentazioni degli agenti nello spazio operativo, risultando dunque fortemente adattativa.

Solitamente, in letteratura, i problemi di *coverage* sono suddivisi principalmente in due sotto-categorie che risultano essere concettualmente diverse tra loro; si parla infatti di:

- problema di *sensor coverage*, che consiste nel caratterizzare e ottimizzare il concetto di qualità di servizio fornito dalla rete di sensori adattativa in un generico ambiente dinamico. Per raggiungere questo obiettivo si ha appunto l'esigenza di ottimizzare, nell'ambiente operativo, la determinazione della posizione da assumere da parte dei sensori posti a bordo dell'agente e quindi anche la posizione dei *robot* stessi; a tale proposito sarà quindi fondamentale risolvere un problema di copertura ottima dell'area di lavoro. La rete di sensori,

opportunamente disposta, consente così di valutare al meglio le variazioni che avvengono nell'ambiente in cui agiscono gli agenti;

- problema di *coverage path planning*, che consiste nel far analizzare a un singolo agente tutti i punti di una specificata traiettoria, pianificata in modo da raggiungere un determinato obiettivo. Questa operazione consente, andando appunto a coprire l'intera traiettoria fornita, di sapere, ad esempio, se tale percorso risulta essere sicuro o meno per una persona.

In questa trattazione in particolare vogliamo concentrarci sulla prima sotto-categoria, o meglio vogliamo occuparci del problema di *coverage* generale che, come detto, consiste nel disporre nel modo migliore gli agenti sulla superficie di lavoro in modo da renderli adibiti a eseguire successivamente un compito specifico. Per risolvere tale problema generale dovremo dunque trovare un metodo di ottimizzazione locale per gestire la distribuzione ottimale degli agenti sulla superficie di lavoro.

I problemi di ottimizzazione locale sono generalmente dei problemi spaziali riguardanti l'allocazione delle risorse, ad esempio dove collocare l'ufficio postale in una città oppure dove posizionare il *server* principale di una rete; essi ricoprono quindi un ruolo centrale per il corretto funzionamento, la corretta copertura e la continuità di servizio della rete in esame. Le principali tecniche per risolvere tale problema di ottimizzazione locale sono basate su metodi numerici che analizzeremo nel prosieguo e che consentono, tramite la costruzione di opportune *mesh and grid,* di ricavare una mappa contenente la corretta distanza tra i punti fondamentali della rete che ne garantiscono la sua massima copertura, risolvendo così il problema di *coverage*.

In questo capitolo si vuole dunque sviluppare uno specifico algoritmo che sfrutti la struttura di controllo distribuito *DPC*, descritta nel Capitolo 4, per la risoluzione del problema di *coverage* per una flotta di *robot*. Grazie a tale struttura di controllo, e grazie all'ausilio della *webcam*, l'algoritmo saprà sempre, come visto nel caso della gestione del problema di *collision avoidance*, dove si trovano tutti i vari agenti all'interno del piano operativo. In particolare si progetterà e implementerà un algoritmo di *coverage* di tipo:

- Adattativo: l'algoritmo permette di trovare una soluzione qualunque sia il numero di agenti utilizzati nel piano di lavoro e qualunque sia la loro posizione di partenza, e quindi il loro conseguente stato iniziale, purché dichiarati esplicitamente nelle condizioni iniziali che definiscono il problema.

- Distribuito: il comportamento di ogni veicolo dipende solo dalla propria posizione, da quella dei sui vicini, e dalla geografia dello spazio operativo. La struttura della comunicazione tra gli agenti non dovrà essere necessariamente rigida, ma, come visto, essa sarà dinamica, ovvero potrà cambiare in base alla struttura evolutiva della rete. Quindi anche se i *robot* si spostano nel piano di lavoro la struttura evolverà con il loro spostamento permettendo in ogni condizione la comunicazione corretta tra i vari agenti. Questo è un pregio dato dalla struttura di controllo *DPC*, come si è analizzato nel Capitolo 4, tuttavia è importante ribadire ora tale concetto proprio a fronte del problema esaminato. Quindi l'algoritmo conoscerà a ogni istante discreto k la posizione di tutti i *robot* nel piano di lavoro e comunicherà ai singoli agenti la posizione dei propri vicini, consentendo così una corretta gestione del problema.

- Asincrono: questo significa che l'algoritmo può essere utilizzato in una rete in cui sono implementati agenti che si muovono a differenti velocità, con differenti capacità computazionali e quindi non uniformi a un'unica tipologia di *robot.* In più, il nostro algoritmo non richiede una sincronizzazione globale, quindi le proprietà di convergenza degli agenti sono preservate anche a fronte di ritardi nelle propagazioni delle comunicazioni tra i vari *robot* vicini.

- Asintoticamente corretto: l'algoritmo garantisce una decrescenza monotona della funzione di costo del problema di ottimizzazione locale che verrà presentato nel seguito e che consente un corretto raggiungimento dell'obiettivo finale voluto.

Utilizzando dunque la struttura logica parallela, sviluppata nel Capitolo 3, e adottando la tecnica di controllo predittivo *DPC*, mostrata nel Capitolo 4, è possibile ottenere la corretta convergenza dello specifico *robot* mobile verso il corrispondente *goal*, garantendo così l'efficiente risoluzione di un problema di *tracking,* come già illustrato anche nei capitoli precedenti. La differenza in questo caso sta nel fatto che i *goal* degli agenti saranno ricavati dalla risoluzione del problema di *coverage* data dalla minimizzazione di una determinata funzione di costo su una specifica regione di interesse. Andiamo ora nel prossimo paragrafo a illustrare nel dettaglio come fare a ottenere una soluzione del problema esaminato.

7.2 Algoritmo specifico

Dopo aver fornito una panoramica generale sul problema di *coverege* per *robot* mobili, si vuole ora descrivere dettagliatamente l'algoritmo utilizzato in questa tesi per risolvere tale problema. Nel considerare il problema di *coverage* da noi risolto si è effettuata una forte ipotesi iniziale che permette di semplificare il problema. Si considera infatti che la superficie di lavoro sia completamente nota, dalla quale è possibile ricavare, tramite una *webcam* posta sopra di essa le posizioni degli agenti a ogni istante k. In sostanza, grazie a questa ipotesi si potrà dunque evitare di eseguire un'operazione di esplorazione della superficie per calcolare le dimensioni, ma si considererà di fatto che tali misure siano note a priori. Questo ci permette quindi di poter trattare solamente problemi di *coverage* di tipo statico che presentano un piano di lavoro noto, non affrontando casi in cui la superficie operativa risulti incognita o vari dinamicamente durante l'esecuzione dell'algoritmo. Tale fatto potrà essere fonte di possibili sviluppi futuri.

Si vanno ora a introdurre e descrivere nello specifico i vari *set*, i parametri utilizzati e le nozioni necessarie per comprendere i vari passi dell'algoritmo implementato.

7.2.1 Set-up e definizione del problema di ottimizzazione locale

Definiamo ora un insieme formato da una flotta di M *robot* che sono contenuti in un ambiente Q, il quale risulta essere un politopo convesso in $\mathbb{R}^2$; inoltre indichiamo con la notazione $\|\cdot\|$ una funzione di distanza Euclidea. Si prenda poi una determinata funzione mappatrice $\phi\colon Q \to \mathbb{R}_+$[1] che rappresenta la distribuzione di densità di probabilità che un dato da misurare venga generato nei punti q dell'insieme Q.

[1] Si rappresenta con $\mathbb{R}_+$ il *set* dei numeri reali non negativi

Prendiamo ora un insieme $P = (p_1, \ldots, p_M)$ che contiene le posizioni in coordinate (x, y) di tutti gli agenti mobili che operano sulla superficie, ognuno dei quali potrà appunto muoversi all'interno di Q. Supponendo che il dato da misurare sia generato nel punto $q \in Q$ la qualità delle performance di rilevamento di tale dato, da parte del sensore presente su un *robot* alla posizione p_i, si degrada in funzione della distanza $\| q - p_i \|$ tra q e p_i. Descriviamo quindi questo degrado con una funzione differenziabile decrescente $f \colon \mathbb{R}_+ \to \mathbb{R}_+$; di conseguenza $f(\| q - p_i \|)$ fornisce un'informazione di come è distribuita, su una determinata superficie, la capacità di rilevamento dell'agente che sarà massima vicina al punto q e tenderà man mano a decrescere più ci allontaniamo da esso. Lo scopo del *coverage* è quello di sfruttare la presenza di più sensori, agenti in modo tale da massimizzare l'efficienza di rilevazione di un dato generato su Q. Le difficoltà principali dipendono da due fatti:

- non è nota deterministicamente la posizione in cui il dato è generato, ma la sua posizione è definita attraverso la densità di probabilità ϕ;
- man mano che ci si allontana dal punto di generazione del dato si perde capacità di rilevarlo, il che è rappresentato dalla funzione f.

In un contesto multi agente, si suppone che ogni sensore venga "allocato" a una regione di spazio politopica $\mathcal{W}_i$, $i = 1, \ldots, M$. I *set* $\mathcal{W}_i$ sono definiti in modo da costituire una partizione dello spazio Q, definita $\mathcal{W} = \{W_1, \ldots, W_M\}$. In base a quanto detto, la soluzione del problema di *coverage* così affrontato equivale a trovare le posizioni p_i e le partizioni $\mathcal{W}_i$ tali che la seguente cifra di costo sia massimizzata:

$$\mathcal{H}(P, \mathcal{W}) = \sum_{i=1}^{M} \int_{W_i} f(\| q - p_i \|) \, \phi(q) \, dq \tag{7.1}$$

Osserviamo inoltre un'importante proprietà: se scambiamo la posizione di due agenti, insieme alla loro associata regione di dominio, il valore della funzione di ottimizzazione locale $\mathcal{H}$ non è influenzato da ciò. Equivalentemente, se Σ_n denota il gruppo discreto di permutazioni di n elementi, allora:

$$\mathcal{H}(p_1, \ldots, p_M, W_1, \ldots, W_M) = \mathcal{H}\left(p_{\sigma(1)}, \ldots, p_{\sigma(M)}, W_{\sigma(1)}, \ldots, W_{\sigma(M)}\right)$$

per tutti gli scambi di posizione $\sigma \in \Sigma_n$.

7.2.2 Partizione di Voronoi

Per una posizione fissata degli agenti, la partizione ottima di Q è data dal cosiddetto diagramma di Voronoi:

$$\mathcal{V}(P) = \{V_1, \ldots, V_M\}$$

generato univocamente dalle posizioni dei *robot* $(p_1, \ldots, p_M)$ in modo tale che:

$$V_i = \{q \in Q \mid \| q - p_i \| \leq \| q - p_j \|, \forall j \neq i\}$$

Per maggiori dettagli e chiarimenti si rimanda a [22], dove viene inoltre riportata la dimostrazione relativa alla costruzione del digramma.

Quando due regioni di Voronoi V_i e V_j sono adiacenti, ovvero presentano un lato in comune tra loro, i due *robot* da esse contenuti, alle posizioni p_i e p_j, verranno

chiamati vicini di Voronoi. A questo punto possiamo quindi definire il *set* dei vicini di Voronoi di p_i che è denotato con $\mathcal{N}(i)$; chiaramente $j \in \mathcal{N}(i)$ se e solo se $i \in \mathcal{N}(j)$. Inoltre per analizzare la partizione effettuata sulla superficie Q ci vengono in aiuto alcuni concetti derivanti dalla teoria Euclidea; in particolare a noi interessa il fatto che se Q è un politopo convesso in uno spazio Euclideo N-dimensionale, il confine di ogni regione V_i non è altro che l'unione di politopi convessi $(N-1)$-dimensionali. Nel nostro caso, lavorando su una superficie, si tratta quindi di segmenti. A fronte di quanto appena detto possiamo ridefinire la cifra di costo (7.1) sostituendo a una generica partizione $\mathcal{W}$ di Q la partizione di Voronoi $\mathcal{V}(P)$, a sua volta dipendente esclusivamente da P (e dalla tipologia di Q). Si scrive cioè:

$$\mathcal{H}_{\mathcal{V}}(P) = \mathcal{H}(P, \mathcal{V}(P))$$

Si ottiene quindi:

$$\mathcal{H}_{\mathcal{V}}(P) = \int_Q \min_{i \in \{1,\dots,M\}} f(\| q - p_i \|) \phi(q)\, dq = E_{(Q,\phi)} \left[\min_{i \in \{1,\dots,M\}} f(\| q - p_i \|) \right] \tag{7.2}$$

che è la funzione di ottimizzazione locale, che può essere interpretata come un valore atteso ricavato tramite un'operazione di minimizzazione. Il problema di *coverage* si presenta usualmente in letteratura come il problema di trovare i valori di p_i, $i = 1, \dots, M$ che massimizzano la cifra di costo appena ricavata. Inoltre è possibile scrivere che:

$$\frac{\partial \mathcal{H}_{\mathcal{V}}}{\partial p_i}(P) = \frac{\partial \mathcal{H}}{\partial p_i}(P, \mathcal{V}(P)) = \int_{V_i} \frac{\partial}{\partial p_i} f(\| q - p_i \|) \phi(q)\, dq$$

cioè le derivate parziali di $\mathcal{H}_{\mathcal{V}}$, rispetto alla posizione del *robot i,* dipendono solamente dalla posizione dell'agente e dalla posizione dei suoi vicini di Voronoi. Per risolvere tale problema attraverso un algoritmo del gradiente è quindi sufficiente adottare un approccio di tipo distribuito.

7.2.3 Centroidi delle partizioni di Voronoi

Richiamiamo ora alcune definizioni base associate a una data regione $V \subset \mathbb{R}^2$ e una funzione di densità di massa ρ. Per massa generalizzata di una data regione si intende:

$$M_V = \int_V \rho(q)\, dq$$

per centroide o centro di massa si intende:

$$C_V = \frac{1}{M_V} \int_V q \rho(q)\, dq$$

vine inoltre definito il momento polare d'inerzia come:

$$J_{V,p} = \int_V \| q - p \|^2 \rho(q)\, dq$$

Per il teorema degli assi paralleli [26] possiamo scrivere che:

$$J_{V,p} = J_{V,C_V} + M_V \| p - C_v \|^2$$

dove $J_{V,C_V} \in \mathbb{R}_+$ è definito come il momento polare d'inerzia della regione V circa il centroide C_V.

Si supponga, inizialmente, che i W_i siano fissati. Consideriamo di nuovo il problema di ottimizzazione (7.2), che si traduce in un problema di minimizzazione della seguente funzione:

$$\tilde{\mathcal{H}}(P, \mathcal{W}) = \sum_{i=1}^{M} \int_{W_i} \| q - p_i \|^2 \, \phi(q) \, dq \tag{7.3}$$

Si ricorda infatti che (7.2) è definita in base a f decrescente, mentre ora (7.3) contiene $\| q - p_i \|^2$, che è crescente con $\| q - p_i \|$. Sfruttando ora il teorema degli assi paralleli [26] e introducendo le definizioni appena viste circa massa generalizzata, centroide e momento polare d'inerzia, possiamo scrivere che:

$$\tilde{\mathcal{H}}_{\mathcal{V}}(P) = \sum_{i=1}^{M} J_{V_i,C_{V_i}} + \sum_{i=1}^{M} M_{V_i} \| p_i - C_{V_i} \|^2$$

Qui la funzione di densità di massa è $\rho = \phi$, che come detto in precedenza rappresenta la distribuzione di densità che una misura di informazione o di probabilità venga correttamente rilevata sulla superficie Q. Ciò è conveniente per definire:

$$\mathcal{H}_{\mathcal{V}1} = \sum_{i=1}^{M} J_{V_i,C_{V_i}}$$

$$\mathcal{H}_{\mathcal{V}2} = \sum_{i=1}^{M} M_{V_i} \| p_i - C_{V_i} \|^2$$

Queste formule ci permettono di giungere alla conclusione che i punti di minimo locale, per risolvere il problema di ottimizzazione locale della funzione $\mathcal{H}_{\mathcal{V}}$, risultano essere i centroidi delle singole superfici di Voronoi, partizioni della superficie complessiva Q. Questo significa che ogni posizione del *robot* p_i che soddisfi simultaneamente la proprietà di essere una posizione generatrice per la regione di Voronoi V_i considerata e un suo centroide

$$C_{V_i} = \underset{p_i}{argmin} \; \mathcal{H}_{\mathcal{V}}(P)$$

risulta essere una soluzione del problema di ottimizzazione locale. Di conseguenza avremo che l'insieme dei centroidi appartenenti alle varie partizioni $V_i \subset Q$ costituiranno l'insieme dei minimi locali, e quindi l'insieme delle soluzioni del problema di ottimizzazione locale, che a loro volta permetteranno una corretta risoluzione del problema di *coverage*. Naturalmente, ricordiamo che le configurazioni dei centroidi di Voronoi dipenderanno, oltre che dal numero di agenti operanti nella superficie di lavoro, dalla specifica distribuzione della funzione di densità $\phi(q)$. Per ulteriori e più approfonditi chiarimenti si rimanda a [24].

7.2.4 Calcoli su poligoni con densità uniforme

Occupiamoci ora del problema, che andremo effettivamente a implementare e risolvere, che rappresenta un caso semplificato del problema di *coverage* analizzato.

A tale proposito si svilupperà un algoritmo che permetta di risolvere il problema di *coverage* in cui è presente una funzione di densità $\phi(q)$ uniforme. Come detto, la regione di Voronoi V_i considerata è un poligono convesso, cioè un politopo in $\mathbb{R}^2$, con N_i vertici etichettati $\{(x_0, y_0), \ldots, (x_{N_i-1}, y_{N_i-1})\}$, come mostrato in Figura 7.1, dove viene utilizzata la notazione $(x_{N_i}, y_{N_i}) = (x_0, y_0)$ in modo da accomunare

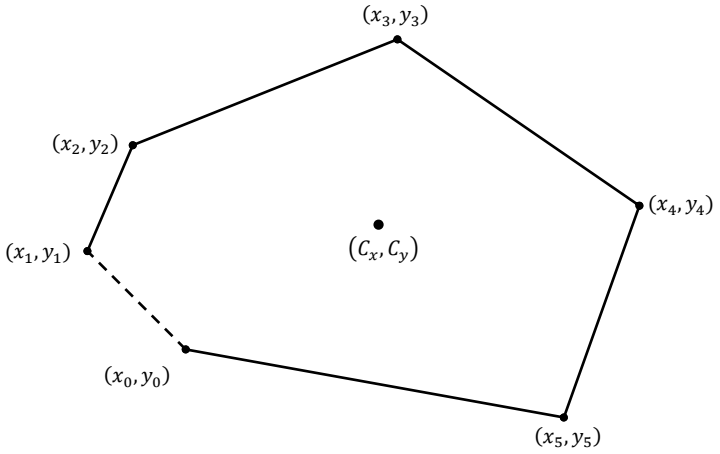


Figura 7.1: Notazione convenzionale per un poligono convesso

il vertice iniziale e finale del poligono, da cui si inizia a contare. In più assumiamo, come anticipato, che la funzione di densità sia costante e definita come $\phi(q) = 1$. Utilizzando le definizioni introdotte in precedenza nel paragrafo 7.2.3, e risolvendo i corrispondenti integrali, possiamo ottenere le seguenti espressioni in forma chiusa:

$$\begin{cases} C_{V_i,x} = & \frac{1}{6M_{V_i}} \sum_{k=0}^{N_i-1} (x_k + x_{k+1})(x_k y_{k+1} - x_{k+1} y_k) \\ C_{V_i,y} = & \frac{1}{6M_{V_i}} \sum_{k=0}^{N_i-1} (y_k + y_{k+1})(x_k y_{k+1} - x_{k+1} y_k) \end{cases}$$

in cui:

$$M_{V_i} = \frac{1}{2} \sum_{k=0}^{N_i-1} (x_k y_{k+1} - x_{k+1} y_k)$$

Presentiamo ora di seguito una semplice formulazione per definire il momento polare d'inerzia:

$$\begin{cases} \bar{x}_k = & x_k - C_{V_i,x} \\ \bar{y}_k = & y_k - C_{V_i,y} \end{cases}$$

per $k \in \{0, \ldots, N_i - 1\}$. Successivamente calcoliamo il momento polare d'inerzia del poligono rispetto il suo centroide $J_{V_i,C}$ ottenendo così:

$$J_{V_i,C_{V_i}} = \sum_{k=0}^{N_i-1} (\bar{x}_k \bar{y}_{k+1} - \bar{x}_{k+1} \bar{y}_k) \left(\bar{x}_k^2 + \bar{x}_k \bar{x}_{k+1} + \bar{x}_{k+1}^2 + \bar{y}_k^2 + \bar{y}_k \bar{y}_{k+1} + \bar{y}_{k+1}^2 \right)$$

La dimostrazione di queste formule è basata sulla scomposizione del poligono come l'unione di più triangoli disgiunti; per ulteriori dettagli e chiarimenti si rimanda a [23], dove sono inoltre contenute le espressioni analoghe per il caso di politopi nello spazio generico $\mathbb{R}^N$.

7.2.5 Algoritmo implementato

Il nostro algoritmo andrà quindi a calcolare *off-line* i vari punti in cui i *robot* dovranno posizionarsi per effettuare la miglior copertura possibile della data superficie di lavoro. Successivamente, si andrà a impostare tali punti calcolati, sotto forma di coordinate (x, y), come *goal* dei vari agenti che, grazie alla solita struttura di controllo *DPC*, riusciranno a raggiungere tali obiettivi risolvendo così il problema analizzato. Si descrive dettagliatamente il procedimento adottato nel nostro algoritmo implementato per la risoluzione del problema di *coverage* per una superficie statica e nota con funzione di densità $\phi(q)$ uniforme. Utilizzando la struttura di controllo predittivo distribuito *DPC* per il controllo della flotta di agenti, l'algoritmo specifico si baserà su alcuni semplici passaggi, ripetuti in modo iterativo, che verranno ora descritti nel seguito:

1. Definizione del poliedro Q che rappresenta il piano di lavoro in cui agiranno i nostri agenti; esso come detto verrà definito staticamente a priori nelle condizioni iniziali e rimarrà sempre costante per tutta la durata del problema.

2. Acquisizione delle posizioni reali $P_R = \{p_1^R, \ldots, p_M^R\}$ dei *robot* nel piano di lavoro tramite la *webcam* e definizione di $P^o = P_R$. Per ogni iterazione $k \geq 0$:

   (a) Calcolo delle partizioni di Voronoi $\mathcal{V}\left(P^k\right)$ della superficie Q dati i punti p^k.

   (b) Calcolo dei corrispondenti centroidi $C_{V_i}^k$ delle superfici di Voronoi V_i^k, $i = 1, \ldots, M$ trovate al punto precedente. Come visto, questi centroidi corrispondono a dei minimi locali per la cifra di costo $\mathcal{H}_{\mathcal{V}2}$.

   (c) Si pone $p_i^k = C_{V_i}^k$.

   (d) Se $\parallel p_i^k - p_i^{k-1} \parallel > soglia$, allora $k = k + 1$ e si ritorna al punto 2.(a). Altrimenti si passa allo *step* 3.

3. Assegnamento delle coordinate dei centroidi ricavate al punto precedente come coordinate dei *goal* dei vari *robot* utilizzati nel piano di lavoro, risolvendo in questo modo il problema di *coverage* per la superficie in esame.

Come definito, l'algoritmo prevede di ripetere gli *step* 2(a), 2(b) e 2(c) finché la posizione (virtuale) degli agenti non converge a dei valori stazionari. Tali valori verranno poi presi come *goal* per il problema di *tracking* dei *robot*, in modo tale da effettuare la misurazione ottimale dello spazio di lavoro.

7.3 Risultati sperimentali

Si vuole ora effettuare una prova sperimentale implementando sul sistema reale l'algoritmo descritto nella sezione precedente in merito al problema di *coverage*. In particolare, si effettua un *test* andando a implementare sugli agenti *e-puck* l'algoritmo che utilizza la tecnica di controllo predittivo *DPC* e presenta la struttura con logica di controllo in parallelo, quindi con anello di *Feedback Linearization* posto internamente all'agente, come trattato al paragrafo 3.4.

In sostanza l'esperimento consiste nel risolvere un problema di *coverage,* data la nostra solita superficie di lavoro di dimensioni 115 x 66 cm^2, utilizzando tre *robot*

e-puck, posti rispettivamente alle coordinate iniziali di $(10, 8)$, $(20, 49)$ e $(90, 50)$. L'algoritmo quindi opererà eseguendo in sequenza due distinte operazioni: la prima che consiste nel calcolare, *off-line*, il diagramma di Voronoi e i rispettivi centroidi delle superfici di Voronoi che permettono la risoluzione del problema di *coverage*. A tale proposito viene mostrato in Figura 7.2 l'evoluzione del diagramma di Voronoi,

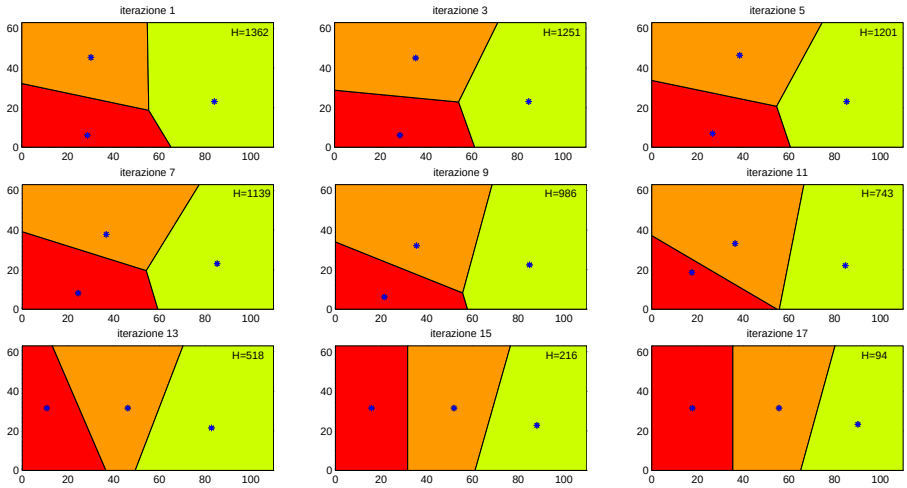


Figura 7.2: Evoluzione del diagramma di Voronoi all'aumentare del numero di iterazioni dell'algoritmo

con rispettivi centroidi, rappresentati tramite asterischi blu, calcolato all'aumentare del numero di iterazioni dell'algoritmo. Inoltre, sempre nella Figura 7.2 viene mostrato, rispetto al corrispondente diagramma di Voronoi considerato, il valore della cifra di costo $\tilde{\mathcal{H}}$ del problema di ottimizzazione (7.3) che coerentemente a quanto voluto tende a decrescere verso un punto di minimo. Nello specifico, dopo un numero di iterazioni pari a 10000, si ottiene il diagramma di Voronoi mostrato in Figura 7.3 con i corrispondenti centroidi delle tre superfici posti alle coordinate: $(18, 32)$, $(55, 31)$ e $(91, 30)$. Quanto appena ricavato ci consente di dimostrare che, utilizzando questo

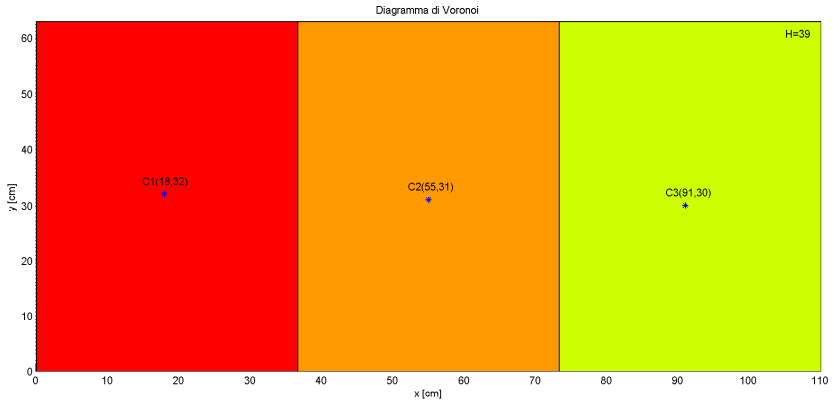


Figura 7.3: Diagramma di Voronoi in cui sono calcolati i centroidi finali delle tre superfici di Voronoi corrispondenti ai tre *robot* utilizzati e che consentono di risolvere il problema di *coverage*

algoritmo, il calcolo *on-line* dei centroidi delle superfici di Voronoi risulterebbe infattibile in quanto i tempi computazionali richiesti sarebbero troppo elevati. Nella Figura 7.4 viene inoltre mostrato come, con il passare delle iterazioni, la norma tra

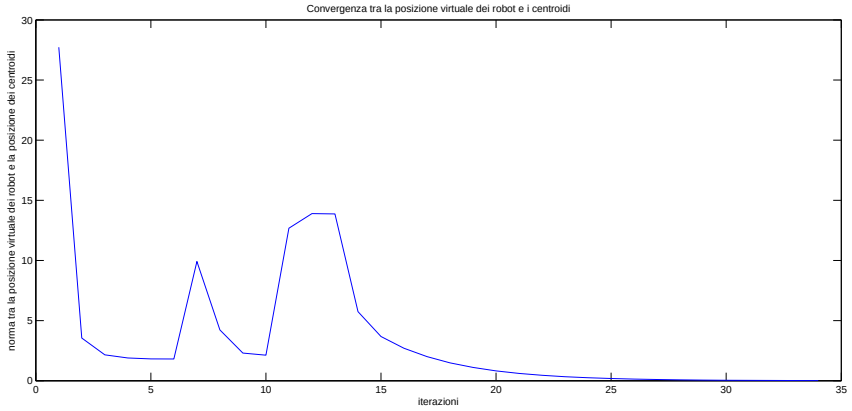


Figura 7.4: Grafico che mostra l'effettiva convergenza tra la posizione virtuale dei *robot* e i centroidi calcolati

la posizione virtuale dei *robot* e le coordinate dei centroidi calcolate tenda a zero. Infatti si vede chiaramente dalla figura come tale norma dopo un numero di iterazioni circa pari a 22 si annulli e ciò implica che le posizioni virtuali dei *robot* convergono ai centroidi calcolati. Fino a questo punto i *robot* di fatto non si sono ancora effettivamente mossi dalla loro posizione iniziale; questa prima operazione è infatti servita a ricavare i loro rispettivi obiettivi finali. Ora verrà svolta dall'algoritmo la seconda operazione, che consiste nell'applicare il classico controllo predittivo *DPC* che, come visto, garantisce una corretta risoluzione dei singoli problemi di *tracking*. In questo modo i singoli agenti in base alla loro posizione iniziale, si recheranno al *goal*, precedentemente calcolato, più vicino a loro. Questo algoritmo così impostato garantisce, per definizione di centroidi delle superfici di Voronoi, che le traiettorie dei *robot*, qualunque siano le loro posizioni di partenza, non si intersechino mai evitando così qualsiasi rischio di collisione tra gli agenti stessi.

Durante la prova sperimentale, in accordo con quanto visto nei capitoli precedentemente illustrati, sono stati utilizzati per il controllo *DPC* i seguenti dati: tempo di campionamento $\tau = 0.5$, Q e R matrici di peso dello stato e dell'ingresso, definite e utilizzate nel paragrafo 3.2 per la costruzione del predittore di *Kalman,* scelte in modo da essere matrici identità.

Utilizzando ora le seguenti figure andiamo ad analizzare i risultati ottenuti dal sistema reale e su esse effettuiamo alcune importanti osservazioni. In primo luogo si analizza come si comporta la traiettoria degli agenti, concentrandoci appunto sulla posizione dei tre *robot*. Dalla Figura 7.5 si vede bene come tutti e tre gli agenti seguano in maniera corretta il proprio riferimento, portandosi così ai rispettivi *goal* che corrispondono ai centroidi delle superfici di Voronoi, ricavati in precedenza e mostrati nella Figura 7.3. Essi, come detto, permettono quindi di ottenere la corretta risoluzione del problema di *coverage* per la superficie considerata. In particolare vediamo che il *robot* 1, partendo dalle coordinate $(10, 8)$, raggiunge il centroide appartenente alla sua corrispondente superficie di Voronoi, ovvero raggiunge il *goal* con coordinate $(18, 32)$, che risulta infatti essere quello più vicino rispetto la sua posi-

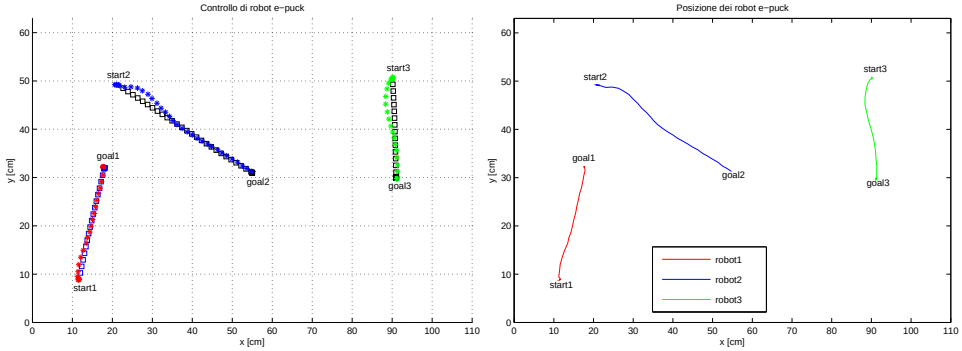


Figura 7.5: Traiettoria mantenuta dai tre *robot* nella risoluzione del problema di *coverage*

zione iniziale. Stesso ragionamento può essere fatto in maniera analoga anche per tutti e due gli altri agenti. In questo modo, per come è stato costruito l'algoritmo, le traiettorie dei tre *robot* non potranno mai incontrarsi garantendo così a priori anche una soluzione per il delicato problema di *collision avoidance*. I due grafici mostrati in Figura 7.5 riportano in sostanza la medesima informazione dove, come fatto per tutta la trattazione, il grafico di sinistra mostra il riferimento degli agenti sotto forma di quadratini, mentre la posizione reale è espressa dagli asterischi colorati in modo differente per ogni *robot*. Il grafico di destra rappresenta invece solamente delle linee continue, colorate sempre in modo differente per ogni *robot*, che interpolano le posizioni reali dei tre agenti. Analizziamo ora la Figura 7.6 dove vengono riportate le

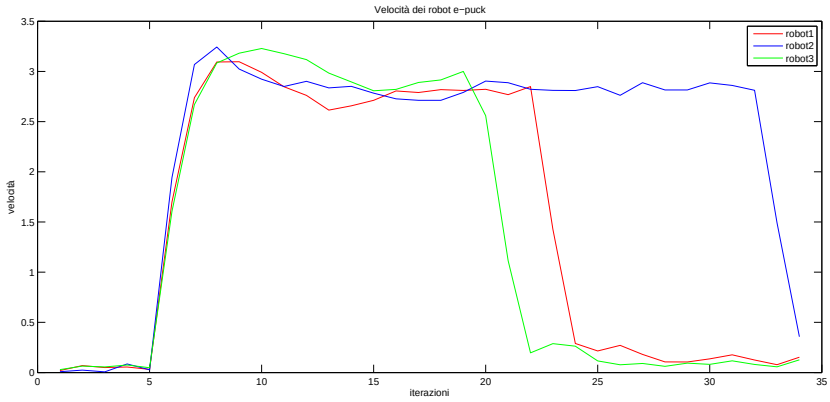


Figura 7.6: Velocità mantenuta dai tre *robot* nella risoluzione del problema di *coverage*

velocità dei tre agenti; da qui si vede come i *robot* 1 (il rosso) e 3 (il verde) arrivino ai propri *goal* circa allo stesso istante di tempo e successivamente, in attesa che anche il *robot* 2 (il blu) arrivi al proprio obiettivo, viene applicata a essi un'accelerazione nulla in modo da non farli così allontanare dai loro *goal*. Inerentemente a tali grafici non vengono fatte ulteriori osservazioni a riguardo delle prestazioni degli agenti in quanto, di fatto, si sta utilizzando nuovamente il controllo predittivo distribuito *DPC* già ampiamente analizzato in precedenza.

Ripetiamo ora la stessa prova sperimentale facendo questa volta partire i *robot* da tre posizioni iniziali vicine tra loro e diverse rispetto a quelle del *test* fatto in precedenza, a tal proposito si vuole dimostrare quanto già anticipato, ovvero che in base alla posizione iniziale da cui partono gli agenti essi si dirigeranno tramite una traiettoria rettilinea al corrispondente *goal*, cioè centroide, più vicino appartenente alla propria regione di Voronoi evitando in questo modo problemi di *collision avoidance*. Nella Figura 7.7 viene infatti riportata la traiettoria seguita dai tre *robot*

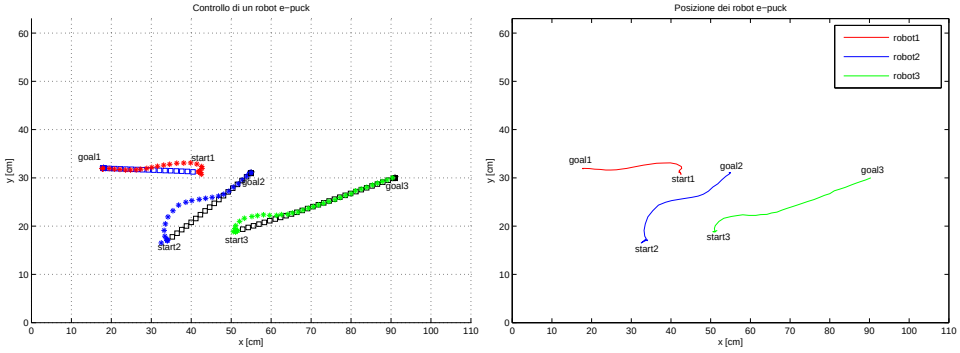


Figura 7.7: Traiettoria mantenuta dai tre *robot* nella risoluzione del problema di *coverage*

e-puck che partono dalle rispettive coordinate $(41, 31)$, $(33, 19)$ e $(50, 20)$. In questo caso i centroidi delle superfici di Voronoi calcolati, quindi i rispettivi *goal* imposti agli agenti, sono ancora posizionati, come nella prova sperimentale precedente; essi infatti avranno coordinate $(55, 31)$, $(18, 32)$ e $(91, 30)$ come è possibile osservare in Figura 7.8. Ciò è dovuto al fatto, come visto nel precedente paragrafo, che i centroidi delle

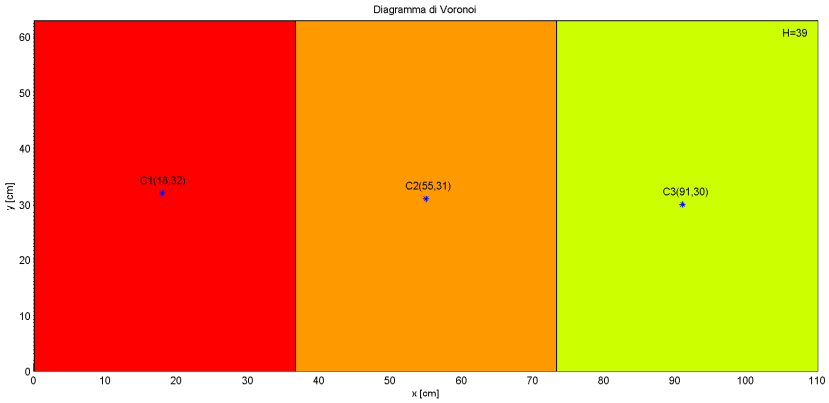


Figura 7.8: Diagramma di Voronoi in cui sono calcolati i centroidi finali delle tre superfici di Voronoi corrispondenti ai tre *robot* utilizzati e che consentono di risolvere il problema di *coverage*

superfici di Voronoi variano sì dinamicamente in base alle posizioni iniziali dei *robot,* ma nel nostro algoritmo essi sono calcolati *off-line* e una volta trovato il digramma di Voronoi esso sarà quindi definito in modo statico per una data superficie. Poiché la superficie di lavoro utilizzata è sempre la stessa, i centroidi calcolati, quindi i *goal* dei

robot impostati, risulteranno sempre posti alle stesse coordinate nonostante varino le posizioni iniziali degli agenti. Naturalmente ciò non sarebbe più vero nel caso di un calcolo *on-line* del diagramma di Voronoi e dei rispettivi centroidi, tuttavia questo non costituirà oggetto di analisi per questa trattazione. Anche per questo *test*, tramite il grafico della Figura 7.9, mostriamo la convergenza tra la posizione virtuale dei *robot* e la posizione dei centroidi calcolati a ogni iterazione del ciclo operativo. In

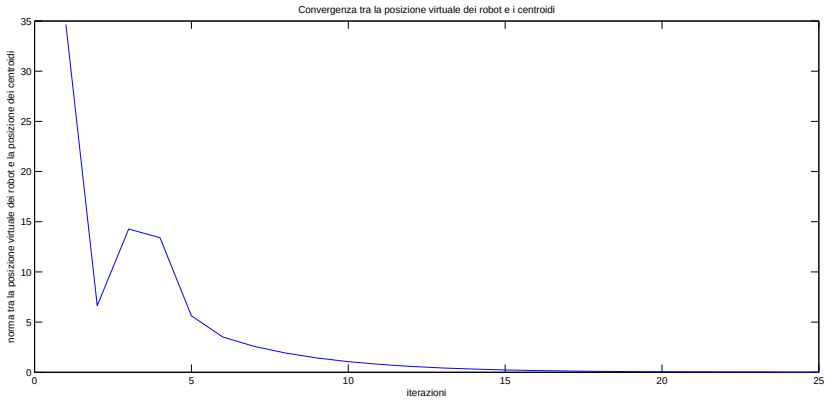


Figura 7.9: Grafico che mostra l'effettiva convergenza tra la posizione virtuale dei *robot* e i centroidi calcolati

questo caso la convergenza tra i due elementi analizzati avviene circa all'iterazione 11, ovvero molto prima rispetto al *test* precedente dove, come mostrato nella Figura 7.3, essa avveniva circa all'iterazione 22. Ciò quindi dipende dalle posizioni iniziali degli agenti che causeranno, a seconda di come vengono prese, una più o meno rapida convergenza tra i due valori analizzati. Come al solito, successivamente, il mantenimento della traiettoria e il conseguente corretto raggiungimento degli obiettivi finali da parte dei *robot* viene garantito da un'adeguata modulazione delle velocità gestita dalla tecnica di controllo predittivo *DPC*.

Capitolo 8

Conclusioni e sviluppi futuri

In questa tesi si è illustrato inizialmente il modello base di un *robot* uniciclo, descrivendone la sua cinematica e analizzando in modo completo l'apparato sperimentale. Successivamente, partendo dalla struttura con logica sequenziale sviluppata in [13], in cui tutti gli algoritmi di controllo erano implementati in modo distribuito su un unico calcolatore, si è passati a una struttura con logica in parallelo. Sfruttando il performante processore a bordo del *robot epuck,* sono stati rielaborati gli algoritmi in modo che alcune parti di essi fossero eseguite localmente all'interno dell'agente stesso, rendendo così il sistema di controllo complessivo più efficiente. Per effettuare tale operazione è stato necessario implementare un predittore di *Kalman* per ricavare una stima dello stato e in particolare della velocità lineare dell'agente. Proprio la conoscenza di quest'ultima grandezza si è rivelata essere fondamentale per permettere il trasferimento dell'anello di controllo più interno, ovvero l'algoritmo di *Feedback Linearization*, sul microcontrollore a bordo di ciascun *robot*. Ciò ha permesso di aumentare notevolmente la frequenza di campionamento, migliorando l'accuratezza e la reattività del sistema e potendo così avere una netta diminuzione del tempo di campionamento τ, come mostrato nel Capitolo 3. A tale proposito, un interessante sviluppo futuro consisterà nello spostare internamente al *robot* anche la soluzione delle equazioni del predittore di *Kalman* avendo così a ogni passo di campionamento, direttamente all'interno dell'agente, un valore stimato della velocità. Altra soluzione per ricavare il valore della velocità degli agenti potrebbe essere quella di utilizzare gli *encoder* presenti sulle ruote dei *robot*, andando così a effettuare una misura effettiva della velocità reale; questo consentirebbe di semplificare notevolmente alcuni problemi trattati in questa tesi e potrebbe inoltre evitare l'utilizzo del predittore di *Kalman* per stimare lo stato del sistema.

Un secondo aspetto degno di nota riguarda la complessità computazionale dell'intero algoritmo di *test* sviluppato. Riuscendo a spostare il problema di *Feedback Linearization* internamente ai singoli agenti è stato infatti possibile, come detto, diminuire i tempi di campionamento dell'algoritmo di controllo esterno implementato tramite un solo calcolatore per emulare la presenza di diversi regolatori locali tra loro comunicanti. Grazie a ciò si è quindi riusciti a non influire eccessivamente sulle prestazioni dell'algoritmo anche a fronte di velocità rilevanti di movimento dei *robot* reali. In aggiunta, per migliorare ulteriormente le prestazioni e velocizzare il tutto, si è adottata la soluzione di semplificare la complessità computazionale dell'algoritmo cercando di riformulare la componente non lineare del problema di ottimizzazione, richiesto dall'algoritmo di controllo predittivo implementato, a vincoli puramente li-

neari, in modo tale da usare algoritmi di ottimizzazione veloci, robusti ed efficienti. A tal proposito sono stati riscritti i vincoli relativi ai problemi di *avoidance* in termini lineari, come mostrato nel Capitolo 5, consentendo così di giungere a un problema puramente quadratico di più facile risoluzione.

Arrivati a questo punto si è quindi ottenuto un sistema nettamente più performante rispetto a quello presentato in [13], tuttavia sarebbe possibile aumentare ulteriormente la sua efficienza andando a rivedere con attenzione ciascuna delle sezioni di codice e ricorrendo, per quanto possibile, a implementazioni di più basso livello che consentano di evitare il ricorso all'ambiente *MATLAB*. Sempre nell'ottica dell'efficienza degli algoritmi utilizzati, sarebbe inoltre interessante trasferire l'algoritmo di rilevamento della *webcam* in linguaggio *C* facendo ricorso alle librerie grafiche *OpenCV* e andando a valutare anche la disponibilità di eventuali algoritmi di ottimizzazione disponibili direttamente come librerie per il linguaggio *C*.

A partire dalla struttura logica così ottenuta e relativa all'implementazione del codice, è stato realizzato un controllo *MPC* con implementazione distribuita, in modo da permettere una risoluzione al problema di coordinamento di un *set* di *robot* mobili in movimento planare. Tale tecnica di controllo predittivo gerarchico consente di ottenere diversi vantaggi rispetto alle numerose tecniche proposte in letteratura. In particolare, la struttura di controllo *DPC* permette di separare il problema di garantire robustezza nell'inseguimento della traiettoria di riferimento, rispetto a quello della generazione della traiettoria stessa, rendendo così quest'ultimo più semplice sia del punto di vista teorico che computazionale. Inoltre la tecnica robusta *Tube-Based MPC*, presentata al Capitolo 4, permette di definire a priori un intorno della traiettoria di riferimento nel quale si ha la certezza di trovare l'agente, indipendentemente dai disturbi che agiscono su di esso, purché rispettino le ipotesi introdotte. Tale struttura di controllo, come detto, permette di ottenere vantaggi anche dal punto di vista computazionale. Essa infatti richiede di risolvere un problema di programmazione lineare quadratica, per il quale esistono efficienti algoritmi risolutivi, contrariamente ai complessi problemi non lineari che saremmo costretti solitamente ad affrontare in letteratura per queste tipologie di problemi.

In seguito sono stati presi in esame i problemi di *obstacle* e *collision avoidance* andando, come detto, a riscrivere i vincoli presentati in [13] in modo lineare e inserendoli poi opportunamente all'interno del problema di ottimizzazione. Per utile confronto con lo schema di controllo predittivo, si è implementata una struttura di controllo, basata su una legge algebrica, in grado di trattare i medesimi problemi di *avoidance* tramite potenziali artificiali (*APF*) andando in seguito a effettuare un confronto diretto con i risultati ottenuti tramite la teoria di controllo predittivo distribuito. In questo ambito, relativamente alla struttura di controllo predittivo, futuri lavori potranno essere dedicati a utilizzare nell'algoritmo i dati forniti dai sensori di bordo degli agenti, con particolare attenzione ai sensori di prossimità disposti perimetralmente lungo il *robot*. Questo potrebbe fornire valide soluzione o alternative al problema di *obstacle avoidance* di tipo dinamico in cui il *robot*, non conoscendo a priori la posizione e il raggio degli ostacoli nello spazio operativo, riuscirà comunque a evitarli arrivando così al suo determinato *goal*. Un ulteriore e differente sviluppo potrebbe essere quello di implementare un algoritmo di visione, come quello atto all'individuazione della posizione e dell'orientamento degli agenti, che consenta di rilevare, sempre tramite *webcam* posta sopra il piano di lavoro, la posizione del centro dell'ostacolo e il suo raggio, in modo da poter poi passare tali valori agli algoritmi

di *avoidance* già implementati in questa trattazione e analizzati nel Capitolo 5.

Infine è stato affrontato il problema del *coverage*, che si sta ritagliando un ruolo di sempre maggiore rilevanza in letteratura in quanto consente la risoluzione di innumerevoli problemi, tra cui il rilevamento. Nel Capitolo 7 si è infatti visto come risolvere un problema di *coverage* statico con funzione di densità uniforme. Sarebbe interessante, a tale proposito, proseguire su questa strada andando a sviluppare un algoritmo risolutivo per problemi di *coverage* dinamico in modo da non considerare più la superficie di lavoro nota a priori, ma trasmettendo ai vari agenti i dati relativi tramite operazioni di esplorazione oppure, nel caso di ambienti limitati, sfruttando la *webcam* posizionata sopra il piano di lavoro.

In conclusione, nella tesi, grazie alla struttura logica in parallelo unita alla struttura di controllo predittivo presentata, si è andati a migliorare notevolmente le capacità prestazionali dei singoli agenti in termini di robustezza ai vari disturbi presenti sul sistema, reattività e velocità nel raggiungimento dell'obiettivo. Naturalmente, tale trattazione, ricoprendo un ambito molto ampio, lascia spazio a interessanti sviluppi futuri, di cui alcuni citati in precedenza, volti ad affrontare problematiche sempre più attuali rispetto allo sviluppo tecnologico raggiunto.

Appendice A

Aspetti implementativi

A.1 Firmware del microcontrollore

Come spiegato, per utilizzare la struttura logica in parallelo presentata al Capitolo 3 si è dovuto trasportare l'anello relativo alla *Feedback Linearization* internamente a ogni *robot* caricandolo direttamente sul suo microcontrollore. A tale proposito viene dedicata questa appendice in cui si illustra, nello specifico, il *firmware* scritto sul microcontrollore degli agenti utilizzato per effettuare tutti i vari *test* grazie ai quali è stato poi possibile riportare i risultati ottenuti in questa trattazione. In tale codice viene inoltre fatto riferimento alle varie librerie fornite dal produttore e liberamente disponibili in rete per la gestione di basso livello dei singoli dispositivi di cui il *robot e-puck* è dotato quali sensori e attuatori. Riportiamo ora di seguito il codice effettivamente implementato:

```
1    // Includiamo le varie librerie e i vari sottoprogrammi che ci servono
2 per interagire col robot
3    #include "e_init_port.h"
4    #include "e_uart_char.h"
5    #include "matlab.h"
6    #include "e_motors.h"
7    #include "e_led.h"
8    #include "e_epuck_ports.h"
9    #include "utility.h"
10   #include <stdlib.h>
11   #include <string.h>
12   #include <ctype.h>
13   #include <stdio.h>
14   #include <math.h>
15   #include <time.h>
16
17
18 // Definiamo il protocollo che ci permette la comunicazione tra PC e robot
19 via bluetooth
20   #define uart_send_text(msg)
21        do { e_send_uart1_char(msg,strlen(msg));
22             while(e_uart1_sending());
```

```
23              }
24         while(0)
25
26
27 // Definiamo alcune costanti che avranno visibilità globale
28    #define PI 3.14159265358979f
29    #define STEPST02PI 1300
30    #define T_SPEED 300
31
32 //regoliamo la temporizzazione in modo che sia in secondi
33    #ifdef CLOCKS_PER_SEC
34    #undef CLOCKS_PER_SEC
35    #endif
36    #define CLOCKS_PER_SEC 16000000
37
38 //inizio del main
39    int main(void)
40    {
41         int selector; //indica la posizione del selettore del robot
42         long i;
43         time_t timer1; //variabile di temporizzazione
44         double dif_sec;
45         char car;
46         float w;
47         int dato[4];
48         int speedl; //velocità ruota sinistra
49         int speedr; //velocità ruota destra
50         float E = 5.2f;
51         float R = 2.05f;
52         float kk = 500.0f/PI; // step/rad
53         float vR;
54         float vL;
55         float wR;
56         float wL;
57         float ax = 0.0f, ay = 0.0f, v = 0.0f, theta = 0.0f;
58
59
60 //Definiamo il tempo di campionamento che viene dato in secondi(da regolare
61 in base alla temporizzazione effettuata)
62         float tau = 0.01f;
63
64
65 //Inizializzazione delle porte, della UART e dei motori.
66         e_init_port();
67         e_init_uart1();
68         e_init_motors();
69         e_set_speed_left(0);
```

```
70          e_set_speed_right(0);
71
72
73  //Lettura della posizione del selettore; la scelta della posizione zero
74  permette di mettere in attesa il microcontrollore così da consentire la
75  programmazione.
76          selector=getselector();
77          if(selector==0){
78               while(1);
79               }
80
81
82  //Nel caso in cui il selettore non sia nella posizione zero inizia il ciclo
83  del programma, la UART viene pulita dai dati in attesa e il LED7 conferma
84  il risultato corretto del set-up.
85          LED7 = 1;
86          int flag = 0;
87
88
89  //Inizio del loop infinito.  Il microcontrollore attende che vi sia un comando
90  sulla seriale e nel caso questo sia quello corretto legge i quattro interi
91  seguenti e li interpreta rispettivamente come accelerazioni, velocità e
92  angolo theta dell'agente.
93          while(1){
94               time(&timer1);
95               e_getchar_uart1(&car);
96
97  //avremo bisogno di inserire inizialmente un carattere 'v' seguito da quattro
98  interi per passare accelerazioni, velocità e angolo al robot
99          if (car=='v'){
100              while(e_receive_int_from_matlab(dato,4)==0);
101              LED4=0;
102
103 //dividiamo per 1000 in modo da avere concordanza con lo scrip di Matlab
104 (dove invece moltiplicheremo tali dati per 1000)
105              ax = dato[0]/1000.0f;
106              ay = dato[1]/1000.0f;
107              v = dato[2]/1000.0f;
108              theta = dato[3]/1000.0f;
109              car = '\0';
110              flag = 1;
111
112         }//Chiusura dell'if in risposta al comando 'v'
113
114
115         while(!e_ischar_uart1()) {
116              timer1=time(NULL);
117
```

```
118 //Aggiornamento Feedback Linearizzation
119 //Gestione dei casi in cui venga data una velocità circa 0
120             if (v >= 0.0f && v < 0.001f)
121                  v = 0.001f;
122             if (v <= 0.0f && v > -0.001f)
123                  v = -0.001f;
124
125 //Applico le equazioni per la Feedback Linearization
126             w = (1/v)*(ay*cos(theta)-ax*sin(theta));
127             v = v +tau*(ax*cos(theta)+ay*sin(theta));
128             theta = theta+tau*w;
129
130 // Conversione in steps/s di rotazione
131 // Velocità lineare ruote
132             vR = v + E * w / 2.0f;
133             vL = v - E * w / 2.0f;
134 // Velocità angolare
135             wL = vL / R;
136             wR = vR / R;
137 // Velocità in step/s
138             speedl = myround(kk * wL);
139             speedr = myround(kk * wR);
140
141 //Saturazione delle velocità inviate al microcontrollore
142             if (speedl>1000) speedl = 1000;
143             if (speedl<-1000) speedl =-1000;
144             if (speedr>1000) speedr = 1000;
145             if (speedr<-1000) speedr =-1000;
146
147 //Stampo sul terminale i valori di velocità
148             char s[150];
149             sprintf(s, "v:  %f; sL: %d; sR: %d", v, speedl, speedr);
150             uart_send_text(s);
151
152 //Attuazione delle velocità alle ruote
153             e_set_speed_left(speedl);
154             e_set_speed_right(speedr);
155             for(i=0;i<30000;i++){
156                  asm("nop");
157             }
158
159 //Facciamo in modo che la temporizzazione del ciclo venga gestita in secondi
160             while(((difftime (time(NULL),timer1))/(float)CLOCKS_PER_SEC)
161 < tau) {
162                  if (flag == 1) {
163                       LED4=1;
164                       flag=0;
165                  }
```

```
166                 }
167             }
168
169     }//Chiusura del loop infinito
170
171     }//Termine del main
```

Bibliografia

[1] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.C. Zufferey, D. Floreano, A. Martinoli. «The e-puck, a Robot Designed for Education in Engineering», (2010). *E-Puck Robot Website*. http://www.e-puck.org.

[2] R. Scattolini e L. Magni. «Complementi di Controlli Automatici». Pitagora Editrice, Bologna (2006).

[3] Y. Cao et al. «Distributed Containment Control for Multiple Autonomous Vehicles With Double-Integrator Dynamics: Algorithms and Experiments». In: *Control Systems Technology* 19, pp. 929-938, (2011).

[4] M. Farina e R. Scattolini. «Distributed predictive control: A non-cooperative algorithm with neighbor-to-neighbor communication for linear systems». In: *Automatica* 48.6, pp. 1088-1096, (2012).

[5] M. Flint, M. Polycarpou e E. Fernandez-Gaucherand. «Cooperative path-planning for autonomous vehicles using dynamic programming". In *World Congress* 15, pp. 1303-1303, (2004).

[6] I. Kolmanovsky e N.H. McClamroch. «Developments in nonholonomic control problems». In *Control Systems, IEEE* 15.6, pp. 20-36, (1995).

[7] École Polytechnique Fèdèrale de Lausanne. *E-Puck Robot Website*. http://www.e-puck.org.

[8] L. Marin, M. Vallès, Á. Valera, P. Albertos «Implementation of a Bug Algorithm in the E-puck from a Hybrid Control Viewpoint». In: *Methods and Models in Automation and Robotics (MMAR), 2010 International Conference 15*, pp. 174-179, (2010).

[9] A.S. Matveev, H. Teimoori e A.V. Savkin. «A method for guidance and control of an autonomous vehicle in problems of border patrolling and obstacle avoidance». In: *Automatica* 47.3, pp. 515-524, (2011).

[10] A.S. Matveev, C. Wang e A.V. Savkin. «Real-time navigation of mobile robots in problems of border patrolling and avoiding collisions with moving and deforming obstacles». In: *Robotics and Autonomous Systems* 60.6, pp. 769-788, (2012).

[11] W. Ran, W. Beard e E.M. Atkins. «Information consensus in multivehicle cooperative control». In: *Control Systems Magazine* 27, pp. 71-82, (2007).

[12] B. Wenfeng, H. Lina e S. Yunfeng. «An improved A* algorithm in path planning». In: *Computer, Mechatronics, Control and Electronic Engineering (CMCE), 2010 International Conference* 3, pp. 235-237, (2010).

[13] A. Perizzato. «Navigazione e coordinamento di agenti mobili mediante tecniche di controllo predittivo distribuito». Tesi di laurea, Politecnico di Milano, (2013).

[14] A.P. Aguiar, J.P. Hespanha. «Trajectory-Tracking and Path-Following of Underactuated Autonomous Vehicles With Parametric Modeling Uncertainty». In: *IEEE Transactions on Automatic Control* 52.8, pp. 1362-1377, (2007).

[15] L. Di Stefano. «Elaborazione dell'Immagine». Tesi di laurea, Politecnico di Torino, (2003).

[16] C.B. Madsen, C.S. Andersen. «Optimal landmark selection for triangulation of robot position*». In: *Robotics and Autonomous Systems* 23, pp. 277-292, (1998).

[17] C.C. Fung, H. Eren, Y. Nakazato. «Position sensing of mobile robots for team operations», In: *IEEE IMTC/94 Conf.; Japan,* pp. 749-752, (1994).

[18] E.B. Lee, L. Markus. «Foundations of Optimal Control Theory». New York: Wiley, (1967).

[19] D.Q. Mayne, S.V. Rakovic, R. Findeisen, F. Allgöwer. «Robust output feedback model predictive control of contrained linear systems». In. *Automatica* 46, pp. 1217-1222, (2006).

[20] S. Mastellone, D.M. Stipanovic, C.R. Graunke, K.A. Intlekofer, M.W. Spong. «Formation Control and Collision Avoidance for Multi-agent Non-holonomic Systems: Theory and Experiments». In: *The International Journal of Robotics Research* 27, pp.107-126, (2008).

[21] M. Farina, G. Betti, L. Giulioni, R. Scattolini. «An approach to distributed predictive control for tracking - theory and applications». In: *IEEE Transactions on Control Systems Technology* 15.4, pp. 1558-1566, (2014).

[22] A. Okabe, B. Boots, K. Sugihara, S.N. Chiu. «Spatial Tessellations: Concepts and Applications of Voronoi Diagrams», 2nd ed, ser. Wiley Series in Probability and Statistics. New York: Wiley, (2000).

[23] C.Cattani, A. Paoluzzi. «Boundary integration over linear polyhedra». In: Comput.-Aided Des., 22.2, pp. 130-135, (1990).

[24] A. Okabe, B. Boots, K. Sugihara. «Nearest neighborhood operations with generalized Voronoi diagrams: A review», In: Int. J. Geograph. Inform. Syst. 8.1, pp. 43-71, (1994).

[25] S. Bittanti. «Teoria della Predizione e del Filtraggio». Pitagora Editrice, Bologna (2005).

[26] G. Diana, F. Cheli. «Dinamica dei sistemi meccanici». Polipress, (2010).