POLITECNICO DI MILANO

Corso di Laurea Magistrale in Ingegneria dell'Automazione

Dipartimento di Elettronica, Informazione e Bioingegneria

NAVIGAZIONE E COORDINAMENTO DI AGENTI MOBILI MEDIANTE TECNICHE DI CONTROLLO PREDITTIVO DISTRIBUITO

Relatore: Prof. Riccardo Scattolini
Correlatore: Prof. Marcello Farina

Tesi di Laurea di:
Andrea Perizzato
matricola 783491

Anno Accademico 2012-2013

A mia mamma

Sommario

Negli ultimi anni la diffusione di agenti autonomi per lo svolgimento di attività a supporto e sostegno dell'uomo è sempre più marcata grazie anche allo sviluppo di tecnologie che permettono l'implementazione di sofisticate tecniche di controllo. Queste ultime si stanno orientando verso soluzioni di tipo distribuito poichè presentano maggiore robustezza rispetto a malfunzionamenti riducendo allo stesso tempo lo sforzo computazionale complessivo. Allo stesso modo, motivazioni economiche, prestazionali e di flessibilità orientano la direzione di sviluppo verso l'impiego di un maggior numero di agenti semplici, dotati di capacità di collaborazione, rispetto a pochi, ma con elevate capacità sensoriali e computazionali, portando però a più complessi problemi di controllo e coordinamento.

Lo scopo della tesi è di fornire una strategia di controllo unificante per diversi problemi inerenti il controllo di agenti mobili in diverse applicazioni. Successivamente all'analisi di una soluzione di controllo predittivo robusto verranno proposte delle soluzioni a classici problemi del controllo di formazione, del contenimento e della navigazione in ambiente sconosciuto. Particolare attenzione sarà data agli aspetti computazionali relativi all'elaborazione della legge di controllo poichè l'obiettivo ultimo è l'utilizzo di tali tecniche in contesti reali in cui i tempi a disposizione per portare a temine le operazioni di calcolo sono spesso limitati. Tutti gli algoritmi proposti saranno validati attraverso prove sperimentali mediante l'impiego di tre robot mobili.

Abstract

Nowadays, robotics applications are growing at a fast rate. In fact, the progress in technology allows for the development and the implementation of advanced control algorithms. The interest has shifted from the design of few and complex robots to the use of a large number of simpler and more inexpensive agents, endowed with cooperation capabilities. Besides the reduction of the overall cost, the use of simpler agents leads to a more robust and flexible system architecture which, however, requires advanced control techniques to provide cooperation between robots. Distributed control solutions have been studied as a result of these needs due to their fault tolerance features and to the lower computational requirements.

The main purpose of this work is to provide a general unifying solution for some common problems concerning the control of multi-agent systems. Starting from the study of a robust predictive control algorithm, solutions to the traditional problems, such as formation control, containment and navigation in unknown environment, will be given. Specific attention will be paid to computational aspects. In fact, the main goal is to provide powerful techniques which can be easily deployed in real robots with limited computational effort and time. Finally, the effectiveness of the algorithms will be demonstrated through real experiments with a three-robot test facility.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Al giorno d'oggi, le applicazioni che coinvolgono agenti autonomi stanno aumentando a un ritmo elevato e si prevede che tale tendenza continui anche nel futuro grazie a progressi tecnologici che apriranno la strada a nuove possibilità applicative. Gli agenti autonomi, detti anche *Unmanned Autonomous Vehicles*, UAV, vengono utilizzati principalmente in ambienti ostili per l'uomo sia dal punto di vista della sicurezza che della accessibilità. Infatti, essi sono particolarmente diffusi in applicazioni militari, ma anche in missioni interplanetarie, di sorveglianza e ispezione di luoghi remoti, quali spazi ristretti o profondità marine, per la ricerca e il recupero di dispersi in seguito ad eventi tragici, come ad esempio terremoti o inondazioni. Interessanti sono i risvolti applicativi nella direzione del supporto all'uomo nella vita di tutti i giorni, come ad esempio l'assistenza alla guida mediante la comunicazione con i veicoli vicini (*cruise control*), la pulizia automatica dei pavimenti della casa e il taglio automatico dell'erba del giardino. Tutte queste semplici applicazioni evidenziano come l'utilizzo di agenti autonomi sia già molto diffuso anche al di fuori della ricerca e di contesti in cui è richiesto un grande sforzo economico.

Grazie ai progressi tecnologici degli ultimi anni, l'interesse in questo settore si è spostato dall'utilizzo di un numero limitato di agenti complessi, dotati di tutte le funzionalità necessarie per portare a termine le richieste, verso l'impiego di un maggior numero di robot più semplici, ma in grado di collaborare fra di loro con il fine di evadere i compiti assegnati. Questo cambiamento è dovuto a diversi fattori, quali il minor costo complessivo, la possibilità di elaborare algoritmi distribuiti che richiedono un minor sforzo computazionale, ma soprattutto la

maggiore tolleranza dei dispositivi rispetto a guasti e, infine, alla sua più facile estensione e riusabilità. D'altro canto, l'impiego di agenti in cooperazione richiede opportuni algoritmi di coordinamento e organizzazione, che, come detto, permettano al sistema nel suo complesso di portare a termine i compiti assegnati. Tali algoritmi sono oggetto di ricerca nel mondo scientifico: l'obietto della ricerca è di ottenere i comportamenti desiderati con implementazioni a basso costo computazionale facilmente programmabili in locale negli agenti. Esempi di studi in questa direzione possono essere trovati in [13] e [19].

In letteratura sono analizzate svariate tecniche per il controllo di agenti autonomi, ma negli ultimi anni si è diffuso il controllo predittivo o *Model Predictive Control*, MPC, nato in ambito industriale per il controllo di processi di elevate dimensioni. Questa tecnica è basata sulla soluzione di un problema di ottimizzazione costruito sulla predizione del comportamento del sistema e presenta il notevole vantaggio di poter considerare in modo implicito vincoli di ogni genere, quali limitazioni dell'azione di controllo e dello stato. A causa degli elevati tempi computazionali generalmente necessari, le applicazioni tipiche di questa metodologia di controllo sono legate alla regolazione di impianti industriali con dinamiche sufficientemente lente da poter utilizzare intervalli di campionamento adatti alla risoluzione dei problemi di ottimizzazione che elaborano l'azione di controllo. Tuttavia, grazie agli sviluppi tecnologici sia nella ricerca di algoritmi di soluzione efficienti che nella velocità computazionale degli elaboratori, al giorno d'oggi è possibile sfruttare il controllo predittivo anche per sistemi con dinamiche più veloci poichè si è in grado di risolvere tali problemi in tempi molto brevi, anche mediante sistemi integrati di dimensioni ridotte, che possono essere facilmente montati a bordo di agenti robotizzati. Negli ultimi anni, la ricerca in questo campo si è focalizzata nell'elaborare problemi di ottimizzazione che garantiscano robustezza del sistema di controllo nel rispetto di disturbi esterni o discrepanze fra il modello e il sistema reale, portando a tecniche di controllo dette di *MPC Robusto* [23]. Un altro ramo della ricerca ha elaborato tecniche di controllo predittivo distribuite [26], tra cui l'algoritmo *Distributed Predictive Control*, DPC [7], che si pongono l'obiettivo di suddividere il problema di ottimizzazione completo in sotto problemi di ridotta complessità, più velocemente risolvibili localmente.

L'analoga direzione di sviluppo del controllo predittivo e della ricer-

ca nei sistemi multi-agente, rende questa soluzione di controllo particolarmente adatta al coordinamento di agenti con le proprietà descritte precedentemente. Oltre al vantaggio computazionale ottenuto dalla suddivisione del problema di controllo complessivo in sotto-problemi più semplici, una soluzione distribuita permette di ridurre lo scambio complessivo di dati e di rendere gli agenti più affidabili. D'altro canto, ciò presenta maggiori difficoltà nella coordinazione delle decisioni prese localmente da ciascuno, in modo tale che, nel complesso, vengano soddisfatti determinati vincoli e requisiti. In letteratura sono state proposte diverse soluzioni a questo problema, basate su strategie diverse quali, ad esempio, la programmazione dinamica [9] e la teoria dei giochi [11] per lo studio della convergenza.

Gran parte degli agenti robotizzati disponibili in ambiente sia di ricerca che applicativo è caratterizzata da una struttura ad uniciclo poichè fornisce un'elevata manovrabilità, ma allo stesso tempo rappresenta un'importante sfida dal punto di vista del controllo. Infatti, grazie al lavoro svolto da Brockett in [29], è noto che essi rientrano nella classe di sistemi anolonomi, tali per cui non è possibile trovare una legge di controllo stabilizzante mediante una retroazione continua. Dunque, il controllo di tali agenti non può essere realizzato attraverso le classiche tecniche lineari, ma richiede l'applicazione di opportune soluzioni, spesso molto complesse o non globalmente definite, come descritto, ad esempio in [14].

1.1 Problemi nel controllo di sistemi multi-agente

Si presentano ora i diversi problemi trattati e risolti in questa tesi nell'ambito del controllo e coordinamento di sistemi multi-agente, con l'obiettivo di fornire al lettore un inquadramento generale delle problematiche, applicazioni e soluzioni disponibili al giorno d'oggi.

1.1.1 Navigazione verso un obiettivo in presenza di ostacoli noti

Il problema della navigazione autonoma verso un obiettivo consiste nel guidare l'agente verso un punto noto dello spazio di lavoro, detto *goal*. Inoltre, se nell'ambiente sono presenti degli ostacoli, supposti noti, si parla di navigazione con *obstacle avoidance*. Si tratta quindi di generare un percorso privo di collisioni (*path planning*) nel rispetto della geome-

tria dell'agente e dell'ambiente circostante, seguito dalla generazione di una traiettoria (*trajectory planning*) che attribuisce al percorso una legge temporale costruita sulla base della dinamica dell'agente sotto controllo.

Il problema in esame non presenta molte applicazioni pratiche poichè non è frequente avere una conoscenza completa dell'ambiente circostante tale da poter calcolare a priori una traiettoria priva di collisioni, ma fornisce interessanti soluzioni e punti di partenza per affrontare problemi di navigazione più complessi, come ad esempio il controllo di agenti in formazione o il controllo di crociera, detto *cruise control*, di veicoli stradali nel quale si richiede che questi ultimi si muovano rispettando il percorso delineato dalla strada, evitando collisioni con i mezzi adiacenti, come rappresentato in Fig. 1.1.

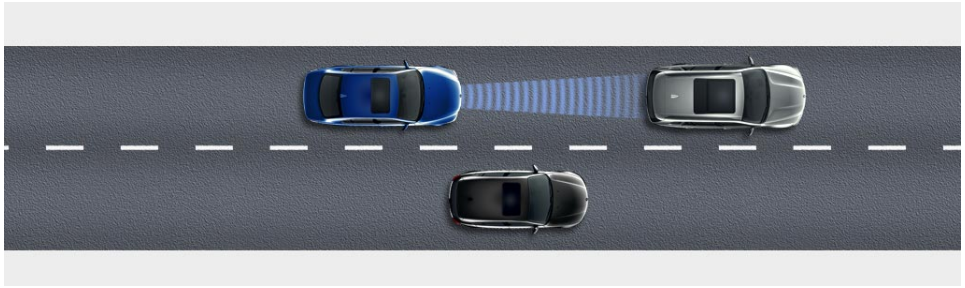


Figura 1.1: Cruise Control

In letteratura sono state proposte svariate tecniche per la soluzione di questo problema. Focalizzando l'attenzione su quelle basate sul controllo predittivo, si ricade molto spesso in problemi di ottimizzazione molto complessi, definiti da funzioni di costo non lineari e non convesse. La complessità di tali problemi è dovuta sia alla dinamica non lineare degli agenti che ai metodi utilizzati per descrivere l'ambiente e gli ostacoli. Ad esempio, l'utilizzo di potenziali artificiali per l'*obstacle avoidance* porta inevitabilmente a funzioni obiettivo molto complesse e computazionalmente impegnative da risolvere a causa della non linearità della funzione potenziale stessa. Ciò rende l'implementazione di tali sistemi di controllo irrealizzabile localmente nei vari agenti portando quindi a soluzioni che, seppur molto valide dal punto di vista teorico, non riscontrano altrettanti meriti nell'ambito applicativo. Esempi di questi algoritmi sono riportati in [6] e [28].
Al di fuori dell'ambito del controllo MPC, sono state sviluppate diverse tecniche per la soluzione di questo problema. Ad esempio, l'algoritmo

A*, presentato in [31], permette di risolvere tale problema mediante la ricerca del cammino minimo in un opportuno grafo. Affichè tale algoritmo converga verso una soluzione è necessario definire la funzione di costo euristica utilizzata dall'algoritmo stesso per la computazione del cammino (sub)ottimo. Poichè la scelta di quest'ultima non è un problema banale e le prestazioni di A* ne sono strettamente legate, tanto da poter anche non convergere a causa di scelte inadatte, esso potrebbe non essere la soluzione più adatta in un contesto generale. Alternativamente, sono presenti soluzioni costruite su basi probabilistiche, quali l'algoritmo RRT, discusso in [30], che individua traiettorie prive di collisioni mediante la costruzione di un albero a partire dalla generazione casuale di punti nello spazio di lavoro. Il maggiore problema di questa tecnica, anche se molto efficiente, è che non permette di includere alcun tipo di ottimizzazione nel processo di generazione della traiettoria in quanto essa viene generata, come detto, esplorando in modo casuale lo spazio di lavoro. Sebbene sia diffusa nell'ambito dei manipolatori robotici, potrebbe non essere la scelta ottimale nel contesto del controllo di agenti autonomi.

1.1.2 Controllo di formazione

Come precedentemente introdotto, la tendenza nella ricerca e nelle applicazioni di sistemi di agenti si sta dirigendo verso l'utilizzo di più agenti di ridotta complessità e costo, grazie agli innumerevoli vantaggi derivanti, quali maggiore robustezza, possibilità di ridondanza, efficienza e flessibilità. Ciò evidenzia la necessità di sviluppare opportune tecniche di controllo che garantiscano un coordinamento delle traiettorie seguite dai diversi agenti con l'obiettivo ultimo di portare a termine gli obiettivi imposti. Da qui nasce il problema del controllo in formazione che consiste nel guidare un gruppo di agenti lungo una traiettoria definita mantenendo una configurazione desiderata, intesa, in questo contesto, come posizione reciproca o, appunto, formazione.

Questo problema presenta un'ampia varietà di applicazioni. In ambito militare, ad esempio, un gruppo di veicoli deve mantenere una desiderata formazione per garantire la massima copertura di una zona soggetta a monitoraggio. Si considerino ad esempio formazioni di aerei da guerra o di veicoli dedicati al pattugliamento marino o terrestre. Nel campo aerospaziale invece una possibile applicazione è la seguente: si vuole che un insieme di satelliti si muova mantenendo una struttura

Figura 1.2: Esempio di Formazione

fissa con il fine di minimizzare il consumo di carburante ed espandere le capacità sensoriali complessive. Infine, si citano anche applicazioni in via di sviluppo, quali sistemi autostradali automatizzati, detti *Automated Highway Systems* AHS, in cui si cerca di massimizzare il volume di veicoli nella rete attraverso la movimentazione di questi ultimi in gruppi ad una desiderata velocità.

Le soluzioni a questo problema possono essere suddivise in tre categorie, a seconda della struttura del sistema di agenti. Il controllo *leader-follower* definisce delle relazioni fra gli agenti in modo tale che i cosiddetti *follower* inseguano, nel rispetto della formazione i propri *leader*, i quali sono autonomamente guidati verso uno specifico obiettivo. La seconda categoria riguarda i controlli detti a *Struttura Virtuale* in cui si vuole che gli agenti si muovano come se fossero vincolati meccanicamente, rimuovendo quindi la necessità di definire le relazioni precedenti. Infine, si hanno le tecniche *Behavior-based* che forniscono a ciascun robot diversi comportamenti anche contrastanti, quali ad esempio, il mantenimento di una formazione, il raggiungimento di un obiettivo e così via, presentando indubbiamente difficoltà nello studio analitico. Ulteriori dettagli e approfondimenti di queste tematiche saranno forniti nel prosieguo della tesi.

1.1.3 Problema del contenimento

Nel problema del contenimento si suddividono, ancora una volta, gli agenti in *leader* e *follower* e si vuole che questi ultimi restino sempre contenuti all'interno della regione dello spazio definita dai *leader*. Il problema è spesso ulteriormente complicato dalla limitata rete di comunicazione fra gli agenti che non permette a ciascuno di conoscere lo stato di tutti gli altri, dando luogo a decisioni inerenti la traiettoria da percorrere basate solo su informazioni locali. Per queste motivazioni, questa problematica rientra nel più generale *problema del consenso* in cui agenti appartenenti ad un gruppo si accordano in base a interazioni locali, con il fine di raggiungere un obiettivo globale.

Il contenimento trova applicazione al problema della misurazione e rilevazione di determinate aree dello spazio di lavoro. Questa problematica può essere infatti risolta mediante l'utilizzo di due categorie di agenti: i *follower* dedicati ad effettuare le misure mediante appositi sensori e i *leader* che hanno l'obiettivo di guidare i primi nelle stabilite aree. In questo modo è possibile considerare separatamente il problema di navigazione, gestito dai *leader*, da quello di posizionamento dei sensori e misura, risolto dai *follower*. Una seconda, ma simile, applicazione riguarda la navigazione in presenza di un numero limitato di agenti provvisti di sensori adatti a identificare l'ambiente circostante, come illustrato in Fig. 1.3.

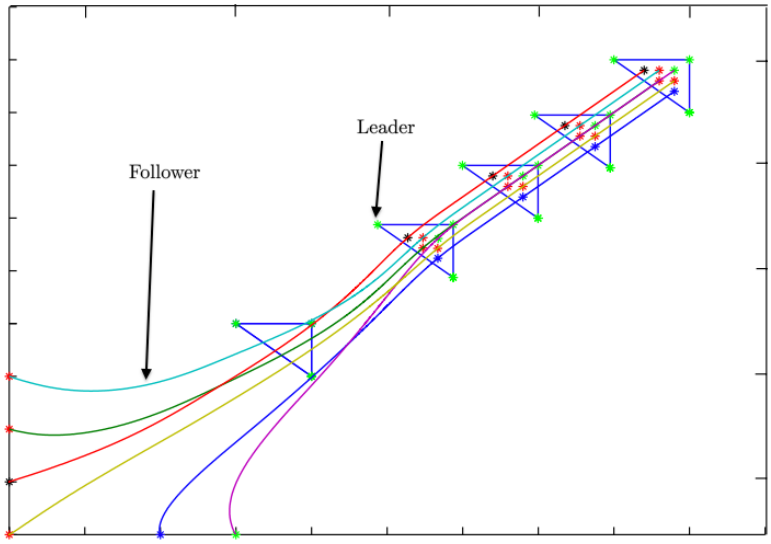


Figura 1.3: Esempio di Contenimento

Le soluzioni generalmente presenti in letteratura sono basate sulla teoria dei grafi ed elaborano leggi di controllo a partire proprio dalle matrici di tali grafi. Esempi di queste tecniche sono riportate in [3] e [17]. Sebbene tali algoritmi garantiscano una convergenza asintotica dei *follower* all'interno della regione definita dai *leader*, non è sempre possibile trarre conclusioni egualmente soddisfacenti riguardo la robustezza di tali tecniche, compromettendone quindi la successiva applicazione pratica.

1.1.4 Problema dei Patrolling e della Navigazione in Ambiente Sconosciuto

Il problema del *patrolling* o della circumnavigazione di un oggetto consiste nel guidare un agente lungo il bordo di un ostacolo garantendo il mantenimento di una distanza minima da esso. Al di là del semplice pattugliamento, il *patrolling* trova applicazione nel più generale problema della navigazione in un ambiente sconosciuto, in cui si vuole che un agente sia in grado di raggiungere un punto nello spazio, detto *goal*, utilizzando solo informazioni locali ottenute attraverso sensori atti alla rilevazione di oggetti. Infatti, la soluzione più intuitiva a questo problema vuole che l'agente si diriga verso il *goal* fino all'individuazione di un ostacolo, per poi procedere al *patrolling* dello stesso fino a quando il percorso per l'obiettivo torna ad essere libero da impedimenti. Una

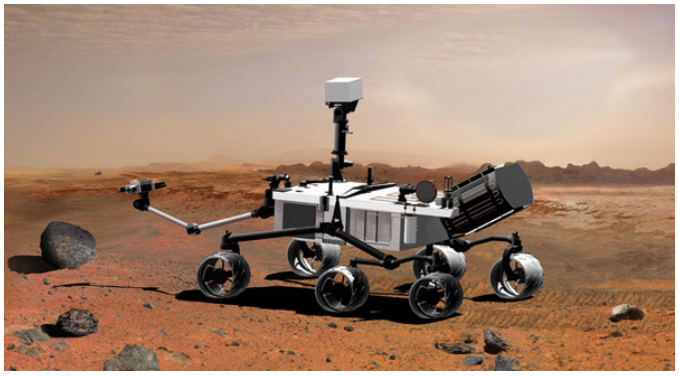


Figura 1.4: Curiosity - NASA

ben nota applicazione di questo problema è l'esplorazione di ambienti, quali pianeti o satelliti, in cui l'agente deve essere in grado di navigare autonomamente nel rispetto delle sole informazioni ottenute dai sen-

sori di bordo. Si cita, ad esempio, la missione *Curiosity* della NASA, Fig. 1.4, che ha come obiettivo l'esplorazione di Marte.

In letteratura sono presenti diverse metodologie per la soluzione di questo problema. In particolare, molto diffusi sono gli algoritmi detti *Bug's Algorithm* che consistono proprio nel superamento degli ostacoli mediante la circumnavigazione degli stessi, come illustrato in Fig. 1.5. Esempi di implementazioni sono presenti in [18] e [34]. Tec-

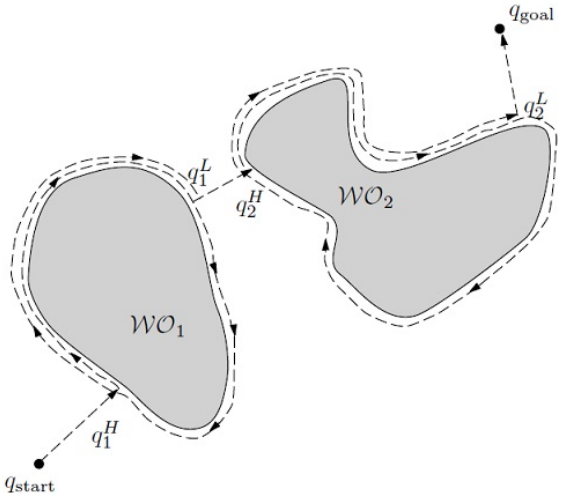


Figura 1.5: Esempio di Navigazione in Ambiente Sconosciuto

niche più elaborate si basano sull'elaborazione di immagini provenienti da telecamere montate a bordo veicolo attraverso le quali è possibile ricavare una rappresentazione dell'ambiente adatta alla comprensione dello stesso, portando quindi a una generazione di traiettorie prive di collisioni.

1.2 Approccio generale proposto

In questa tesi verrà presentata e studiata una tecnica di controllo con l'obiettivo di fornire una soluzione unificante alle problematiche esposte. La struttura proposta è di tipo gerarchico, organizzata in differenti livelli basati su tecniche di controllo predittivo. Tale scelta permette di risolvere separatamente diversi obiettivi:

1. controllo (robusto rispetto a possibili disturbi) del moto dei singoli agenti

2. generazione della traiettoria di riferimento nel rispetto dei vincoli imposti dallo specifico problema.

Come sarà discusso nel prosieguo del lavoro, la soluzione qui presentata permette di trasferire la complessità nel solo livello di generazione della traiettoria, grazie alle proprietà di robustezza che i livelli di controllo sottostanti garantiscono. In aggiunta, il sotto-problema della generazione sarà indipendente dalla dinamica dell'agente rendendo quindi l'algoritmo utilizzato di carattere generale. Questa suddivisione permette inoltre di ottenere notevoli vantaggi dal punto di vista computazionale grazie al fatto che la generazione può essere realizzata mediante problemi di ottimizzazione a un passo che, anche se non lineari, possono essere risolti in tempi brevi grazie alla loro ridotta dimensione.

1.3 Struttura della tesi

La tesi è strutturata nel modo seguente.

Nel Capitolo 2 si presentano l'apparato sperimentale e le tecniche utilizzate per la rilevazione della posizione degli agenti nel piano di lavoro.

Nel Capitolo 3 si analizza lo stato dell'arte inerente al problema del controllo in formazione, validando sperimentalmente alcune tecniche presenti in letteratura.

Nel Capitolo 4, successivamente a una descrizione del controllo predittivo, si presenta la struttura di controllo robusto e distribuita oggetto di questo lavoro.

Nel Capitolo 5 si applica la tecnica introdotta per la soluzione del problema del coordinamento di più agenti in formazione, valutando diverse soluzioni.

Nel Capitolo 6 si illustrano gli aspetti implementativi degli algoritmi presentati con l'obiettivo di rendere più chiara la formulazione matematica alla base e permettere quindi una più immediata implementazione in contesti reali.

Nei Capitoli 7 e 8 si utilizzano le tecniche proposte per la soluzione dei problemi del contenimento e navigazione in ambiente sconosciuto con l'obiettivo di provare i vantaggi derivanti da tali tecniche.

Nel Capitolo 9 si presentano i risultati sperimentali ottenuti mediante le diverse tecniche di controllo presentate. Si eseguono inoltre dei

paragoni con altre tecniche presenti nella letteratura al fine di provarne la validità.

Nel Capitolo 10 si riassumono i risultati ottenuti e si presentano dei possibili sviluppi futuri.

Infine, nell'Appendice sono riportati e spiegati gli algoritmi implementati per la verifica sperimentale delle tecniche oggetto di questa tesi.

Capitolo 2

Apparato Sperimentale e Sistema Controllato

2.1 Robot

Gli agenti utilizzati nella presente tesi sono dei robot, detti "ad uniciclo", largamente diffusi nella letteratura come sistemi di riferimento per i problemi di controllo di agenti multipli, grazie alla semplicità del modello unita alla non linearità che lo caratterizza e lo rende un ottimo punto di riferimento per lo studio e l'analisi delle tecniche di controllo proposte. Si rimanda a [32], [12], e [8] per esempi sull'utilizzo di tale modello. Nelle prove sperimentali si sono utilizzati dei robot *e-puck* [15], sviluppati presso l'*École Polytechnique Fédérale de Lausanne*, grazie alla loro flessibilità e facile gestione.
Ciascun robot è equipaggiato con un modulo *bluetooth* che permette sia la programmazione che lo scambio di dati con altri dispositivi. L'attuazione e movimentazione avvengono mediante due motori passo-passo, che consentono un controllo indipendente delle velocità di ogni singola ruota. Il robot dispone inoltre di diversi sensori, quali una telecamera VGA e sensori di prossimità posti sul perimetro.

La struttura del robot lo rende adatto per essere modellato attraverso il modello uniciclo caratterizzato da un vincolo non olonomo che limita la velocità del robot ad essere ortogonale all'asse che collega le ruote, come rappresentato in Fig. 2.2.

Un modello generalmente utilizzato in letteratura per questa tipologia

Figura 2.1: E-Puck

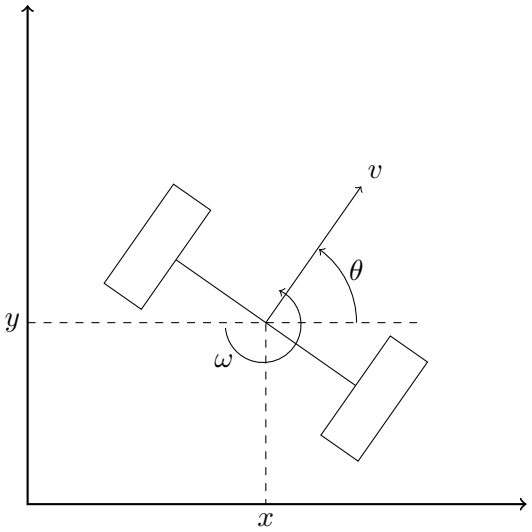


Figura 2.2: Modello uniciclo

di robot è il seguente:

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega \end{cases} \tag{2.1}$$

dove il vincolo anolonomo corrisponde a:

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{2.2}$$

Tale modello presenta alcune approssimazioni: le variabili di ingresso considerate sono la velocità lineare v e la velocità angolare ω o, equivalentemente, le velocità angolari di ciascuna ruota, quando in realtà,

i veri ingressi sono le coppie applicate alle ruote dai motori. In questo modo si trascura quindi la dinamica delle velocità e l'inerzia del robot durante il movimento. Inoltre si è ignorato l'effetto del punto di appoggio sulla dinamica.

2.2 Setup Sperimentale

L'apparato sperimentale utilizzato in questa tesi, consiste in tre robot liberi di muoversi su un piano di lavoro e osservati da una telecamera a colori utilizzata per determinarne posizione e orientamento. Lo schema utilizzato è rappresentato in Fig. 2.3. In particolare si evidenziano le seguenti parti:

Comunicazione: L'azione di controllo viene inviata tramite *bluetooth* ai robot, i quali la applicano ai motori passo-passo in modo da ottenere le velocità desiderate.

Rilevamento posizione: Il rilevamento della posizione dei robot avviene mediante le immagini acquisite dalla telecamera montata sopra il piano di lavoro. Sopra ciascun robot è montato un triangolo colorato: i diversi colori permettono di segmentare l'immagine isolando i diversi agenti, mentre la forma triangolare è stata scelta per permettere la rilevazione dell'orientamento θ. Si rimanda alla Sezione 2.3 per una spiegazione più dettagliata dell'algoritmo utilizzato.

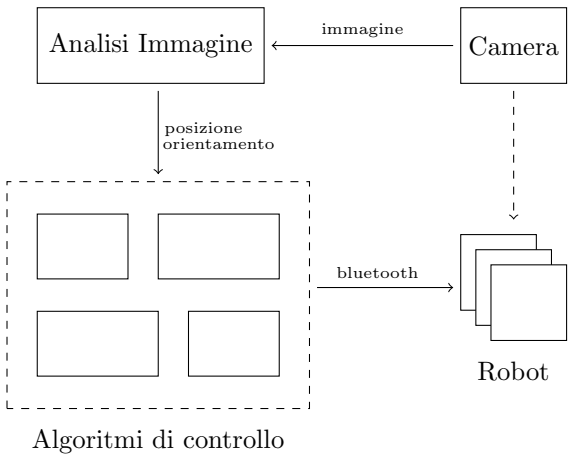


Figura 2.3: Setup sperimentale

Gli algoritmi relativi al processo di elaborazione dell'immagine e al controllo saranno implementati in un unico computer, il quale comunica con gli agenti attraverso un canale *bluetooth*.

2.3 Analisi dell'immagine

Per l'individuazione della posizione degli agenti mobili, è stato sviluppato un algoritmo, il cui scopo è quello di discriminare i diversi agenti e trovare le loro posizioni e orientamento. Si ricorda, come accennato, che i robot sono equipaggiati con un triangolo colorato.

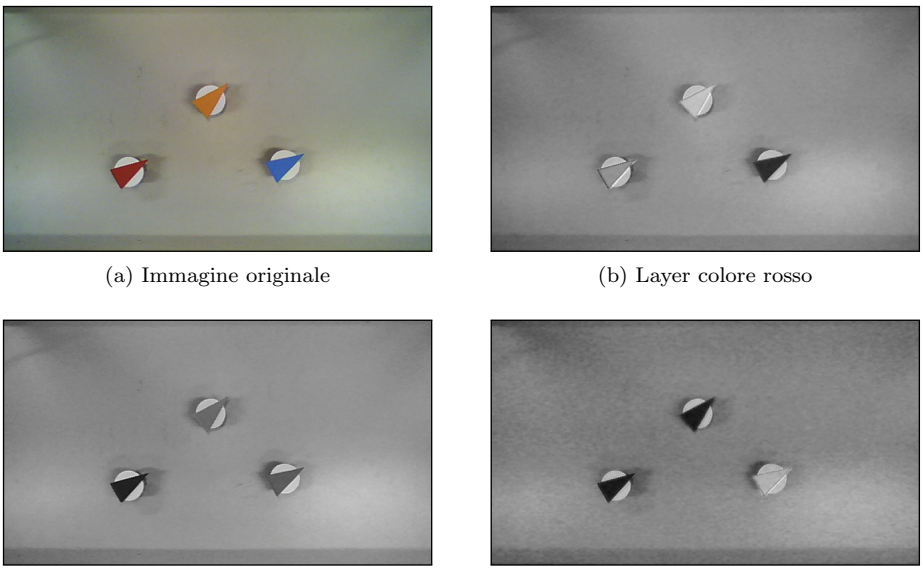
2.3.1 Segmentazione immagine

I colori scelti per i triangoli sono il blu, il rosso e l'arancione. La scelta dei primi due è dovuta alla natura dello spazio dei colori RGB, che permette di separare i livelli di rosso, blu e verde in modo elementare, senza alcuna operazione. L'arancione è stato preferito all'intuitiva scelta del verde, successivamente a prove sperimentali che hanno evidenziato una bassa qualità nel risultato ottenuto. La segmentazione dell'arancione è comunque semplice poichè è l'unico dei tre ad avere elevate componenti sia di verde che di rosso. Nella Fig. 2.4 è presentata l'immagine del piano di lavoro acquisita dalla telecamera unita alla sua scomposizione nel tre diversi livelli di colore. Oltre all'ovvia dominanza dei rispettivi colori, si noti come il rosso non abbia componenti nel livello verde. A partire da queste osservazioni è quindi possibile separare facilmente i triangoli ottenendo tre differenti immagini binarie, ciascuna contenente informazioni riguardanti un solo triangolo. Il risultato è riportato in Fig. 2.5

2.3.2 Posizione e orientamento

L'idea utilizzata per sviluppare un algoritmo che permetta di determinare posizione e orientamento di un triangolo isoscele consiste nell'individuarne i tre vertici da cui è facilmente possibile ottenere le informazioni cercate. L'algoritmo consiste nei seguenti passi:

Determinazione dei vertici attraverso un opportuno algoritmo di *Corner Detection* che fornisce le coordinate corrispondenti. Il risultato di quest'operazione è riportato in Fig. 2.6, in cui, per



(a) Immagine originale

(b) Layer colore rosso

(c) Layer colore verde

(d) Layer colore blu

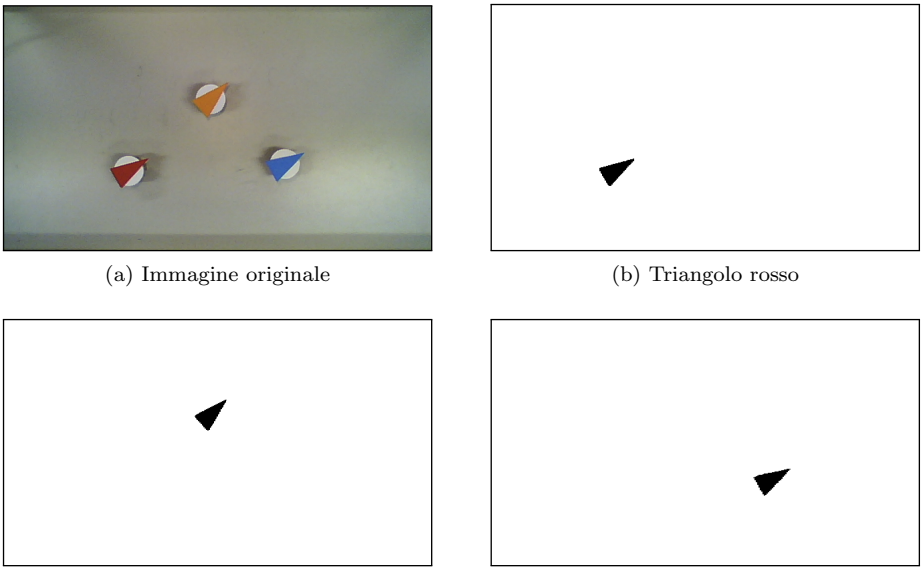
Figura 2.4: Immagine del piano di lavoro e relativi livelli di colore rosso, verde e blu

comodità di rappresentazione, si sono uniti i tre triangoli nella stessa immagine.

Inviduazione del vertice frontale attraverso il calcolo delle distanze fra i tre vertici rilevati. Poichè il triangolo utilizzato è isoscele, il vertice frontale è quello la cui distanza dagli altri due è maggiore.

Calcolo posizione ed orientamento attraverso semplici calcoli geometrici. Il risultato di queste due ultime operazioni è illustrato in Fig. 2.7, in cui si è collegato il punto centrale con il vertice individuato come frontale.

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 alcuna distorsione della videocamera. Le distorsioni sono però presenti specialmente ai bordi del campo visivo, zone in cui ci si aspetta quindi un maggiore errore. Una migliore implementazione può risolvere il problema mediante l'utilizzo di



(a) Immagine originale

(b) Triangolo rosso

(c) Triangolo arancione

(d) Triangolo blu

Figura 2.5: Segmentazione triangoli di diverso colore

strumenti di calibrazione dell'immagine che permettono, appunto, di rimuovere tali distorsioni intrinseche della videocamera.

2.4 Linearizzazione del modello dell'uniciclo tramite Feedback Linearization

Gli algoritmi di controllo presentati nei capitoli successivi richiedono di risolvere un problema di ottimizzazione ad ogni istante di campionamento. Per poter realizzare un'efficiente implementazione *on-line* di tali metodi, è necessario che gli ingredienti del sistema di controllo siano semplici. In questo contesto, ciò significa utilizzare modelli lineari e, relativamente ai problemi di ottimizzazione discussi nel prosieguo, cifre di costo quadratiche unite a eventuali vincoli definiti su insiemi politopici.
Il modello uniciclo (2.1) non è lineare e le forti non linearità presenti non possono essere trascurate, in quanto determinanti nella dinamica dell'agente. Come descritto in [11], si è scelto di applicare un anello di controllo linearizzante che permette di ottenere un sistema in anello chiuso lineare, basato sulla tecnica *Feedback Linearization*.

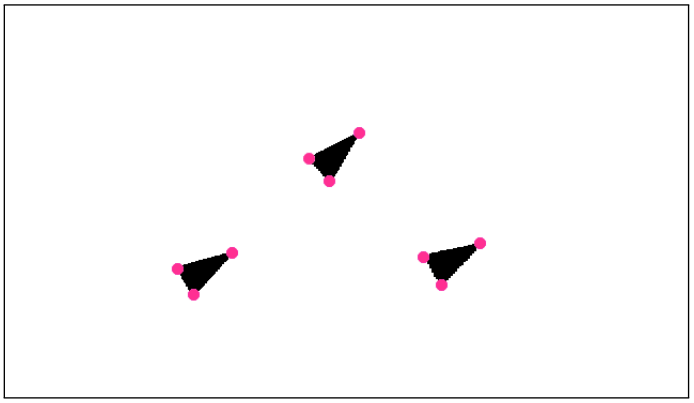


Figura 2.6: Determinazione vertici

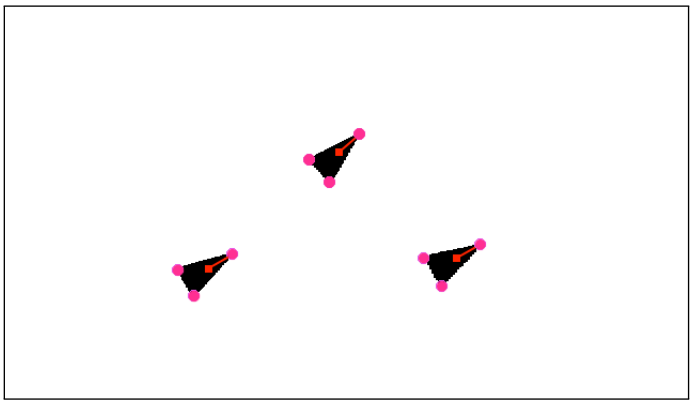


Figura 2.7: Risultato segmentazione e calcolo posizione

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 ha:

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

se $v \neq 0$, la trasformazione è invertibile e si ottiene il seguente controllore linearizzante dinamico del primo ordine:

$$\begin{cases} \dot{v} = a_x\cos\theta + a_y\sin\theta \\ \omega = \dfrac{-a_x\sin\theta + a_y\cos\theta}{v} \end{cases} \tag{2.3}$$

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

$$\begin{cases} \ddot{x} = a_x \\ \ddot{y} = a_y \end{cases} \tag{2.4}$$

che, come voluto, è lineare. Lo schema di controllo è presentato in Fig. 2.8.

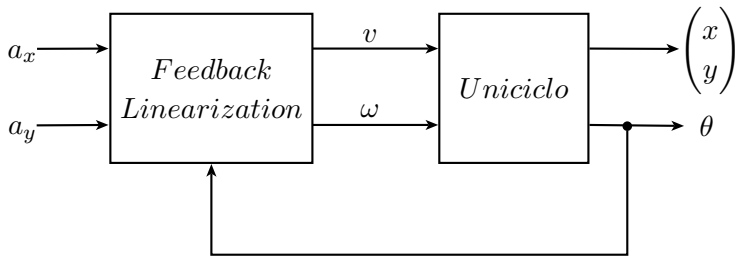


Figura 2.8: Schema controllo uniciclo con Feedback Linearization

Una semplice implementazione discreta dell'anello linearizzante si ottiene mediante il metodo di discretizzazione di Eulero in avanti, risultando nella seguente regola di aggiornamento del controllore:

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

in cui τ è il tempo di campionamento e v_p è l'incremento di velocità. La scelta della frequenza di campionamento per questo anello deve essere fatta ricordando che esso deve operare a velocità molto maggiori di quelle di eventuali anelli di controllo esterni. Una scelta empirica richiede di operare ad una frequenza di almeno un ordine di grandezza superiore.

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

$$\begin{cases} \begin{pmatrix} \dot{\xi}_1 \\ \dot{\xi}_2 \\ \dot{\xi}_3 \\ \dot{\xi}_4 \end{pmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_4 \end{pmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} a_x \\ a_y \end{pmatrix} \\ \begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_4 \end{pmatrix} \end{cases} \tag{2.5}$$

Le tecniche di controllo discusse nei successivi capitoli sono basate su modelli a tempo discreto e pertanto tale modello deve essere discretizzato. Facendo ricorso alla discretizzazione esatta, si ottiene il seguente sistema a tempo discreto:

$$\begin{cases} \begin{pmatrix} \dot{\xi}_1(k+1) \\ \dot{\xi}_2(k+1) \\ \dot{\xi}_3(k+1) \\ \dot{\xi}_4(k+1) \end{pmatrix} = \left[\begin{array}{cc:cc} 1 & \tau & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hdashline 0 & 0 & 1 & \tau \\ 0 & 0 & 0 & 1 \end{array}\right] \begin{pmatrix} \xi_1(k) \\ \xi_2(k) \\ \xi_3(k) \\ \xi_4(k) \end{pmatrix} + \left[\begin{array}{c:c} \tau^2/2 & 0 \\ \tau & 0 \\ \hdashline 0 & \tau^2/2 \\ 0 & \tau \end{array}\right] \begin{pmatrix} a_x(k) \\ a_y(k) \end{pmatrix} \\ \begin{pmatrix} x(k) \\ y(k) \end{pmatrix} = \left[\begin{array}{cc:cc} 1 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 1 & 0 \end{array}\right] \begin{pmatrix} \xi_1(k) \\ \xi_2(k) \\ \xi_3(k) \\ \xi_4(k) \end{pmatrix} \end{cases}$$

in cui τ è il tempo di campionamento. Si noti, come evidenziato, che la dinamica lungo le due direzioni x e y è disaccoppiata, permettendo di risolvere separatamente il problema di controllo. In forma più compatta si può scrivere:

$$\begin{cases} \xi(k+1) = A\xi(k) + Bu(k) \\ z(k) = C\xi(k) \end{cases} \tag{2.6}$$

Capitolo 3

Stato dell'arte

3.1 Approcci al controllo di formazione

Nella letteratura sono stati sviluppati principalmente tre approcci per il controllo di più agenti in formazione:

- Leader-Follower
- Virtual Structure
- Behavior-Based.

Nell'approccio *Leader-Follower*, la formazione può essere rappresentata con un grafo orientato in cui ogni nodo definisce un agente e ogni arco una relazione *leader-follower* fra due agenti (Figura 3.1). Ciascun *follower* ha quindi il compito di seguire il proprio gruppo di *leader* mantenendo una determinata configurazione, generalmente definita da distanza e angolo relativi.

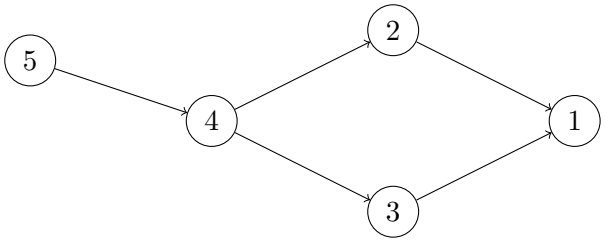


Figura 3.1: Esempio di grafo per la definizione di agenti e di relazioni leader-follower

Il vantaggio di questo approccio è la riduzione della complessità del problema in quanto i controllori dei singoli agenti sono disposti in modo gerarchico: il controllo di ogni *follower* si riduce infatti ad un più

semplice problema di inseguimento di un riferimento. Un punto debole è la forte dipendenza dall'abilità del *leader* di raggiungere l'obiettivo e dall'affidabilità di tutti gli agenti. Infatti, un eventuale guasto di un agente porterebbe al conseguente malfunzionamento di tutti gli agenti dei i quali è *leader*. In aggiunta, l'assenza di un esplicito *feedback* dai *follower* al *leader* può implicare una perdita della formazione nel caso di guasto di un *follower* in quanto il moto del *leader*, non essendo lo stesso a conoscenza dello stato dei suoi *follower*, procederebbe inalterato.

L'approccio *Virtual-Structure* si basa sull'idea di corpo rigido in cui tutti i punti del corpo si muovono mantenendo una ben definita relazione geometrica. Analogamente, la struttura è definita mediante relazioni tra gli agenti che un opportuno sistema di controllo dovrà essere in grado di soddisfare.

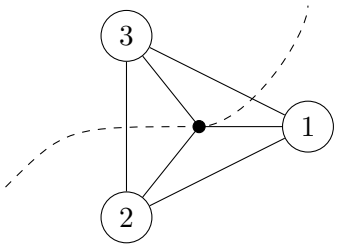


Figura 3.2: Esempio di struttura virtuale

Le traiettorie desiderate non sono assegnate ad ogni singolo robot, ma all'intera struttura. Ciò assicura un'ottima predizione del comportamento della formazione, ma allo stesso tempo richiede un'elevata banda del canale di comunicazione fra i vari agenti coinvolti poichè la posizione di ogni agente deve essere nota a tutti. Questo approccio è, infatti, centralizzato: l'azione di controllo di ogni agente dipende dallo stato dell'intero sistema. È quindi presente un *feedback* implicito fra i vari agenti che, con il fine di mantenere la struttura, rendono questo metodo più robusto rispetto a malfunzionamenti. Inoltre non è richiesta la definizione dei *leader*.

Sistemi controllati attraverso approcci *Behavior-Based* integrano diversi comportamenti atti a raggiungere contemporaneamente obiettivi indipendenti e con priorità differenti. Nel caso del controllo di robot in formazione, gli obiettivi tipici sono:

- Evitare ostacoli statici
- Evitare collisioni con gli altri robot
- Raggiungere la posizione obiettivo
- Mantenere la formazione

in cui l'implementazione specifica di ogni comportamento è libera.
Questa soluzione di controllo presenta evidenti vantaggi dal punto di vista delle abilità degli agenti, ma la formulazione matematica, a causa della sua complessità, rende difficile l'analisi di convergenza della formazione. In altre parole, a causa dei vari comportamenti introdotti nel sistema di controllo, fra loro spesso contrastanti, risulta difficile garantire che i robot si portino nella configurazione voluta.
A causa della sua generalità e complessità, questo approccio non verrà ulteriormente approfondito e si rimanda, ad esempio, a [2].

3.2 Approccio Leader-Follower

L'approccio *Leader-Follower* è molto diffuso in letteratura grazie alla proprietà di ridurre il problema di controllo di formazione ad un problema di inseguimento di più semplice soluzione. In Fig. 3.3 è rappresentato uno schema concettuale di controllo di un *follower* basato su questo approccio. Si noti come l'azione di controllo di tale *follower* dipenda esclusivamente dalla posizione del *leader* e come il movimento di quest'ultimo sia totalmente indipendente dagli altri agenti. Come detto in precedenza, infatti, non è presente alcun *feedback* dai *follower* verso i *leader*.

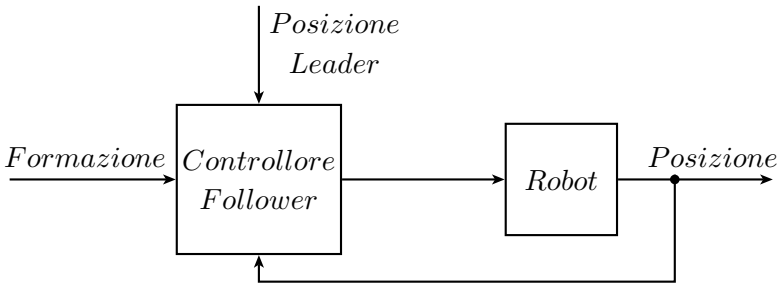


Figura 3.3: Schema di controllo concettuale Leader-Follower

Si vuole ora analizzare, fra i vari sistemi di controllo proposti in letteratura, quello descritto in [8] che definisce due tipologie di configurazione *leader-follower* e due relative soluzioni di controllo.

3.2.1 Separation Bearing Control $SB_{ij}C$

Si considerino due robot R_i e R_j. Si vuole che il *follower* R_j segua il *leader* R_i mantenendo una distanza l_{ij}^d ed un angolo ψ_{ij}^d, come indicato in Fig.3.4.

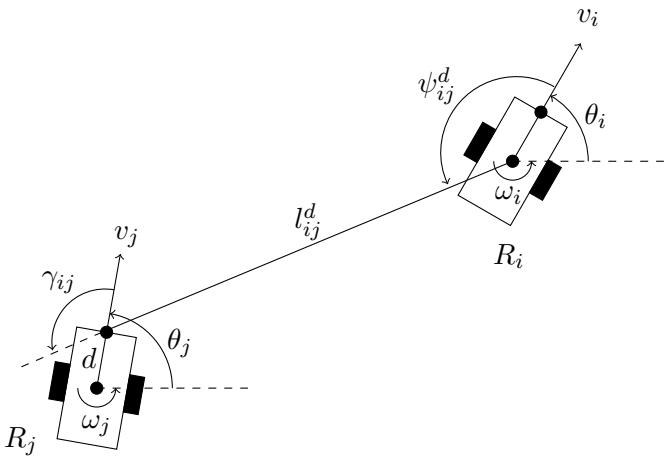


Figura 3.4: Problema $SB_{ij}C$

I robot sono modellati come unicicli, perciò il modello di R_i è:

$$\begin{cases} \dot{x_i} = v_i \cos\theta_i \\ \dot{y_i} = v_i \sin\theta_i \\ \dot{\theta}_i = \omega_i \end{cases} \tag{3.1}$$

Posizione e orientamento di R_j possono essere invece espressi in modo relativo come segue:

$$\begin{cases} \dot{l}_{ij} = v_j \cos\gamma_{ij} - v_i \cos\psi_{ij} + \omega_j d \sin\gamma_{ij} \\ \dot{\psi}_{ij} = \dfrac{1}{l_{ij}}[v_i \sin\psi_{ij} - v_j \sin\gamma_{ij} + d\omega_j \cos\gamma_{ij} - l_{ij}\omega_i] \\ \dot{\theta}_j = \omega_j \end{cases} \tag{3.2}$$

in cui per i simboli si fa riferimento alla Fig.3.4.
Ignorando la dinamica dell'angolo θ_j, il sistema dinamico non lineare

(3.2) può essere riscritto nella forma:

$$\underbrace{\begin{pmatrix} \dot{l}_{ij} \\ \dot{\psi}_{ij} \end{pmatrix}}_{\dot{x}} = \underbrace{\begin{pmatrix} -v_i \cos\psi_{ij} \\ \dfrac{v_i}{l_{ij}} \sin\psi_{ij} - \omega_i \end{pmatrix}}_{f(x)} + \underbrace{\begin{bmatrix} \cos\gamma_{ij} & d\sin\gamma_{ij} \\ \dfrac{-\sin\gamma_{ij}}{l_{ij}} & \dfrac{d\cos\gamma_{ij}}{l_{ij}} \end{bmatrix}}_{g(x)} \underbrace{\begin{pmatrix} v_j \\ \omega_j \end{pmatrix}}_{u} \tag{3.3}$$

che rientra nella generale famiglia di sistemi:

$$\dot{x} = f(x) + g(x)u \tag{3.4}$$

Supponendo $g(x)$ invertibile, è possibile costruire una legge di controllo che renda il sistema in anello chiuso lineare, attraverso *feedback linearization*. A tal proposito si consideri la seguente legge:

$$u = g(x)^{-1}(Kx + v - f(x)) \tag{3.5}$$

in cui K è un parametro del controllore e v un ingresso aggiuntivo utilizzabile, ad esempio, per chiudere un secondo anello esterno. Sostituendo (3.5) in (3.4) si ottiene il seguente sistema in anello chiuso

$$\dot{x} = Kx + v \tag{3.6}$$

che, come voluto, risulta lineare.
Quando l'obiettivo di controllo è l'inseguimento di un riferimento x^d, è possibile scegliere v come:

$$v = -Kx^d \tag{3.7}$$

ottenendo un sistema in anello chiuso:

$$\dot{x} = K(x - x^d). \tag{3.8}$$

il cui punto di equilibrio è $x = x^d$. Attraverso un'opportuna scelta del parametro K che assicuri l'asintotica stabilità, si ottiene una convergenza esponenziale di x a x^d.
Di conseguenza, poichè nel problema considerato $g(x)$ risulta sempre invertibile, infatti:

$$\det(g(x)) = \frac{d}{l_{ij}}\cos^2\gamma_{ij} + \frac{d}{l_{ij}}\sin^2\gamma_{ij} = \frac{d}{l_{ij}} \neq 0,$$

sostituendo (3.3) in (3.5) si ottiene la legge di controllo:

$$\begin{cases} v_j = s_{ij}\cos\gamma_{ij} - l_{ij}(b_{ij}+\omega_i)\sin\gamma_{ij} + v_i\cos(\theta_i-\theta_j) \\ w_j = \dfrac{1}{d}[s_{ij}\sin\gamma_{ij} + l_{ij}(b_{ij}+\omega_i)\cos\gamma_{ij} + v_i\sin(\theta_i-\theta_j)] \end{cases} \tag{3.9}$$

in cui si è posto $K = diag(-k_1, -k_2)$, $v = (l_{ij}^d, \psi_{ij}^d)^\top$, $s_{ij} = k_1(l_{ij}^d - l_{ij})$ e $b_{ij} = k_2(\psi_{ij}^d - \psi_{ij})$.
Il sistema in anello chiuso lineare risulta:

$$\begin{cases} \dot{l}_{ij} = k_1(l_{ij}^d - l_{ij}) \\ \dot{\psi}_{ij} = k_2(\psi_{ij}^d - \psi_{ij}) \end{cases} \tag{3.10}$$

scegliendo $k_1, k_2 > 0$, si garantisce una convergenza esponenziale di l_{ij} e ψ_{ij} come discusso nel caso generale.

3.2.2 Separation Separation Control $S_{ik}S_{jk}C$

Si considerino ora tre robot R_i, R_j e R_k. Si vuole che il *follower* R_k segua i *leader* R_i e R_j rispettivamente con distanze pari a l_{ik}^d e l_{jk}^d, come indicato in Fig. 3.5. Si assume che $l_{ij}(t) < l_{ik}^d + l_{jk}^d \, \forall t$, cioè che la distanza fra R_i e R_j sia sempre inferiore alla somma delle distanze di riferimento fra ciascun *leader* e il *follower*. In caso tale condizione non sia soddisfatta, la formazione non è realizzabile.
Analogamente al problema $SB_{ij}C$, la dinamica dei *leader* è espressa da (3.1), mentre il modello del robot *follower* R_k può essere ancora una volta descritto in funzione della posizione relativa rispetto ai *leader*:

$$\begin{cases} \dot{l}_{ik} = v_k\cos\gamma_{ik} - v_i\cos\psi_{ik} + d\omega_k\sin\gamma_{ik} \\ \dot{l}_{jk} = v_k\cos\gamma_{jk} - v_j\cos\psi_{jk} + d\omega_k\sin\gamma_{jk} \\ \dot{\theta}_k = \omega_k \end{cases} \tag{3.11}$$

in cui $\gamma_{ik} = \theta_i + \psi_{ik} - \theta_k$ e $\gamma_{jk} = \theta_j + \psi_{jk} - \theta_k$ come in Fig. 3.5.
Il sistema (3.11), rientra nella generale famiglia di sistemi (3.4):

$$\underbrace{\begin{pmatrix} \dot{l}_{ik} \\ \dot{l}_{jk} \end{pmatrix}}_{\dot{x}} = \underbrace{\begin{pmatrix} -v_i\cos\psi_{ik} \\ -v_j\cos\psi_{jk} \end{pmatrix}}_{f(x)} + \underbrace{\begin{bmatrix} \cos\gamma_{ik} & d\sin\gamma_{ik} \\ \cos\gamma_{jk} & d\sin\gamma_{jk} \end{bmatrix}}_{g(x)} \underbrace{\begin{pmatrix} v_k \\ \omega_k \end{pmatrix}}_{u} \tag{3.12}$$

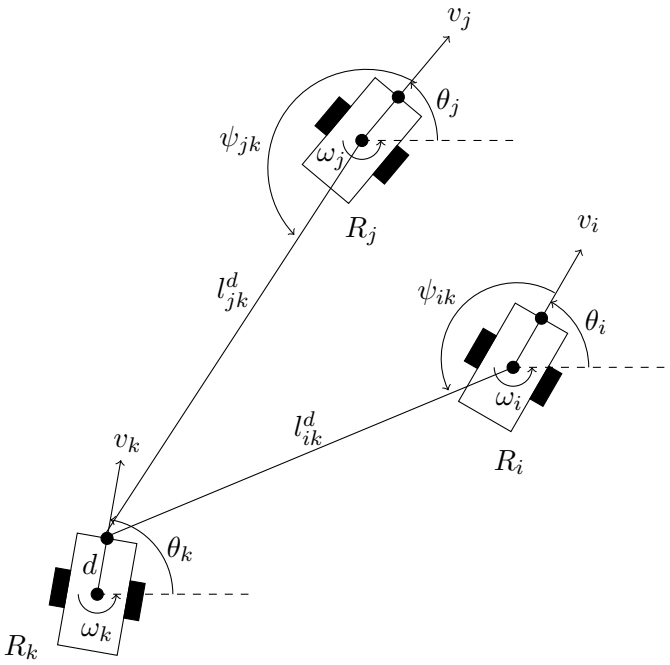


Figura 3.5: Problema $S_{ik}S_{jk}C$

a cui è possibile applicare la legge di controllo (3.5) definita solo quando $g(x)$ risulta invertibile:

$$\begin{aligned} \det g(x) &= d(\cos\gamma_{ik}\sin\gamma_{jk} - \sin\gamma_{ik}\cos\gamma_{jk}) \\ &= d\sin(\gamma_{ik} - \gamma_{jk}) \\ &\neq 0 \iff \gamma_{ik} - \gamma_{jk} \neq k\pi, k = 0, 1, 2, \ldots \end{aligned}$$

cioè quando i vettori $\vec{l_{jk}}$ e $\vec{l_{ik}}$ non sono linearmente dipendenti o, analogamente, quando i tre robot non sono allineati. La configurazione critica in cui i robot giacciono sullo stessa retta sarà evitata utilizzando un'opportuna tecnica di commutazione, come discusso nella sezione successiva.
Sostituendo (3.12) in (3.5) si ottiene la legge di controllo:

$$\begin{cases} v_k = \dfrac{s_{ik}\sin\gamma_{jk} - s_{jk}\sin\gamma_{ik} + v_i\cos\psi_{ik}\sin\gamma_{jk} - v_j\cos\psi_{jk}\sin\gamma_{ik}}{\sin(\gamma_{jk} - \gamma_{ik})} \\ w_k = \dfrac{-s_{ik}\cos\gamma_{jk} + s_{jk}\cos\gamma_{ik} - v_i\cos\psi_{ik}\cos\gamma_{jk} + v_j\cos\psi_{jk}\cos\gamma_{ik}}{d\sin(\gamma_{jk} - \gamma_{ik})} \end{cases} \tag{3.13}$$

in cui si è posto $K = diag(-h_1, -h_2)$ e $v = (l^d_{ik}, l^d_{jk})^\top$, $s_{ik} = h_1(l^d_{ik} - l_{ik})$ e $s_{jk} = h_2(l^d_{jk} - l_{jk})$.

Il sistema lineare in anello chiuso risulta:

$$\begin{cases} \dot{l}_{ik} = h_1(l_{ik}^d - l_{ik}) \\ \dot{l}_{jk} = h_2(l_{jk}^d - l_{jk}) \end{cases} \tag{3.14}$$

che garantisce quindi una convergenza esponenziale di l_{ik} e l_{jk}, supponendo $h_1, h_2 > 0$.

3.2.3 Strategia di commutazione per il controllo di tre robot

Si consideri il problema di mantenere in formazione tre robot R_1, R_2 e R_3. L'algoritmo di controllo $S_{ik}S_{jk}C$ non permette di ottenere sempre la formazione voluta indipendentemente dalle condizioni iniziali a causa delle seguenti problematiche:

1. la legge di controllo non è definita se i robot sono allineati
2. se la distanza fra i *leader* è maggiore della somma delle distanze da ciascun *leader* al *follower*, la formazione non è realizzabile e quindi l'algoritmo non è applicabile
3. l'equilibrio in cui si portano i robot potrebbe non essere quello desiderato. Infatti, potrebbe esserci più di un punto alle distanze desiderate da entrambi i *leader*.

La soluzione proposta in [8] consiste nell'implementare un meccanismo di commutazione fra le tecniche $SB_{ij}C$ e $S_{ik}S_{jk}C$ a seconda della posizione del *follower*. Si definiscono due regioni circolari di raggi r_1 e r_2, con $r_1 < r_2$, centrate nei *leader* e si utilizza il seguente criterio, rappresentato graficamente in Fig.3.6:

$$\begin{aligned} \textbf{if } (l_{13} < l_{23}) \ \&\ (l_{23} > r_1) \ \&\ (l_{13} < r_2) \textbf{ then } SB_{13}C \\ \textbf{if } (l_{13} > l_{23}) \ \&\ (l_{13} > r_1) \ \&\ (l_{23} < r_2) \textbf{ then } SB_{23}C \\ \textbf{if } (l_{13} < r_1) \ \&\ (l_{23} < r_1) \textbf{ then } S_{13}S_{23}C \\ \textbf{if } (l_{13} > r_2) \ \&\ (l_{23} > r_2) \textbf{ then } \textit{Navigazione Autonoma} \end{aligned}$$

Quando il *follower* è esterno all'area delimitata dalle circonferenze di raggio r_2, detta *regione di commutazione*, l'algoritmo prevede che esso navighi autonomamente. Tale navigazione può, ad esempio, consistere nell'attuare un controllo che diriga il *follower* nella posizione corrente

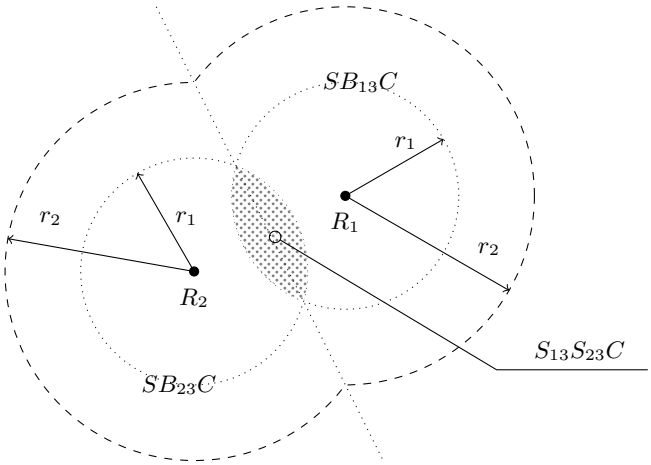


Figura 3.6: Logica di Switching

del *leader*: in tal modo, il *follower* si avvicinerà autonomamente fintantochè non entrerà nella regione di commutazione e quindi uno degli algoritmi sopra esposti verra attivato.

3.2.4 Obstacle Avoidance

Gli algoritmi presentati possono essere sfruttati anche per controllare i robot in modo tale che evitino degli ostacoli noti a priori. L'idea è di considerare un robot fittizio e una regione di attivazione per ogni ostacolo. Quando un robot entra nella regione di attivazione, viene applicata la legge di controllo $S_{ik}S_{jk}C$ che manterrà tale robot ad una distanza voluta dall'ostacolo evitando così la collisione.

3.2.5 Prove Sperimentali

Le leggi di controllo presentate sono state implementate utilizzando l'apparato sperimentale descritto nel Capitolo 2. Tutte le prove sono state validate eseguendo in parallelo simulazioni con il modello del robot, in modo da poter confrontare il risultato teorico con quello sperimentale.

La *prima prova* consiste nell'inseguimento in coda rettilineo da parte di un singolo *follower* di un *leader* il cui moto è imposto esternamente. In particolare, è stato quindi utilizzato l'algoritmo $SB_{ij}C$ con una distanza ed un angolo di riferimento rispettivamente pari a $l_{ij}^d = 8.5cm$ e $\psi_{ij}^d = \pi$, ottenendo il risultato riportato in Fig. 3.7. Il risultato è sod-

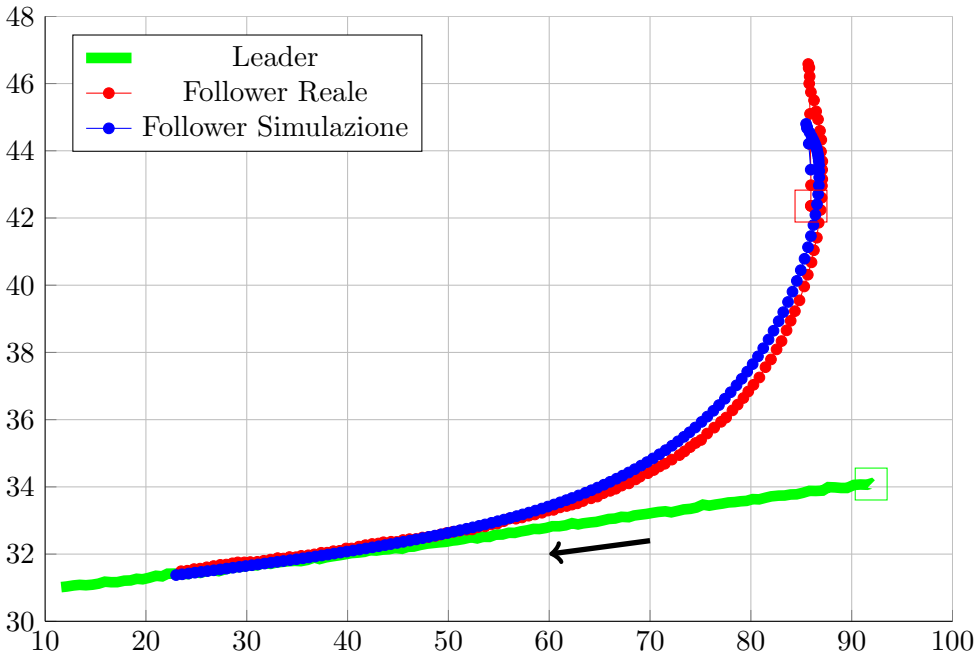


Figura 3.7: Simulazione Inseguimento in coda rettilineo

disfacente, infatti la traiettoria effettuata del robot reale è paragonabile a quella del robot simulato. Inoltre, si ha una convergenza esponenziale a zero della distanza fra il *leader* e il *follower*, come discusso precedentemente.

Nella *seconda prova*, rispetto alla precedente, varia solo la traiettoria seguita dal *leader*, non più rettilinea, bensì circolare. Il risultato, riportato in Fig. 3.8, è ancora soddisfacente, ma si nota un non perfetto inseguimento, anche del robot simulato. Ciò è dovuto a una ridotta banda del sistema di controllo, corrispondente a piccoli valori dei guadagni k_1, k_2 del controllore. Purtroppo, tali valori non possono essere aumentati arbitrariamente poichè si verificherebbero saturazioni delle variabili di controllo. I guadagni utilizzati sono quindi un compromesso fra velocità di risposta e distanza dalla saturazione delle variabili di controllo.

La *terza prova* coinvolge un *leader*, chiamato R_1, e due *follower*, detti R_2 e R_3, in formazione triangolare (base $12cm$, altezza $12cm$). Il *leader* segue una traiettoria rettilinea, R_2 utilizza una legge di controllo $SB_{12}C$, mentre R_3 alterna $SB_{13}C, SB_{23}C$ e $S_{12}S_{23}C$ utilizzando la tecnica di commutazione presentata nella Sezione 3.2.3 con $r_1 = 10cm$

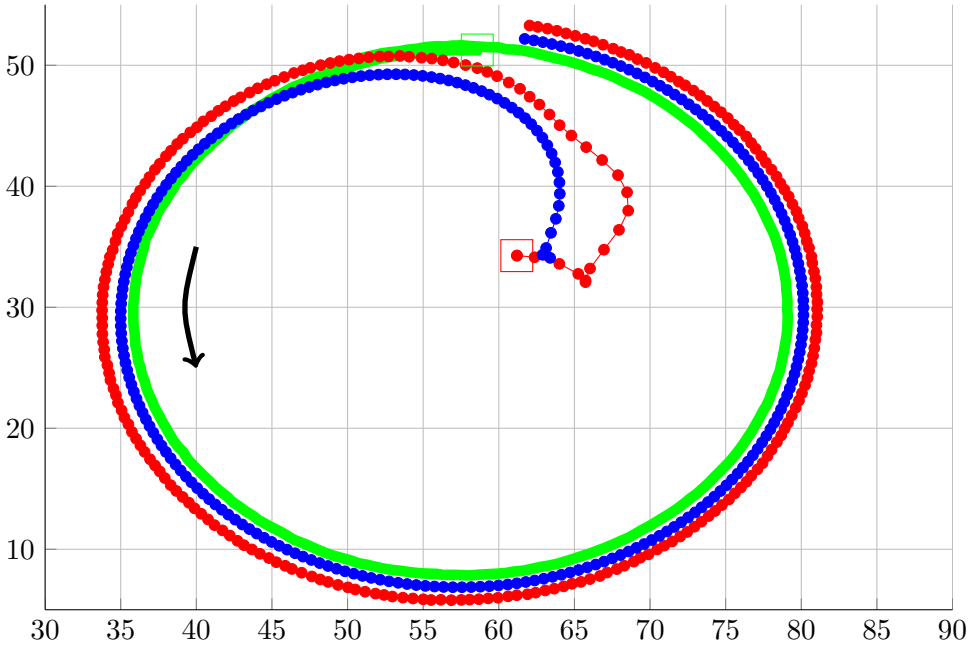
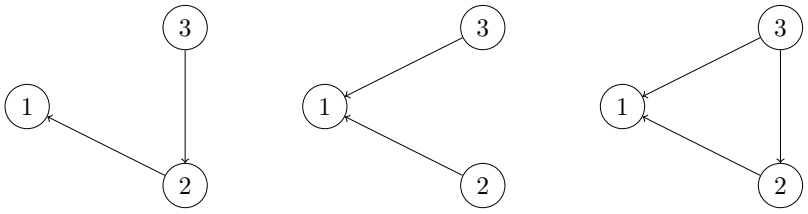


Figura 3.8: Simulazione Inseguimento in coda circolare

e $r_2 = 50cm$. Il raggio r_2 è stato scelto in modo tale che R_3 non entri mai nella regione di navigazione autonoma, eliminando così il problema di dover definire un ulteriore algoritmo per la gestione del robot in tale situazione. A causa dei tre differenti algoritmi utilizzati per il controllo di R_3, si ottengono tre diversi grafi che descrivono le relazioni *leader-follower* nella formazione, mostrati in Fig. 3.9.



(a) $SB_{23}C$ (b) $SB_{13}C$ (c) $S_{13}S_{23}C$

Figura 3.9: Grafici Formazione

I robot partono da una posizione in cui i *follower* sono scambiati rispetto alla formazione finale. Inoltre, R_3 parte orientato nella direzione opposta. Il risultato, presentato in Fig. 3.10, è ancora una volta soddisfacente e i robot raggiungono la formazione voluta. Si nota

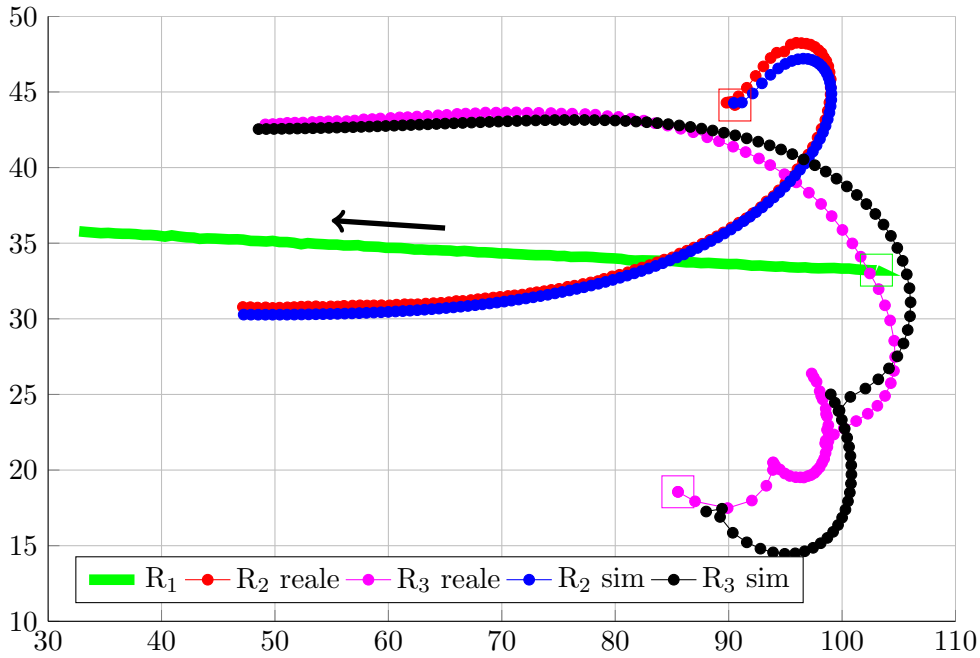


Figura 3.10: Simulazione Formazione Triangolare

una maggiore discrepanza fra modello e robot reale quando quest'ultimo compie movimenti veloci (è il caso di R_3 quando inverte il verso di marcia). Ciò è dovuto alla semplicità del modello considerato, come discusso nel Capitolo 2, e ad eventuali errori di acquisizione della telecamera.

La *quarta prova* è analoga alla precedente, a meno della posizione iniziale dei robot che sono disposti in linea con entrambi i *follower* orientati nel verso corretto. Il risultato, presentato in Fig. 3.11, è ancora buono, infatti la formazione finale viene raggiunta. Si notano però delle oscillazioni nella traiettoria di R_3, sia nel modello simulato che, in particolare, nel robot reale. Questo effetto è dovuto alla discretizzazione: le leggi di controllo sono state studiate a tempo continuo, ma l'effettiva implementazione è avvenuta utilizzando un sistema a tempo discreto. Poichè la legge di controllo viene mantenuta costante fra un campione e il successivo, che corrisponde all'utilizzo di un mantenitore di ordine zero, si ha un ritardo nell'anello e quindi una diminuzione di margine di fase che spiega tali oscillazioni.

La *quinta prova* utilizza il metodo descritto precedentemente per evitare la collisione con un ostacolo. La configurazione iniziale dei

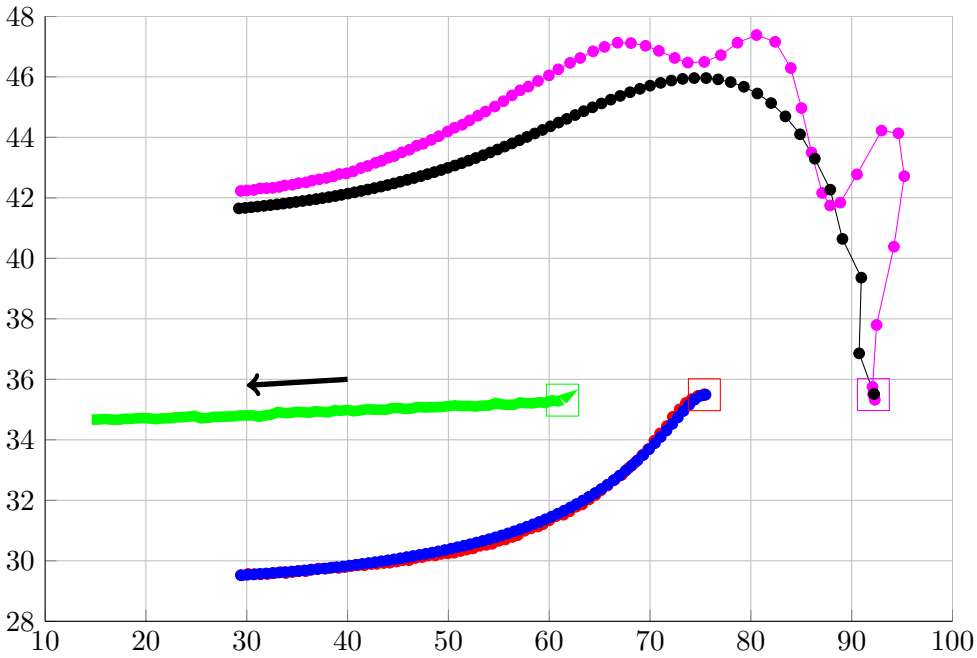


Figura 3.11: Simulazione Formazione Triangolare, partenza in linea

robot e la formazione desiderata coincidono con quanto utilizzato nella terza prova. La Fig. 3.12 evidenzia ancora una volta che le leggi di controllo utilizzate portano a un risultato soddisfacente.

3.2.6 Conclusioni

Gli algoritmi presentati portano ad una semplice e facile soluzione del problema *Leader-Follower*, come evidenziato dalle precedenti prove. Il problema principale riscontrato nell'utilizzo di queste tecniche riguarda la saturazione delle variabili di controllo. Infatti, nella prove effettuate, è stato necessario ridurre i valori dei guadagni k_1, k_2, h_1, h_2 in modo tale da ovviare a questo problema. In particolare si è notato che tale fenomeno dipende dalla distanza fra i robot, come, per altro, è anche chiaramente espresso dalle leggi di controllo (3.13), (3.9). Una soluzione potrebbe quindi essere quella di considerare guadagni tempo varianti, espressi come funzione di tale distanza. Quest'idea non è però di facile trattazione e non verrà ulteriormente studiata. Una seconda problematica, già accennata, riguarda l'implementazione digitale rispetto allo studio a tempo continuo del sistema. Infatti a causa

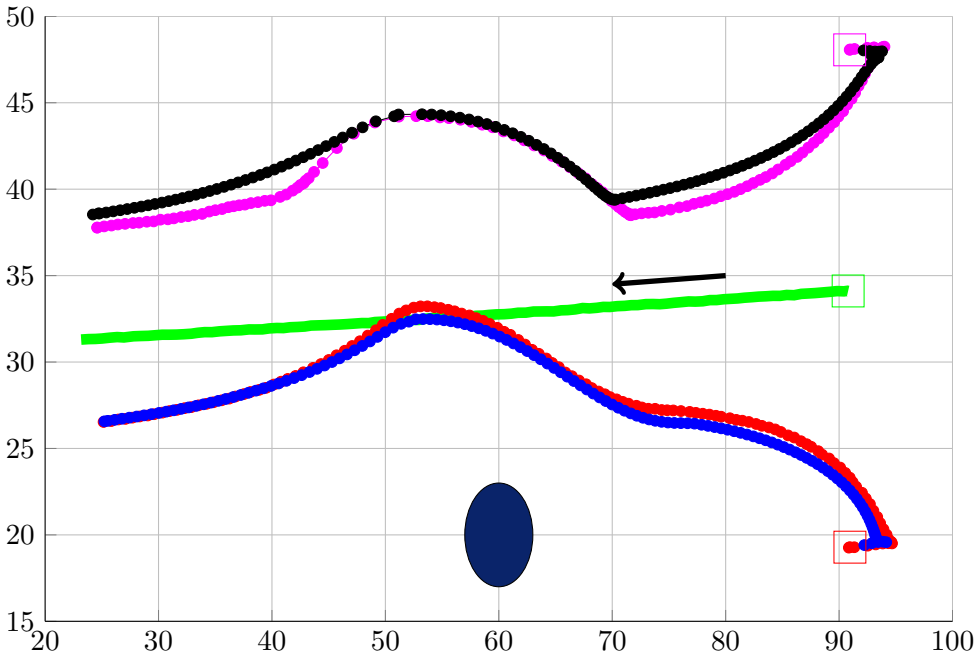


Figura 3.12: Simulazione Formazione Triangolare con ostacolo

di elevati tempi di campionamento, dovuti alla necessità di acquisire ed elaborare un'immagine in ogni intervallo di tempo, si introducono rilevanti ritardi nell'anello che portano ad un degrado delle prestazioni dando luogo a sovraelongazioni oppure a oscillazioni.

3.3 Approccio Virtual Structure

Il controllo di agenti in formazione secondo la tecnica *Virtual Structure* si basa sull'idea che essi siano delle particelle vincolate ad una struttura meccanica. I vincoli meccanici dovuti da tale struttura sono però imposti da un opportuno sistema di controllo: in questo modo gli agenti si comporteranno come se fossero meccanicamente vincolati. Si parla quindi di Struttura Virtuale.
La definizione formale di una struttura virtuale si basa sul concetto di corpo rigido:

> *Un corpo rigido è un insieme di punti materiali sottoposti a vincoli olonomi tali che* $|r_i - r_j| = d_{ij} = cost.$*, dove* r_i *e* r_j *sono le posizioni delle particelle. I punti sono quindi fissi*

rispetto ad un sistema di riferimento solidale con la struttura stessa.

Segue la definizione:

> *Una Struttura Virtuale è una collezione di agenti che mantengono reciprocamente una definita (semi-)rigida relazione geometrica.*

Il problema può quindi essere formulato nel seguente modo:

> *Si considerino n agenti la cui posizione è rappresentata dai vettori* $r_1^W, \ldots, r_n^W$ *dove* W *indica il sistema di riferimento assoluto. Si definisca una struttura virtuale come un insieme di n punti rappresentati dai vettori* $p_1^R, \ldots, p_n^R$ *in cui* R *indica un sistema di riferimento solidale con la struttura stessa. Sia* I_R^W *la matrice di trasformazione che mappa i vettori definiti in* R *in vettori definiti in* W*, i.e.* $p_i^W = I_R^W p_i^R$*. Gli agenti si muovono quindi in formazione perfetta se* $r_i^W = p_i^W, i = 1, \ldots, n$*.*

In letteratura è possibile trovare diverse soluzioni al problema appena illustrato. L'approccio generale consiste nel valutare l'errore di formazione di ciascun agente rispetto alla posizione della struttura e determinare quindi opportune leggi di controllo: lo schema di controllo è centralizzato.
Una soluzione di semplice intuizione è stata proposta in [16] mediante il seguente algoritmo:

1. Allineare la struttura virtuale con le correnti posizioni dei robot
2. Muovere la struttura virtuale di Δx e $\Delta\theta$
3. Calcolare la traiettorie di ogni robot in modo da raggiungere la corrispondente posizione nella struttura
4. Impostare le velocità delle ruote di ogni robot in base alla traiettoria calcolata

5. Ripetere da 1.

Si consideri il primo punto: si tratta di un problema di *fitting* della struttura rispetto alle attuali posizioni dei robot risolvibile attraverso un problema di ottimizzazione. La funzione di costo proposta in [16] consiste nella somma della distanza di ogni robot rispetto alla posizione assegnata nella struttura che dà luogo ad un problema non lineare. Per quanto riguarda la scelta degli incrementi Δx e $\Delta\theta$ della struttura virtuale, essi devono essere tali da permettere ad ogni robot di raggiungere la nuova posizione nel tempo a disposizione. In caso contrario, si avranno degli errori di posizionamento. Infine, i punti 3 e 4 possono essere implementati utilizzando un anello interno di controllo di posizione per ogni singolo robot, il cui set point è ottenuto dal punto 2.
L'algoritmo non verrà qui ulteriormente discusso, in quanto l'approccio più adeguato agli scopi cercati è, per le sue proprietà, il *leader-follower*. Per i dettagli implementativi si rimanda a [16].

L'algoritmo presentato permette di introdurre i principali vantaggi dell'approccio *Virtual Structure*. Il controllo ottenuto è di tipo bidirezionale, infatti si esegue un *fitting* della struttura rispetto alla posizione dei robot (punto 1) seguito dal *fitting* dei robot rispetto alla struttura (punti 3, 4). Perciò il comportamento del sistema non dipende nè solo dai robot, nè solo dalla struttura virtuale. Tale bidirezionalità fornisce una notevole robustezza rispetto a malfunzionamenti di un robot: la formazione assegnata sarà sempre mantenuta, eventualmente a discapito del raggiungimento dell'obiettivo finale. Si noti inoltre come tale controllo possa essere applicato a qualsiasi tipo di struttura geometrica. Infine, come evidenziato in [16], si ottiene un'elevata precisione nel mantenimento della formazione. Le considerazioni appena discusse, sono illustrate nello schema di controllo concettuale in Fig. 3.13, in cui è presente un *feedback* da tutti gli agenti al controllore centralizzato, che evidenzia come le azioni di controllo dipendano dallo stato dell'intero sistema. Si noti infine come nello schema di controllo relativo al problema *Leader-Follower*, riportato in Fig. 3.3, non sia presente alcun anello esterno rispetto al semplice controllo di posizione degli agenti. Ciò è la sorgente delle problematiche di robustezza, precedentemente discusse, intrinseche di tale metodologia.

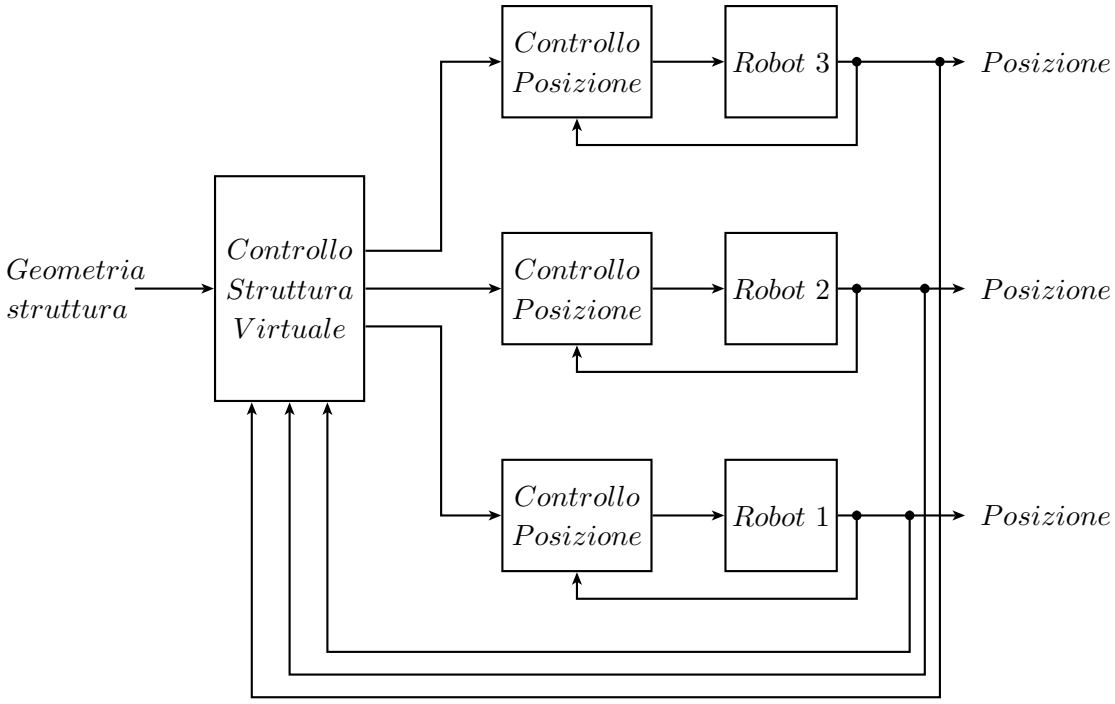


Figura 3.13: Schema di controllo concettuale - Virtual Structure

3.4 Approccio Behavior-Based

Sistemi controllati attraverso approcci *Behavior-Based* integrano diversi comportamenti atti a raggiungere contemporaneamente obiettivi indipendenti e con priorità differenti. Nel caso del controllo di robot in formazione, gli obiettivi tipici sono:

- Evitare ostacoli statici
- Evitare collisioni con gli altri robot
- Raggiungere la posizione obiettivo
- Mantenere la formazione

L'implementazione specifica di ogni comportamento è libera.
Questa soluzione di controllo presenta evidenti vantaggi dal punto di vista delle abilità degli agenti. Come già accennato, la formulazione matematica, a causa della sua complessità, rende difficile l'analisi di convergenza della formazione. In altre parole, a causa dei vari comportamenti introdotti nel sistema di controllo, fra loro spesso contrastanti,

risulta difficile garantire che i robot si portino nella configurazione voluta.
A causa della sua generalità e complessità, questo approccio non verrà ulteriormente approfondito e si rimanda, ad esempio, a [2].

Capitolo 4

Controllo Predittivo di un agente per la regolazione e l'inseguimento di una traiettoria di riferimento

4.1 Introduzione

Il controllo predittivo, o *Model Predictive Control* (MPC), è una tecnica di controllo sviluppata, principalmente in ambito industriale, a partire dagli anni '80, che viene ora largamente utilizzata nella soluzione di generali problemi di controllo di sistemi multivariabili. Il problema di controllo viene formulato come un problema di ottimizzazione in cui è possibile inserire obiettivi diversi, anche contrastanti, oltre a includere vincoli espliciti sia sulle variabili di controllo che sullo stato del sistema permettendo, ad esempio, di tenere esplicitamente conto di saturazioni degli attuatori.
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. Il risultato dell'ottimizzazione è la sequenza ottima di ingressi da applicare al sistema lungo l'orizzonte. MPC è comunemente utilizzato congiuntamente alla tecnica *Receding Horizon* (RH), che consiste nel risolvere nuovamente il problema di ottimizzazione ad ogni instante di campionamento e applicare al sistema solo il primo ingresso della sequenza

ottima di ingressi calcolata, scartando i restanti. Al successivo istante si risolve nuovamente il problema, muovendo di un passo in avanti l'orizzonte di predizione, come illustrato in Fig. 4.1 per un sistema SISO.

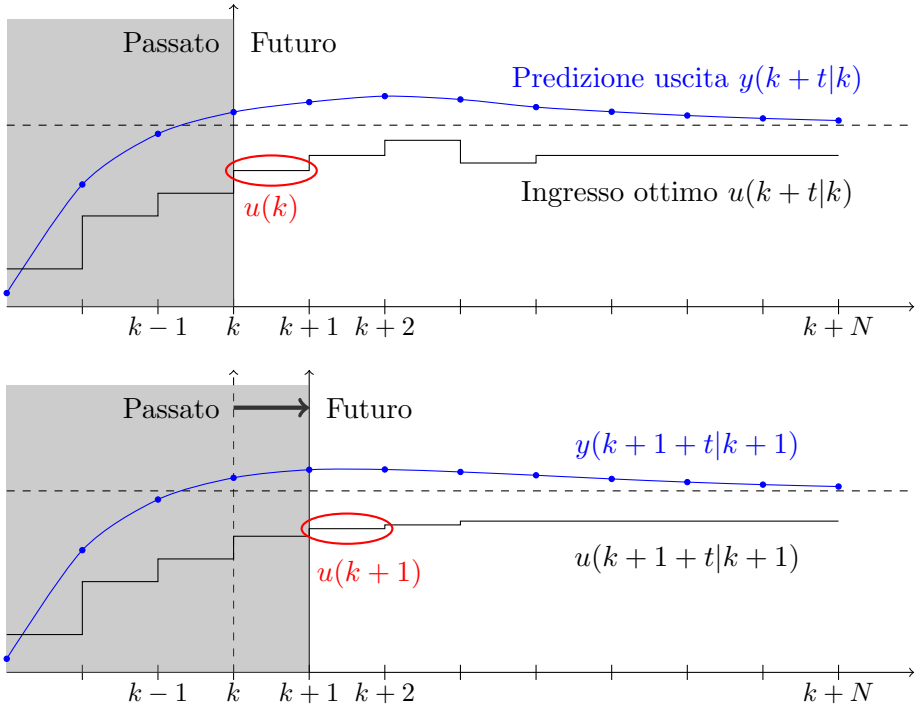


Figura 4.1: MPC con Receding Horizon

4.2 Controllo predittivo di sistemi lineari

Si consideri un generico sistema lineare a tempo discreto:

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) \end{cases} \tag{4.1}$$

in cui $x(k) \in \mathbb{R}^n$, $u(k) \in \mathbb{R}^m$ e $y(k) \in \mathbb{R}^p$ rappresentano rispettivamente lo stato, l'ingresso e l'uscita. All'istante k si vuole trovare la sequenza ottima di ingressi $\mathcal{U}^\circ(k) = [u(k), u(k+1), \ldots, u(k+N-1)]$, per un fissato orizzonte di predizione N, che minimizzi la cifra di merito:

$$V^N(x(k), U(k)) = \sum_{i=0}^{N-1} l(x(k+i), u(k+i)) + V^f(x(k+N)) \tag{4.2}$$

in cui $l(\cdot,\cdot)$ è una funzione dello stato e dell'ingresso e V^f è un peso terminale sullo stato alla fine dell'orizzonte di predizione. Il problema di ottimizzazione può essere 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 su cui si vuole che lo stato e gli ingressi assumano valori. Inoltre si richiede che lo stato alla fine dell'orizzonte di predizione appartenga ad un certo insieme terminale $\mathbb{X}^f$:

$$x(k+N) \in \mathbb{X}^f \subseteq \mathbb{X} \tag{4.4}$$

Una scelta comune per la cifra di merito è quella di ricorrere a funzione quadratiche[1]:

$$\begin{cases} l(x(k+h), u(k+h)) = ||x(k+h)||_Q^2 + ||u(k+h)||_R^2 \;\; h = 0, \ldots, N-1 \\ \qquad V^f(x(k+N)) = ||x(k+N)||_S^2 \end{cases} \tag{4.5}$$

in cui $Q = Q^\top \succeq 0$, $R = R^\top \succ 0$ e $S = S^\top \succeq 0$ sono matrici di dimensioni opportune che determinano i pesi di stato e ingresso nella cifra di costo. Come precedentemente introdotto, applicando la tecnica *Receding Horizon*, all'istante k si risolve il problema di ottimizzazione e si applica al sistema l'ingresso $u(k|k)$, cioè la prima componente della sequenza ottima $\mathcal{U}^\circ(k)$. Al passo successivo $k+1$, si risolve nuovamente il problema applicando così al sistema $u(k+1|k+1)$, primo elemento di $\mathcal{U}^\circ(k+1)$. Così facendo si ottiene una legge di controllo tempo invariante del tipo $u(k) = K_{MPC}(x(k))$.

4.2.1 Stabilità

Per valutare le condizioni di stabilità del controllo MPC con RH, si introduce il concetto di insieme *Positivamente Invariante*:

Definizione 1. *Un insieme Θ è detto Positivamente Invariante per il sistema autonomo $x(k+1) = f(x(k))$ se:*

$$x(\bar{k}) \in \Theta \Rightarrow x(k) \in \Theta, \forall k \geq \bar{k}$$

[1] Si utilizza la simbologia $||x||_Q^2$ per indicare la forma quadratica $x^\top Q x$

Cioè si richiede che se in un certo istante $\bar{k}$ lo stato è contenuto in tale insieme, allora lo sarà anche in tutti gli instanti successivi.
Si consideri ora una legge di controllo ausiliaria del tipo

$$u(k) = k_a(x(k)) \tag{4.6}$$

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

$$x(k+1) = Ax(k) + Bk_a(x(k)) \tag{4.7}$$

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 $k_a(x) \in \mathbb{U}, \forall x \in \mathbb{X}^f$. Infine, il peso terminale viene scelto tale da essere una funzione di Lyapunov per il sistema controllato, cioè tale che

$$V^f(Ax(k) + Bk_a(x(k))) \leq V^f(x(k)) - l(x(k), k_a(x(k))), \forall x(k) \in \mathbb{X}^f \tag{4.8}$$

In questo modo è possibile garantire la stabilità del controllo MPC. Per una trattazione più completa si rimanda a [22].

4.2.2 Scelta dei Parametri

Fra i vari metodi presentati in letteratura per la scelta del peso e set terminale, V^f e $\mathbb{X}^f$ rispettivamente, si presenta il *Quasi-infinite Horizon MPC*.
L'idea è di prendere V^f uguale al valore della cifra di costo V^N per $N \to \infty$, detto V_∞, cioè il valore che assumerebbe nel corrispondente problema ad orizzonte infinito. A causa dei vincoli sul problema, V_∞ non è noto e si sceglie quindi di prendere l'equivalente, $\hat{V}_\infty$, del problema non vincolato:

$$\hat{V}_\infty(x) = \sum_{k=0}^{\infty} ||x(k)||_Q^2 + ||u(k)||_R^2 \tag{4.9}$$

che corrisponde al classico problema di controllo Lineare Quadratico (LQ). Scelte arbitrariamente le due matrici Q e R, la legge ausiliaria è quindi:

$$u(k) = -K_{LQ}x(k) \tag{4.10}$$

in cui K_{LQ} è ottenuta dalla soluzione P dell'equazione algebrica di Riccati:

$$\begin{cases} P = A^\top PA + Q - A^\top PB(R + B^\top PB)^{-1}B^\top PA \\ K_{LQ} = (R + B^\top PB)^{-1}B^\top PA \end{cases} \tag{4.11}$$

Il valore ottimo della cifra di merito $\hat{V}_\infty(x)$ è pertanto dato da:

$$\hat{V}_\infty^\circ(x) = ||x||_P^2 = x^\top P x \tag{4.12}$$

Si possono quindi scegliere:

$$\begin{cases} V^f = \hat{V}_\infty^o(x) = ||x||_P^2 \\ \mathbb{X}^f = \{x : x^\top P x \leq \alpha\} \subseteq \mathbb{X}, \;\; \alpha \geq 0 \end{cases} \tag{4.13}$$

in cui α viene preso in modo tale che $\mathbb{X}^f \subseteq \mathbb{X}$. La scelta appena discussa per V^f verifica la condizione di stabilità (4.8), infatti:

$$\begin{aligned} \Gamma(x(k)) &\triangleq V^f(Ax(k) - BK_{LQ}x(k)) - V^f(x(k)) + l(x(k), -K_{LQ}x(k)) \\ &= x^\top(k)\{(A - BK_{LQ})^\top P(A - BK_{LQ}) - P + (Q + K_{LQ}^\top R K_{LQ})\}x(k) \\ &= 0 \end{aligned}$$

poichè il nucleo della forma quadratica ottenuta è una riscrittura dell'equazione di Riccati algebrica. Inoltre, poichè $\Gamma(x(k)) = 0$, $\mathbb{X}^f$ risulta essere invariante rispetto alla legge ausiliaria (4.10), in quanto è definito su una linea di livello della funzione di Lyapunov associata al sistema retroazionato. Si può quindi concludere che questa scelta dei parametri garantisce la stabilità del sistema controllato. Per maggiori dettagli si rimanda a [27].

4.3 Tube Based MPC

Le tecniche classiche 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 eccessivamente da quella predetta. Questo problema è però di notevole interesse nel caso pratico poichè incertezze e disturbi sono sempre presenti a causa, ad esempio, di errori di modello, errori di misura ed errori di attuazione. Come descritto nel Capitolo 2, il modello utilizzato per descrivere la dinamica di ciascun robot presenta delle approssimazioni. Inoltre, la misura della posizione è affetta da imprecisioni dovute sia alla distorsione della telecamera che alla semplicità dell'algoritmo di elaborazione dell'immagine implementato. Con il fine di ottenere elevate prestazioni del sistema di controllo, è quindi opportuno progettare una soluzione che garantisca robustezza

rispetto alle problematiche evidenziate.
A tal proposito si presenta la tecnica *Tube-based MPC* [23], la quale garantisce che lo stato del sistema controllato resti all'interno di un insieme limitato nell'intorno dello stato nominale a fronte di disturbi limitati. Il nome di questa tecnica deriva dal fatto che la traiettoria reale dello stato giace all'interno di un "tubo" centrato nella traiettoria nominale, come rappresentato in Fig. 4.2 per un sistema del secondo ordine. Tale garanzia sarà utile per risolvere in maniera distribuita il problema di coordinamento, analizzato nel prosieguo della tesi. Inoltre, si ricorda che, oltre a garantire robustezza, è anche necessario soddisfare eventuali vincoli su stato e ingressi.

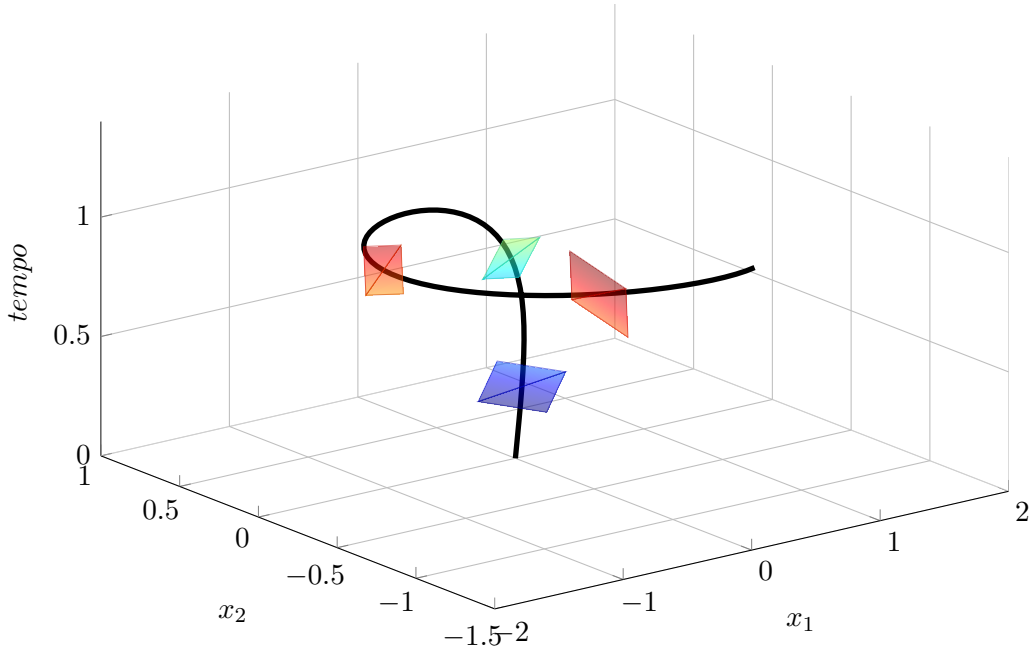


Figura 4.2: Evoluzione della traiettoria dello stato con Tube-based MPC

Si consideri un sistema lineare:

$$x(k+1) = Ax(k) + Bu(k) + w(k),\ w(k) \in \mathbb{W} \tag{4.14}$$

in cui $w(k)$ è un disturbo agente sul sistema e $\mathbb{W}$ è un insieme compatto e contenente l'origine, noto a priori. L'obiettivo è quindi quello di trovare una legge di controllo che garantisca proprietà di convergenza ed ottimalità, nel rispetto dei vincoli, indipendentemente dal valore assunto da tale disturbo, purchè in $\mathbb{W}$. Questo disturbo è volto a

rappresentare le problematiche discusse precedentemente, quali errori di misura, di attuazione ed incertezze sul modello stesso. L'idea di questa soluzione di controllo robusto è di utilizzare un sistema nominale in parallelo al sistema reale su cui verrà applicato il controllo MPC classico, che garantirà tale robustezza mediante vincoli più stringenti nel problema di ottimizzazione, come verrà ora studiato.
Si consideri il sistema nominale associato a (4.14):

$$\hat{x}(k+1) = A\hat{x}(k) + B\hat{u}(k) \tag{4.15}$$

Il problema può essere ora riformulato come quello di trovare un ingresso $u(k)$ per il sistema reale a partire dall'ingresso ottimo per il sistema nominale $\hat{u}(k : k+N-1)$ [2] che mantenga lo stato reale $x(k)$ all'interno di un ben definito insieme $\mathbb{X}$, indipendentemente dal valore assunto da $w(k) \in \mathbb{W}$. In particolare, l'ingresso robusto applicato al sistema reale viene ottenuto aggiungendo un termine in *feedback* legato all'errore fra lo stato nominale e lo stato reale:

$$u(k) = \hat{u}(k) + K(x(k) - \hat{x}(k)) \tag{4.16}$$

Si valuta ora la dinamica dell'errore dello stato reale rispetto a quello nominale, chiamato $z(k) = x(k) - \hat{x}(k)$. Sottraendo l'equazione del sistema reale (4.14) a quella del sistema nominale (4.15) si ottiene:

$$x(k+1) - \hat{x}(k+1) = A(x(k) - \hat{x}(k)) + B(u(k) - \hat{u}(k)) + w(k)$$

sostituendo quindi (4.16) si ha:

$$\begin{aligned} x(k+1) - \hat{x}(k+1) &= A(x(k) - \hat{x}(k)) + BK(x(k) - \hat{x}(k)) + w(k) \\ &\Downarrow \\ z(k+1) &= (A + BK)z(k) + w(k) \end{aligned} \tag{4.17}$$

in cui, si ricorda, $z(k) = x(k) - \hat{x}(k)$. Si può quindi affermare che, se $(A + BK)$ è asintoticamente stabile, allora esiste un insieme robustamente positivamente invariante $\mathbb{Z}$ tale che:

$$z(k) \in \mathbb{Z}, w(t) \in \mathbb{W},\ \forall t \geq k \ \Rightarrow\ z(k+h) \in \mathbb{Z}, \forall h \geq 0 \tag{4.18}$$

in cui si è utilizzata la terminologia *robustamente positivamente invariante* (RPI), per indicare che $z(k+h) \in \mathbb{Z}, \forall h > 0$, indipendentemente

[2] Con $\hat{u}(k : k+N-1)$ si intende la sequenza di ingressi $\hat{u}(k), \ldots, \hat{u}(k+N-1)$

dall'entità del disturbo $w(k) \in \mathbb{W}$, purchè $z(k) \in \mathbb{Z}$. In altre parole, se lo stato iniziale è tale che $z(0) \in \mathbb{Z}$, allora $z(k)$ rimarrà sempre all'interno di $\mathbb{Z}$ indipendentemente dall'entità del disturbo $w(k)$, cioè [3]:

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

Questo risultato è di notevole importanza, in quanto garantisce che lo stato del sistema reale sia sempre in un intorno prefissato dello stato del sistema nominale.

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

$$x(k) \in \mathbb{X}, \;\; u(k) \in \mathbb{U} \tag{4.19}$$

poichè il problema di ottimizzazione è risolto sul sistema nominale, è necessario trasformare questi vincoli in vincoli sulle relative variabili di stato e di ingresso. In particolare, i nuovi vincoli devono essere tali da garantire che (4.19) siano rispettati indipendentemente dal valore assunto dal disturbo $w(k)$ e risulteranno quindi più stringenti dei precedenti:

$$\begin{aligned} \hat{x}(k) = x(k) - z(k) \;\; &\Rightarrow \;\; \hat{x}(k) \in \hat{\mathbb{X}} = \mathbb{X} \ominus \mathbb{Z} \\ \hat{u}(k) = u(k) - Kz(k) \;\; &\Rightarrow \;\; \hat{u}(k) \in \hat{\mathbb{U}} = \mathbb{U} \ominus K\mathbb{Z} \end{aligned} \tag{4.20}$$

Si noti che tali vincoli hanno senso solo se lo stato nominale all'istante iniziale k è tale che l'errore $z(k) \in \mathbb{Z}$, altrimenti non c'è nessuna garanzia che $z(\cdot)$ rimanga in $\mathbb{Z}$. Una possibile scelta è quella di porre $\hat{x}(0) = x(0)$, che chiaramente soddisfa quanto detto, e lasciare che $\hat{x}(k)$ evolva autonomamente secondo la legge (4.15).

Prima di definire il problema di ottimizzazione si devono 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 ausiliaria $\hat{u}(k) = K\hat{x}(k)$ e tale che l'ingresso sia ammissibile nel rispetto dei vincoli, cioè:

$$\hat{x}(\bar{k}) \in \hat{\mathbb{X}}^f \;\Rightarrow\; \begin{cases} \hat{x}(k+1) = (A + BK)\hat{x}(k) \in \hat{\mathbb{X}}^f \\ \hat{u}(k) = K\hat{x}(k) \in \hat{\mathbb{U}} \end{cases} \quad , \forall k \geq \bar{k} \tag{4.21}$$

[3] Il simbolo $\oplus$ indica la somma di Minkowsi fra due insiemi, definita come: $A \oplus B = \bigcup_{b \in B} \{a + b | a \in A\}$. Analogamente, il simbolo $\ominus$ indica la differenza di Minkowski definita come: $C = A \ominus B = \{c : c \oplus B \subseteq A\}$.

A questo punto si definisce il problema di ottimizzazione mediante la ricerca del minimo di una funzione di costo quadratica, come presentato nella Sezione 4.2, soggetto ai vincoli appena presentati:

$$\begin{aligned}\min_{\hat{u}(k:k+N-1)} & \sum_{h=0}^{N-1} \left\|\hat{x}(k+h)\right\|_Q^2 + \left\|\hat{u}(k+h)\right\|_R^2 + \left\|\hat{x}(k+N))\right\|_P^2 \\ & \hat{x}(k+1) = A\hat{x}(k) + B\hat{u}(k) \\ & \hat{x}(k+h) \in \hat{\mathbb{X}},\ h = 0, \ldots, N-1 \\ & \hat{u}(k+h) \in \hat{\mathbb{U}},\ h = 0, \ldots, N-1 \\ & \hat{x}(k+N) \in \hat{\mathbb{X}}^f\end{aligned} \tag{4.22}$$

Tornando alla scelta dello stato iniziale del sistema nominale, si ricorda che è sufficiente garantire che $x(k) - \hat{x}(k) \in \mathbb{Z}$ per ottenere una traiettoria dello stato reale tale che $x(k+h) \in \hat{x}(k+h) \bigoplus \mathbb{Z}$. É quindi possibile rendere tale stato iniziale una nuova variabile di ottimizzazione per il problema precedente, imponendo il vincolo descritto. Il problema di ottimizzazione è perció riformulato come segue:

$$\begin{aligned}\min_{\hat{x}(k),\hat{u}(k:k+N-1)} & \sum_{h=0}^{N-1} \left\|\hat{x}(k+h)\right\|_Q^2 + \left\|\hat{u}(k+h)\right\|_R^2 + \left\|\hat{x}(k+N))\right\|_P^2 \\ & \hat{x}(k+1) = A\hat{x}(k) + B\hat{u}(k) \\ & \hat{x}(k+h) \in \hat{\mathbb{X}},\ h = 0, \ldots, N-1 \\ & \hat{u}(k+h) \in \hat{\mathbb{U}},\ h = 0, \ldots, N-1 \\ & \hat{x}(k+N) \in \hat{\mathbb{X}}^f \\ & x(k) - \hat{x}(k) \in \mathbb{Z}\end{aligned} \tag{4.23}$$

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

$$u(k) = \hat{u}(k|k) + K(x(k) - \hat{x}(k|k)) \tag{4.24}$$

in cui $\hat{u}(k|k)$ e $\hat{x}(k|k)$ sono ottenuti dall'ottimizzazione del problema (4.23) all'istante k. Lo schema di controllo è presentato in Fig. 4.3. Utilizzando questo schema, dunque, si ha che, nelle ipotesi di disturbi limitati, l'evoluzione del sistema reale si mantiene vicina a quella predetta.

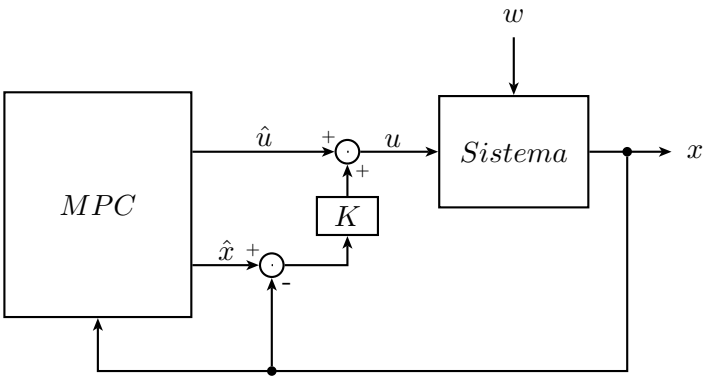


Figura 4.3: Schema di controllo Tube-based MPC

4.4 Applicazione del Controllo Predittivo robusto per l'inseguimento di traiettorie di riferimento delle uscite

Si vuole ora studiare il problema di controllare un sistema in modo tale che le sue uscite inseguano un dato riferimento, garantendo robustezza rispetto ad incertezze e disturbi agenti sul sistema. In particolare, nel contesto di questo lavoro, tale problema può essere interpretato come quello di far inseguire ad un robot una data traiettoria, la cui generazione è momentaneamente trascurata. In particolare si utilizzerà il controllo *Tube-Based MPC* con l'obiettivo di garantire che le uscite del sistema sotto controllo si mantengano in un intorno di dimensioni note della traiettoria di riferimento.
La strategia di controllo proposta opera su due livelli:

(a) Generazione delle traiettorie di riferimento per lo stato e per l'ingresso.

(b) Controllo MPC robusto con l'obiettivo di minimizzare la distanza fra le traiettoria di riferimento generate al punto precedente e quelle effettivamente seguite dal sistema reale.

In altre parole, il problema di inseguimento di traiettorie di riferimento delle uscite viene tradotto in un problema di inseguimento di traiettorie di stato ed ingresso del sistema. Verranno ora analizzati nel dettaglio i due livelli sopra descritti.

4.4.1 Generazione delle traiettorie di riferimento per lo stato e per l'ingresso

Si supponga di conoscere una traiettoria di riferimento da inseguire nell'orizzonte di predizione:

$$\tilde{y}(k : k+N-1) = \Big(\tilde{y}(k), \ldots, \tilde{y}(k+N-1)\Big)$$

Si vogliono ricavare la corrispondente traiettoria $\tilde{x}$ che lo stato del sistema deve nominalmente inseguire e i corrispondenti ingressi $\tilde{u}$ da applicare al sistema affinchè la traiettoria di riferimento $\tilde{y}(k : k+N-1)$ sia correttamente inseguita. A tal proposito, diversamente da quanto fatto in [11] in cui si è optato per un osservatore, si utilizza un classico sistema per l'inseguimento di un riferimento per le variabili di uscita che rispetta il principio del modello interno. Tale principio prevede l'introduzione di una variabile di stato aggiuntiva corrispondente all'errore di inseguimento integrato. Poichè ogni punto della traiettoria di riferimento può essere visto come un riferimento costante che il sistema deve inseguire, la variabile di stato aggiuntiva sarà un integratore, risultando nel seguente sistema:

$$\begin{cases} \tilde{x}(k+1) = A\tilde{x}(k) + B\tilde{u}(k) \\ \tilde{e}(k+1) = \tilde{e}(k) + \tilde{y}(k+1) - C\tilde{x}(k) \\ \tilde{u}(k) = K_x\tilde{x}(k) + K_e\tilde{e}(k) \end{cases} \tag{4.25}$$

in cui le matrici A, B e C replicano la dinamica dell'agente controllato e K_x, K_e sono parametri.
La scelta di tali parametri deve essere tale da garantire la stabilità del sistema (4.25); sostituendo l'equazione di $\tilde{u}(k)$, si ottiene la seguente forma matriciale:

$$\underbrace{\begin{pmatrix} \tilde{x}(k+1) \\ \tilde{e}(k+1) \end{pmatrix}}_{\tilde{\chi}(k+1)} = \underbrace{\begin{bmatrix} A+BK_x & BK_e \\ -C & I \end{bmatrix}}_{\mathcal{F}} \underbrace{\begin{pmatrix} \tilde{x}(k) \\ \tilde{e}(k) \end{pmatrix}}_{\tilde{\chi}(k)} + \underbrace{\begin{bmatrix} 0 \\ I \end{bmatrix}}_{\mathcal{G}} \tilde{y}(k+1) \tag{4.26}$$

che risulta asintoticamente stabile se la matrice $\mathcal{F}$ ha tutti gli autovalori all'interno della circonferenza unitaria centrata nell'origine: una matrice che gode di questa proprietà è detta *Schur* [4]. Si devono quindi trovare K_x e K_e tali da ottenere $\mathcal{F}$ Schur. Riscrivendo la matrice

[4] Con l'espressione "matrice asintoticamente stabile" si intenderà matrice Schur

dinamica $\mathcal{F}$ come segue:

$$\mathcal{F} = \begin{bmatrix} A + BK_x & BK_e \\ -C & I \end{bmatrix} = \underbrace{\begin{bmatrix} A & 0 \\ -C & I \end{bmatrix}}_{\bar{A}} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{\bar{B}} \begin{bmatrix} K_x & K_e \end{bmatrix} \tag{4.27}$$

ci si è riportati ad un classico problema di stabilizzazione per il quale esistono efficienti algoritmi di soluzione, purchè la coppia $(\bar{A}, \bar{B})$ 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 $\tilde{y}(\cdot)$ sia costante. In altre parole, il sistema (4.25) garantisce un perfetto inseguimento nel caso in cui si mantenga il riferimento $\tilde{y}_k$ costante fintantochè 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 però sempre verificato, in quanto si hanno traiettorie tempo-varianti: è allora opportuno studiare quanto la traiettoria inseguita dal sistema (4.25), si discosti da quella di riferimento. A tal proposito, si consideri il valore di regime, chiamato $\tilde{\chi}^{SS}(k)$, dello stato allargato $\tilde{\chi}(k)$, definito in (4.26), corrispondente al riferimento costante pari a $\tilde{y}(k)$. Imponendo la condizione di regime al sistema (4.26) si ottiene [5]:

$$\tilde{\chi}_k^{SS} = \mathcal{F}\tilde{\chi}_k^{SS} + \mathcal{G}\tilde{y}_k \tag{4.28}$$

Analogamente, imponendo il riferimento costante pari a $\tilde{y}_{k+1}$, si ha:

$$\tilde{\chi}_{k+1}^{SS} = \mathcal{F}\tilde{\chi}_{k+1}^{SS} + \mathcal{G}\tilde{y}_{k+1} \tag{4.29}$$

Sottraendo le due equazioni appena ricavata si ottiene:

$$\begin{aligned} \tilde{\chi}_{k+1}^{SS} - \tilde{\chi}_k^{SS} &= \mathcal{F}(\tilde{\chi}_{k+1}^{SS} - \tilde{\chi}_k^{SS}) + \mathcal{G}(\tilde{y}_{k+1} - \tilde{y}_k) \\ &\Downarrow \\ \tilde{\chi}_{k+1}^{SS} - \tilde{\chi}_k^{SS} &= (I - \mathcal{F})^{-1}\mathcal{G}(\tilde{y}_{k+1} - \tilde{y}_k) \end{aligned} \tag{4.30}$$

Con l'obiettivo di trovare un'equazione dinamica che descriva l'evoluzione dell'errore fra lo stato $\tilde{\chi}_k$ ed il corrispondente valore di regime $\tilde{\chi}_k^{SS}$, si consideri la differenza fra (4.26) e (4.29):

$$\begin{aligned} \tilde{\chi}_{k+1} - \tilde{\chi}_{k+1}^{SS} &= \mathcal{F}(\tilde{\chi}_k - \tilde{\chi}_{k+1}^{SS}) \\ &= \mathcal{F}(\tilde{\chi}_k - \tilde{\chi}_k^{SS}) - \mathcal{F}(\tilde{\chi}_{k+1}^{SS} - \tilde{\chi}_k^{SS}) \\ &= \mathcal{F}(\tilde{\chi}_k - \tilde{\chi}_k^{SS}) - \mathcal{F}(I - \mathcal{F})^{-1}\mathcal{G}(\tilde{y}_{k+1} - \tilde{y}_k) \\ &= \mathcal{F}(\tilde{\chi}_k - \tilde{\chi}_k^{SS}) + \tilde{w}_k \end{aligned} \tag{4.31}$$

[5] Di qui in avanti si introduce la notazione $w_k = w(k)$ per facilitare la lettura

A questo punto, assumendo che:

$$\tilde{y}_{k+1} - \tilde{y}_k \in \beta_\varepsilon(0) \tag{4.32}$$

con $\beta_\varepsilon(0)$ un insieme limitato, compatto e centrato nell'origine, si ha che il disturbo $\tilde{w}_k$ è limitato, infatti:

$$\tilde{w}_k = -(I - \mathcal{F})^{-1}\mathcal{F}\mathcal{G}(\tilde{y}_{k+1} - \tilde{y}_k) \in -(I - \mathcal{F})^{-1}\mathcal{F}\mathcal{G}\beta_\varepsilon(0) = \tilde{\mathbb{W}} \tag{4.33}$$

Il vincolo (4.32) corrisponde ad una limitazione della distanza fra due punti successivi della traiettoria generata o, equivalentemente ad una limitazione implicita della velocità dell'agente. Questo aspetto verrà discusso nei capitoli successivi, congiuntamente all'effetto che tale vincolo porta alle prestazioni del sistema di controllo. Dunque, se la matrice $\mathcal{F}$ è asintoticamente stabile, allora esiste un set robustamente positivamente invariante Δ^χ per il sistema (4.31), ottenuto come:

$$\Delta^\chi = \bigoplus_{h=0}^{\infty} \mathcal{F}^h \tilde{\mathbb{W}} \tag{4.34}$$

Ciò significa che l'evoluzione dell'errore $\tilde{\chi}_{k+h} - \tilde{\chi}^{SS}_{k+h}, \forall h > 0$ resta confinata in Δ^χ indipendentemente dal valore assunto da w_k, posto che $\tilde{\chi}_k - \tilde{\chi}^{SS}_k \in \Delta^\chi$. Il set Δ^χ 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, ricordando la definizione della variabile $\tilde{\chi}_k$ si può scrivere:

$$\begin{cases} \tilde{\chi}_k \in \tilde{\chi}^{SS}_k \oplus \Delta^\chi \\ \tilde{x}_k = \begin{bmatrix} I & 0 \end{bmatrix} \tilde{\chi}_k \end{cases} \Rightarrow \quad \tilde{x}_k \in \tilde{x}^{SS}_k \oplus \begin{bmatrix} I & 0 \end{bmatrix} \Delta^\chi \tag{4.35}$$

Inoltre, poichè a regime si ha che $C\tilde{x}^{SS}_k = \tilde{y}_k$, si può ulteriormente scrivere:

$$C\tilde{x}_k \in C\tilde{x}^{SS}_k \oplus C \begin{bmatrix} I & 0 \end{bmatrix} \Delta^\chi = \tilde{y}_k \oplus C \begin{bmatrix} I & 0 \end{bmatrix} \Delta^\chi \tag{4.36}$$

Tale informazione permetterà, unita a quanto discusso nelle sezioni successive, di determinare a priori un intorno della traiettoria di riferimento in cui si avrà la certezza di trovare l'agente, che potrà essere utilizzato, ad esempio, per evitare la collisione con eventuali ostacoli.

4.4.2 Inseguimento robusto dello stato e dell'ingresso di riferimento

Si vuole ora applicare la tecnica *Tube-based MPC* con l'obiettivo di ottenere un inseguimento robusto delle traiettorie di riferimento dello stato e dell'ingresso generate dal sistema introdotto nella sezione precedente. Si ripercorreranno i passaggi illustrati nella Sezione 4.3 con riferimento al controllo di un singolo agente, il cui modello è:

$$x_{k+1} = Ax_k + Bu_k + w_k, \quad w_k \in \mathbb{W} \tag{4.37}$$

in cui w_k è un disturbo limitato agente sul sistema, volto a rappresentare errori di modello e di attuazione e disturbi nella misura della posizione. Il sistema nominale, scelto per essere poi utilizzato nel controllo MPC robusto, ha la struttura:

$$\hat{x}_{k+1} = A\hat{x}_k + B\hat{u}_k \tag{4.38}$$

Con riferimento a [11], si noti la semplificazione ottenuta nel modello nominale, che coincide ora con il sistema reale (4.37) a meno del disturbo additivo w_k.
Ricordando che l'ingresso del sistema reale è ottenuto come:

$$u_k = \hat{u}_k + K(x_k - \hat{x}_k) = \hat{u}_k + K\varepsilon_k \tag{4.39}$$

la dinamica dell'errore $\varepsilon_k = x_k - \hat{x}_k$, ottenuta come in (4.17), risulta

$$\varepsilon_{k+1} = (A + BK)\varepsilon_k + w_k \tag{4.40}$$

Assumendo che $(A + BK)$ sia asintoticamente stabile, si definisce l'insieme positivamente robustamente invariante per l'errore come:

$$\mathcal{E} = \bigoplus_{h=0}^{\infty} (A + BK)^h \, \mathbb{W} \tag{4.41}$$

Ricordando la definizione di set RPI, si può quindi affermare che lo stato x_k del sistema reale è sempre in un intorno limitato dello stato nominale $\hat{x}_k$, infatti:

$$\varepsilon_k = x_k - \hat{x}_k \in \mathcal{E} \;\Rightarrow\; x_k \in \hat{x}_k \oplus \mathcal{E} \tag{4.42}$$

Si consideri ora il seguente vincolo:

$$C(\hat{x}_k - \tilde{x}_k) \in \Delta^z \;\Rightarrow\; C\hat{x}_k \in C\tilde{x}_k \oplus \Delta^z, \quad k = 0, \ldots, N-1 \tag{4.43}$$

in cui C è la trasformazione di uscita. L'utilità di tale vincolo verrà chiarita nel prosieguo. Si noti, per il momento, che esso porta a definire un intorno dell'uscita riferita alla traiettoria di riferimento dello stato in cui si è certi trovare l'uscita del sistema reale. Infatti, unendo (4.43) e (4.42) si ottiene:

$$x_k \in \hat{x}_k \oplus \mathcal{E} \Rightarrow Cx_k \in C\hat{x}_k \oplus C\mathcal{E} \Rightarrow Cx_k \in C\tilde{x}_k \oplus \Delta^z \oplus C\mathcal{E} = C\tilde{x}_k \oplus \mathcal{Z} \tag{4.44}$$

L'informazione fornita dal set $\mathcal{Z} = \Delta^z \oplus C\mathcal{E}$, come sarà discusso nel seguito, 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.

Problema di ottimizzazione

Come analizzato nella Sezione 4.3, il problema di ottimizzazione è formulato rispetto al vettore di ingresso nominale nell'orizzonte di predizione $\hat{u}_{k:k+N-1}$ e allo stato iniziale $\hat{x}_k$. Poichè l'obiettivo è minimizzare lo scostamento fra lo stato e l'ingresso nominale rispetto ai rispettivi riferimenti, si considera il seguente problema:

$$\min_{\hat{x}_k, \hat{u}_{k:k+N-1}} \sum_{h=0}^{N-1} \left\|\hat{x}_{k+h} - \tilde{x}_{k+h}\right\|_Q^2 + \left\|\hat{u}_{k+h} - \tilde{u}_{k+h}\right\|_R^2 + \left\|\hat{x}_{k+N} - \tilde{x}_{k+N}\right\|_P^2 \tag{4.45}$$

soggetto ai seguenti vincoli, precedentemente ricavati:

- Lo stato inziale del sistema nominale non si deve discostare eccessivamente dallo stato misurato del sistema reale:

$$x_k - \hat{x}_k \in \mathcal{E} \tag{4.46}$$

- I vincoli imposti su stato ed ingresso del sistema reale devono essere trasformati in vincoli più stringenti su stato ed ingresso nominale, garantendo che l'agente li rispetti indipendentemente dal disturbo. Analogamente a (4.20):

$$x_{k+h} \in \mathbb{X} \;\Rightarrow\; \hat{x}_{k+h} \in \hat{\mathbb{X}} = \mathbb{X} \ominus \mathcal{E}, \;\; \forall h = 0, \ldots, N-1 \tag{4.47}$$

$$u_{k+h} \in \mathbb{U} \;\Rightarrow\; \hat{u}_{k+h} \in \hat{\mathbb{U}} = \mathbb{U} \ominus K\mathcal{E}, \;\; \forall h = 0, \ldots, N-1 \tag{4.48}$$

- Limite allo scostamento fra la posizione nominale e quella di riferimento:

$$C(\hat{x}_{k+h} - \tilde{x}_{k+h}) \in \Delta^z, \;\; \forall h = 0, \ldots, N-1 \tag{4.49}$$

- Lo stato nominale alla fine dell'orizzonte di predizione deve appartenere ad un set Σ robustamente positivamente invariante rispetto al sistema in anello chiuso a cui è applicata la legge di controllo ausiliaria $\hat{u}_k = K\hat{x}_k$. Poichè tale sistema in anello chiuso ha la stessa struttura di (4.40), il set cercato si trova come in (4.41):

$$\hat{x}_{k+1} = (A + BK)\hat{x}_k + w_k \;\Rightarrow\; \Sigma = \bigoplus_{h=0}^{\infty} (A + BK)^h \, \mathbb{W} = \mathcal{E}$$

  Poichè si è scelto di utilizzare lo stesso parametro K sia nelle legge ausiliaria che in (4.39), si ottiene $\Sigma = \mathcal{E}$. Il vincolo da porre è quindi:

$$\hat{x}_{k+N} - \tilde{x}_{k+N} \in \mathcal{E} \tag{4.50}$$

  Si noti come il set terminale non sia definito in modo assoluto, ma dipenda dallo stato di riferimento alla fine dell'orizzonte di predizione.

Il problema di ottimizzazione completo è quindi:

$$\begin{aligned}
\min_{\hat{x}_k, \hat{u}_{k:k+N-1}} \; & \sum_{h=0}^{N-1} \left\| \hat{x}_{k+h} - \tilde{x}_{k+h} \right\|_Q^2 + \left\| \hat{u}_{k+h} - \tilde{u}_{k+h} \right\|_R^2 + \left\| \hat{x}_{k+N} - \tilde{x}_{k+N} \right\|_P^2 \\
& \hat{x}_{k+1} = A\hat{x}_k + B\hat{u}_k \\
& \hat{x}_{k+h} \in \hat{\mathbb{X}}, \;\; \forall h = 0, \ldots, N-1 \\
& \hat{u}_{k+h} \in \hat{\mathbb{U}}, \;\; \forall h = 0, \ldots, N-1 \\
& x_k - \hat{x}_k \in \mathcal{E} \\
& C(\hat{x}_{k+h} - \tilde{x}_{k+h}) \in \Delta^z, \;\; \forall h = 0, \ldots, N-1 \\
& \hat{x}_{k+N} - \tilde{x}_{k+N} \in \mathcal{E}
\end{aligned} \tag{4.51}$$

4.5 Generazione della traiettoria per un singolo agente

Si consideri ora il problema di generare una traiettoria di riferimento che l'agente dovrà inseguire utilizzando le tecniche di controllo descritte nella Sezione 4.4. La struttura di controllo robusto presentata permette di 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 presentato

in 4.4.1 e 4.4.2 è la traiettoria di riferimento delle uscite, su cui non sono imposti vincoli riguardanti la tipologia e la dinamica degli agente. 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. Rispetto a quanto trattato in [11], in cui la generazione della traiettoria era realizzata insieme all'inseguimento robusto di stato ed ingresso, si hanno quindi due problemi separati che possono essere risolti indipendentemente, in modo più veloce e anche, come detto, attraverso algoritmi di natura diversa.

In vista dello studio del coordinamento discusso nel capitolo successivo, si presenta una tecnica per la generazione della traiettoria basata sul Controllo Predittivo. Si consideri il semplice problema di raggiungere un *goal* noto, chiamato $y_{setpoint}$, da parte di un singolo agente, in un ambiente privo di ostacoli. La soluzione ideale, che corrisponde ad imporre come riferimento il goal stesso, renderebbe il problema di controllo generalmente non risolvibile in quanto l'agente potrebbe non essere in grado di raggiungere tale riferimento nell'orizzonte di predizione, portando ad un problema di ottimizzazione non ammissibile. L'idea è quindi quella di generare, in modo incrementale, una successione di riferimenti intermedi $\tilde{y}_{k+h}$ che portino al *goal*, rispettando però dei limiti di distanza fra due punti successivi. A tal proposito si ricorre al seguente problema di ottimizzazione:

$$\begin{aligned} \min_{\bar{y}_{k+N}} & \gamma \left\| \bar{y}_{k+N} - \tilde{y}_{k+N-1} \right\|^2 + \left\| \bar{y}_{k+N} - y_{setpoint} \right\|_T^2, \quad T \succ \gamma I \\ & \bar{y}_{k+N} - \tilde{y}_{k+N-1} \in \beta_\varepsilon(0) \end{aligned} \tag{4.52}$$

Si tratta di un'ottimizzazione ad un passo in cui il nuovo riferimento $\bar{y}_{k+N}$ sarà vicino al precedente e diretto verso il *goal*, grazie ai contributi dei due termini della cifra di costo. L'evoluzione della generazione della traiettoria è rappresentata in Fig. 4.4, in cui si è scelto $\beta_\varepsilon(0)$ circolare. I grafici nella figura citata permettono di osservare l'effetto del vincolo *hard* imposto al problema di ottimizzazione, che per altro corrisponde a (4.32): la distanza massima fra due punti successivi è limitata dalla dimensione di tale vincolo. Infatti, esprimendolo nella forma

$$\bar{y}_{k+N} \in \tilde{y}_{k+N-1} \oplus \beta_\varepsilon(0)$$

è evidente che il punto $\bar{y}_{k+N}$ sia nell'intorno definito da $\beta_\varepsilon(0)$, centrato in $\tilde{y}_{k+N-1}$. Un'altra, ma equivalente, interpretazione di questo vincolo

è la conseguente limitazione della velocità di movimento dell'agente nel piano di lavoro.

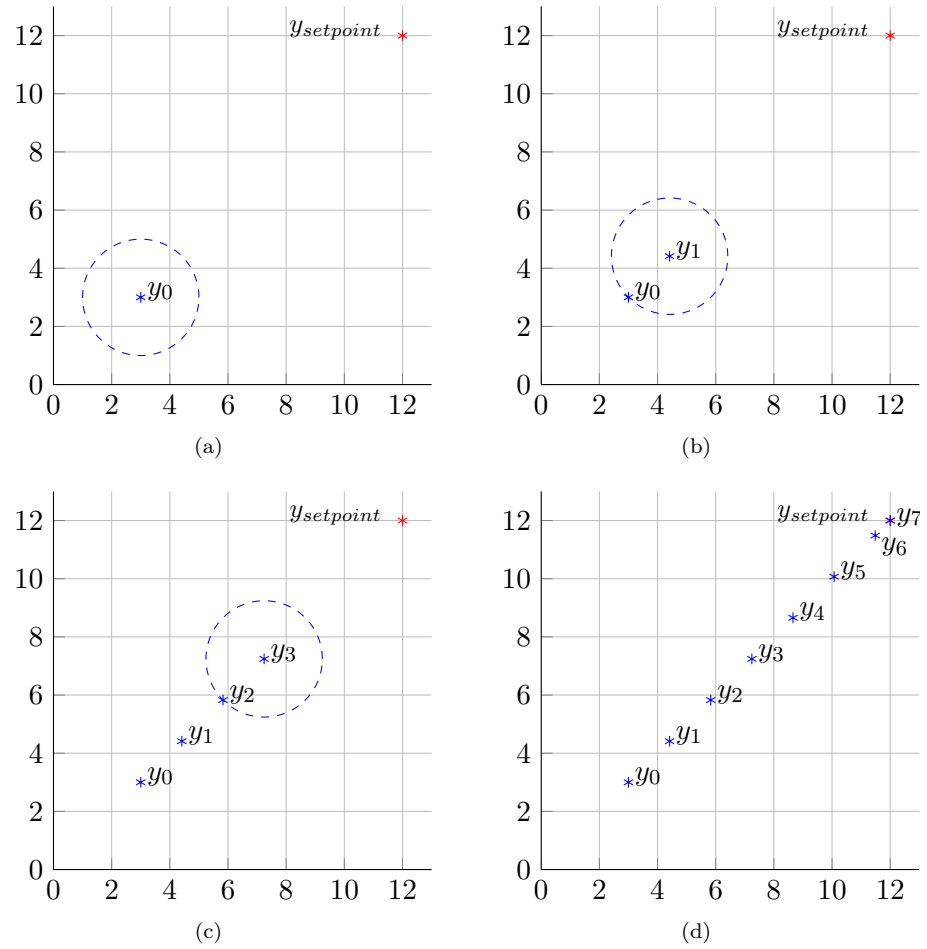


Figura 4.4: Evoluzione della generazione della traiettoria: (a) posizione inziale da cui si ricava la regione in cui si troverà il punto successivo; (b) posizione successiva e relativa regione; (c) terzo punto della traiettoria; (d) traiettoria completa.

Si vuole ora definire una regione di incertezza definita attorno alla traiettoria di riferimento, in cui si avrà la certezza di trovare l'agente. A tal proposito, si ricordano i risultati precedentemente ottenuti:

- La relazione (4.36) impone che l'uscita di riferimento $C\tilde{x}_k$ si trovi in un intorno noto della traiettoria di riferimento $\tilde{y}_k$

- Il vincolo (4.44) limita invece lo scostamento fra l'uscita del sistema reale $y_k = Cx_k$ e quella di riferimento $C\tilde{x}_k$.

Combinando i risultati si è quindi in grado di legare direttamente la traiettoria di riferimento $\tilde{y}_k$ all'uscita del sistema reale y_k. In particolare si ha:

$$\begin{cases} C\tilde{x}_k \in \tilde{y}_k \oplus C \begin{bmatrix} I & 0 \end{bmatrix} \Delta^{\chi} \\ y_k = Cx_k \in C\tilde{x}_k \oplus \mathcal{Z} \end{cases} \Rightarrow \; y_k \in \tilde{y}_k \oplus C \begin{bmatrix} I & 0 \end{bmatrix} \Delta^{\chi} \oplus \mathcal{Z} = \tilde{y}_k \oplus \mathcal{Y} \tag{4.53}$$

Il set $\mathcal{Y} = C \begin{bmatrix} I & 0 \end{bmatrix} \Delta^{\chi} \oplus \mathcal{Z}$ indica quindi l'intorno cercato della traiettoria di riferimento, nella quale si ha la certezza di trovare l'agente. Quest'informazione sarà particolarmente utile nel problema di coordinamento di più agenti, poichè permetterà di evitare la collisione fra gli agenti stessi. L'importanza e l'uso di questo set sarà chiarita nel capitolo successivo, in cui verrà trattato il citato problema del coordinamento.

La scelta del set $\beta_{\varepsilon}(0)$ presenta un *trade-off* fra velocità di movimento del robot e dimensione della regione di incertezza. Infatti, maggiore è la dimensione di $\beta_{\varepsilon}(0)$, maggiore sarà la velocità dell'agente, ma allo stesso tempo, maggiore sarà anche la dimensione del set $\mathcal{Y}$.

4.6 Obstacle Avoidance

Il problema dell'*obstacle avoidance*, grazie alla tecnica di controllo appena presentata, può essere risolto semplicemente aggiungendo un vincolo nel problema di ottimizzazione, che mantenga l'agente ad una distanza minima da ciascun ostacolo, supponendo di conoscere sia la dimensione che la posizione di tutti gli ostacoli presenti.
Questo problema, congiuntamente al problema del *collision avoidance*, verrà analizzato e studiato nel dettaglio nel capitolo successivo, in cui si discuteranno tutti gli aspetti legati alla scelta della citata distanza minima. Si noti che i problemi di *collision* e *obstacle avoidance* sono del tutto analoghi se un ostacolo viene considerato come un robot, la cui posizione è fissa. Per questa motivazione si è scelto di trattare la soluzione di questo problema nel capitolo successivo, dedicato appunto al coordinamento di più agenti.

Capitolo 5

Controllo Predittivo per il Coordinamento

5.1 Introduzione

Si vuole ora considerare il problema di coordinamento di più agenti mediante tecniche di controllo predittivo. In particolare si analizzerà il controllo in formazione abbinato alla necessità di raggiungere un certo punto nello spazio di lavoro. MPC rappresenta una metodologia di controllo particolarmente adatta in questo contesto poichè, come anticipato, permette di inserire nella funzione di costo che caratterizza il problema obiettivi diversi ed eventualmente anche in contrasto fra di loro come, appunto, il mantenimento di una formazione rispetto al raggiungimento di un punto nel piano di lavoro. Poichè il problema di ottimizzazione viene risolto ad ogni istante di campionamento, è possibile variare dinamicamente sia i vincoli che la cifra di costo in funzione della configurazione degli agenti stessi, permettendo così di risolvere problemi di notevole complessità.
MPC fornisce anche la predizione futura dell'uscita del sistema per i successivi N passi che, nel contesto corrente, corrisponde alla traiettoria di ciascun agente; tale predizione, unita alla garanzia di rispettare la traiettoria prevista entro certi limiti, può essere utilizzata per informare gli altri agenti della propria futura posizione, in modo da ottenere una generazione della traiettoria priva di collisioni.

La struttura del sistema di controllo scelta è di tipo distribuito, che consiste in un sistema costituito da diversi controllori locali, distribuiti nei vari agenti, i quali si scambiano informazioni utili a risolvere il pro-

blema di controllo globale. Nel contesto di questo lavoro, gli agenti si scambieranno le informazioni relative alle rispettive posizioni cosicchè la configurazione del sistema sia nota ai singoli controllori, che potranno quindi elaborare adeguate traiettorie. Tale soluzione di controllo, che unisce il controllo predittivo ad un struttura distribuita, è detta *Distributed Predictive Control* (DPC). La rete di comunicazione fra gli agenti può essere descritta attraverso un grafo orientato, i cui nodi indicizzano gli agenti e gli archi rappresentano le trasmissioni fra le varie coppie di agenti. Si consideri, a titolo di esempio, il grafo in Fig. 5.1: l'agente 1 riceve le informazioni sulle posizioni da tutti gli altri agenti nel sistema, il 2 riceve informazioni solo dall'agente 4, il quale non è a conoscenza dello stato degli altri robot.

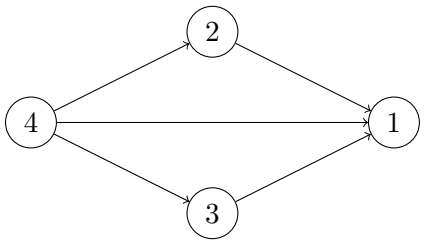


Figura 5.1: Grafo rete di comunicazione nel controllo distribuito

5.2 Struttura del sistema di controllo

Si consideri un sistema di M agenti. Ciascun agente, indicato con $i = 1, \ldots, M$, è dotato di un controllore basato sulle tecniche presentate nel capitolo precedente, la cui struttura è costituita da tre livelli [1]:

1. **Generazione della Traiettoria**: produce una traiettoria di riferimento, intesa come una successione di posizioni, che l'agente dovrà inseguire. Come anticipato, questo livello di controllo non è legato alla dinamica degli agenti, ma ha come unico obiettivo quello di generare, ad ogni instante di tempo, un nuovo riferimento, indicato con $\tilde{y}_k^{[i]}$, in modo tale che, la traiettoria risultante, converga all'obiettivo desiderato, denotato con $y_{setpoint}^{[i]}$, rispettando dei vincoli, quali ad esempio, limitazioni della velocità e dello spazio di lavoro, mantenimento della formazione, *collision*

[1] Le grandezze riferite all'i-esimo agente saranno indicate con l'apice $^{[i]}$

ed *obstacle avoidance*. Questo problema verrà risolto utilizzando MPC, attraverso la definizione di un opportuno problema di ottimizzazione.

2. **Generazione delle traiettorie di riferimento per gli ingressi e le variabili di stato**: in base alla traiettoria generata al livello superiore, viene generata la corrispondente traiettoria che lo stato e gli ingressi dell'agente dovranno inseguire. Dunque, ad ogni istante di tempo, a partire dal riferimento $\tilde{y}_k^{[i]}$, vengono prodotti un nuovo riferimento per lo stato e per gli ingressi, indicati rispettivamente con $\tilde{x}_k^{[i]}$ e $\tilde{u}_k^{[i]}$. Questo livello può essere visto come un'interfaccia tra la traiettoria e la dinamica dell'agente.

3. **Inseguimento degli ingressi e delle variabili di stato**: obiettivo di questo livello è garantire che, anche a fronte di disturbi, la traiettoria dello stato e dell'ingresso dell'agente e rimangano vicine ai riferimenti generati dal livello precedente. In altre parole, verrà generata un'opportuna azione di controllo $u_k^{[i]}$ da applicare all'agente in modo tale che le traiettorie $\tilde{x}_k^{[i]}$ e $\tilde{u}_k^{[i]}$, prodotte al livello superiore, siano inseguite nel rispetto dei vincoli, da parte dello stato $x_k^{[i]}$ e dell'ingresso $u_k^{[i]}$ dell'agente stesso. Come dimostrato nel Capitolo 4, questo garantisce che la traiettoria di riferimento generata nel primo livello dell'architettura di controllo sia inseguita con un'incertezza nota e di entità limitata.

Il livello di generazione della traiettoria, coerentemente con l'analisi nella Sezione 4.5, è indipendente dai livelli sottostanti. Ciò permette una più facile soluzione del problema di coordinamento, poichè le interazioni fra gli agenti e tutte le problematiche relative al mantenimento della formazione sono gestite in tale livello, ignorando il problema dell'inseguimento delle traiettorie generate. In Fig. 5.2, è presentato lo schema di controllo discusso, in cui si nota, come detto, che l'interazione fra quest'ultimi è gestita solo nel livello superiore. Perciò, i problemi di generazione della traiettoria ed inseguimento sono risolti indipendentemente, rendendo così, rispetto a quanto trattato in [11], più semplice il primo problema e permettendo quindi la definizione di comportamenti più complessi.

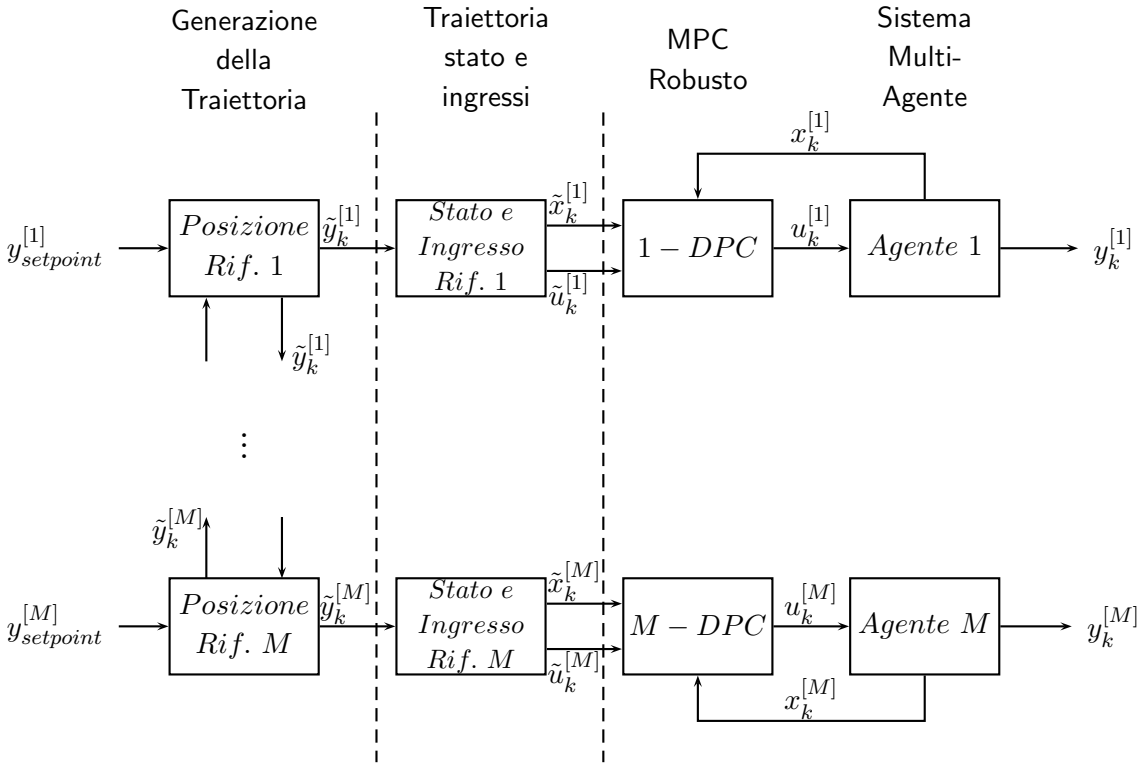


Figura 5.2: Schema di controllo per il coordinamento

5.3 Generazione delle Traiettorie

Il livello di generazione della traiettoria ha il compito di fornire i riferimenti di posizione ai livelli sottostanti nel rispetto di determinati vincoli. In particolare si vuole studiare il problema di mantenere in formazione tre agenti evitando la collisione sia con ostacoli noti che fra loro stessi. In questo livello di controllo si è scelta una strategia MPC non robusta poichè l'obiettivo è, come detto, la sola generazione dei riferimenti di posizione; la robustezza del sistema di controllo viene fornita dai livelli inferiori attraverso le tecniche sopra descritte.

Nel prosieguo del capitolo verranno presentate due diverse soluzioni al problema indicato. In particolare, esse si differenziano nella metodologia utilizzata per impostare il problema della generazione della traiettoria di riferimento dei *follower*. Si ribadisce ancora una volta, che il problema del coordinamento viene risolto nel solo livello di generazione della traiettoria, grazie alla proprietà del sistema di controllo descritte precedentemente.

5.3.1 Mantenimento della formazione con raggiungimento di obiettivi individuali

Si consideri il problema di controllare due agenti, detti *follower*, in modo che "scortino" in formazione triangolare un terzo agente, detto *leader*, fintantochè quest'ultimo non raggiunge il proprio obiettivo. A questo punto i *follower* navigano verso un proprio *goal*, liberi da vincoli di formazione. La flessibilità ottenuta grazie all'utilizzo di cifre di costo, permette di risolvere facilmente il problema pesando opportunamente l'errore di formazione di ciascun *follower* nella prima fase e annullando tale fattore di costo nella seconda.
Si definisce inizialmente la rete di comunicazione fra gli agenti in Fig. 5.3: il *leader* non riceve informazioni sulla posizione dei *follower* e quindi la generazione della traiettoria ad esso relativa, è indipendente da quella degli altri agenti e può essere dunque risolta utilizzando il problema (4.52). Per quanto riguarda i *follower*, è necessario studiare il grafo per determinare le precedenze reciproche, con l'obiettivo di evitare possibili collisioni fra gli stessi. In particolare, dal grafo presentato, si ha che il *follower* F_2 deve dare precedenza al *follower* F_1 poichè il viceversa non sarebbe realizzabile dato che F_1 non è a conoscenza della posizione di F_2. Perciò nel problema di ottimizzazione del *follower*

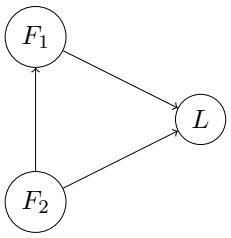


Figura 5.3: Grafo Formazione

F_1 si deve includere un vincolo che garantisca il mantenimento della distanza minima voluta, chiamata h_{1L}, dal solo *leader*. La scelta della distanza minima deve considerare l'incertezza fra la posizione reale dell'agente e quella di riferimento generata. A tal proposito si richiama il set limitato $\mathcal{Y}$, definito in (4.53), che rappresenta l'intorno della traiettoria di riferimento, nel quale si ha la certezza di trovare l'agente. Come mostrato in Fig. 5.4, è sempre possibile determinare la bolla di raggio minimo d_{min} contenente il set $\mathcal{Y}$. La lunghezza d_{min} del raggio trovato corrisponde quindi alla distanza minima da imporre fra due agenti per garantire una traiettoria priva di collisioni. Tale traiettoria

è perciò realizzabile se $h_{1L} > d_{min}$. Si noti, come rimarcato nella Sezio-

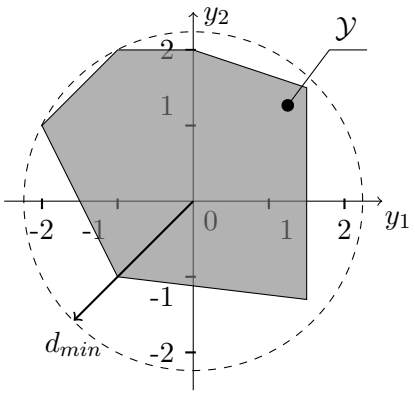


Figura 5.4: Esempio determinazione distanza minima per evitare collisioni fra agenti

ne 4.5, l'importanza della conoscenza a priori del set $\mathcal{Y}$: esso permette di disaccoppiare i livelli di controllo, garantendo comunque una traiettoria priva di collisioni, senza dover considerare l'effettiva posizione del robot. Il problema della generazione della traiettoria del *follower* F_1, può essere facilmente ottenuto a partire dal problema (4.52), aggiungendo un peso legato all'errore di formazione e un vincolo sulla distanza, risultando in:

$$\begin{aligned} \min_{\bar{y}_{k+N}^{[F_1]},\delta} \; & \gamma \left\| \bar{y}_{k+N}^{[F_1]} - \tilde{y}_{k+N-1}^{[F_1]} \right\|^2 + \left\| \bar{y}_{k+N}^{[F_1]} - y_{setpoint}^{[F_1]} \right\|_T^2 + D\delta^2 \\ & \left\| \bar{y}_{k+N}^{[F_1]} - \tilde{y}_{k+N-1}^{[L]} \right\|_2 = h_{1L} + \delta, \quad h_{1L} \geq d_{min} \\ & \bar{y}_{k+N}^{[F_1]} - \tilde{y}_{k+N-1}^{[F_1]} \in \beta_\varepsilon(0) \\ & \delta \geq 0 \end{aligned} \tag{5.1}$$

in cui δ rappresenta l'errore di formazione e D il suo peso nella cifra di costo.
Analogamente per il secondo *follower*, ricordando che esso deve dare

la precedenza al primo e al *leader*, si utilizza il seguente problema:

$$\begin{aligned}
\min_{\bar{y}_{k+N}^{[F_2]},\delta_L,\delta_{F_1}} \quad & \gamma \left\| \bar{y}_{k+N}^{[F_2]} - \tilde{y}_{k+N-1}^{[F_2]} \right\|^2 + \left\| \bar{y}_{k+N}^{[F_2]} - y_{setpoint}^{[F_2]} \right\|_T^2 + D_1\delta_L^2 + D_2\delta_{F_1}^2 \\
& \left\| \bar{y}_{k+N}^{[F_2]} - \tilde{y}_{k+N-1}^{[L]} \right\|_2 = h_{2L} + \delta_L, \quad h_{2L} \geq d_{min} \\
& \left\| \bar{y}_{k+N}^{[F_2]} - \tilde{y}_{k+N-1}^{[F_1]} \right\|_2 = h_{21} + \delta_{F_1}, \quad h_{21} \geq d_{min} \\
& \bar{y}_{k+N}^{[F_2]} - \tilde{y}_{k+N-1}^{[F_2]} \in \beta_\varepsilon(0) \\
& \delta_L, \delta_{F_1} \geq 0
\end{aligned} \tag{5.2}$$

in cui δ_L e δ_{F_1} sono gli errori di formazione e D_1, D_2 i relativi pesi. La presenza di un vincolo non lineare potrebbe, all'apparenza, complicare notevolmente l'efficienza nella ricerca dell'ottimo, ma il problema proposto è di ridotta complessità e perciò questo aspetto è trascurabile. Si noti inoltre, che, poichè la cifra di costo è quadratica, non sono presenti dei minimi locali, eliminando così un problema tipico nelle ottimizzazioni di funzioni non lineari.
Agendo sui pesi D, D_1 e D_2 è possibile ottenere il comportamento desiderato: supponendo di partire da una configurazione iniziale in cui il *leader* è distante dal proprio obiettivo, si impostano i tali pesi in modo che essi siano determinanti nella cifra di costo, cosicchè i *follower* siano forzati nel restare in formazione, in quanto, eventuali errori sono "costosi" in termini della cifra di merito del problema. Quando il *leader* raggiunge l'obiettivo i pesi vengono annullati in modo che non sussista più nessun vincolo *soft* di formazione ed i *follower* siano quindi liberi di muoversi verso i rispettivi obiettivi.

Si noti che se il vincolo di velocità dei tre agenti è il medesimo, ossia la dimensione della bolla $\beta_\varepsilon(0)$ è uguale nei tre problemi di ottimizzazione, nel caso i *follower* non partano in formazione, essi non saranno in grado di raggiungere la formazione ideale poichè il *leader* procederebbe indisturbato alla massima velocità. Per ovviare è tale problema è possibile variare dinamicamente la dimensione di tale vincolo in funzione dell'errore di formazione in modo che il *leader* rallenti quando i *follower* sono fuori formazione. Si noti che questo accorgimento richiede che i *follower* scambino informazione riguardo alla propria posizione con il *leader*, dando quindi luogo ad un *feedback*, prima non presente, dai *follower* al *leader* stesso. Tornando alla soluzione del problema, ad ogni istante di campionamento si sceglie la dimensione della bolla $\beta_\varepsilon(0)$

calcolando un opportuno valore di ε. Una possibile scelta è:

$$\varepsilon_k = \varepsilon_{min} + \frac{2}{\pi}(\varepsilon_{MAX} - \varepsilon_{min}) \arctan\left(\frac{\lambda}{d}\right) \tag{5.3}$$

in cui ε_{MAX} e ε_{min} sono i limiti desiderati per ε, λ è un parametro e d è l'errore totale di formazione. L'andamento di ε ottenuto con questa scelta è riportato in Fig. 5.5. Infine, poichè quindi la dimensione di

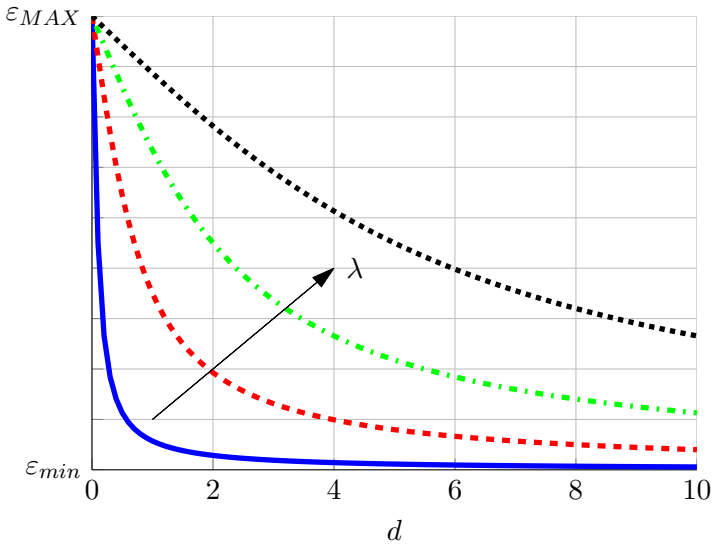


Figura 5.5: Raggio $\beta_\varepsilon(0)$

$\beta_\varepsilon(0)$ relativa alla generazione della traiettoria del *leader* varia istante per istante, il calcolo della dimensione d_{min} del set di incertezza $\mathcal{Y}$ va eseguito nel caso peggiore, cioè quando la dimensione è massima, che equivale a $\varepsilon = \varepsilon_{MAX}$.

5.3.2 Mantenimento della formazione mediante setpoint variabili

Si vuole ora utilizzare un approccio diverso per risolvere il problema del controllo in formazione andando a modificare, ad ogni istante di tempo, il *goal* dei *follower* in funzione della posizione del *leader*. A tal proposito, per ciascun *follower*, si definiscono un angolo ψ_{iL} e una distanza l_{iL} che esso dovrà mantenere dal *leader* come indicato in Fig. 5.6.
Conoscendo direzione e posizione del *leader* è quindi possibile calcolare i *setpoint* per i *follower* utilizzando la caratterizzazione di formazione

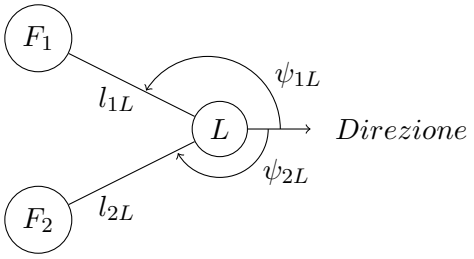


Figura 5.6: Definizione Formazione - Setpoint Variabili

appena introdotta, come segue:

$$y_{setpoint}^{[F_i]} = \tilde{y}^{[L]} + l_{iL} \begin{bmatrix} \cos(\theta + \psi_{iL}) \\ \sin(\theta + \psi_{iL}) \end{bmatrix} \tag{5.4}$$

in cui θ è l'orientamento del *leader*. Il problema nell'utilizzo di questa soluzione è la non conoscenza della direzione del *leader*, per il quale si propongono due metodologie risolutive:

1. Si utilizza la misura dell'orientamento presente nell'anello di linearizzazione presentato nella Sezione 2.4. Questa semplice soluzione, porta ad un accoppiamento dei livelli della struttura di controllo perdendo la generalità propria del livello di generazione della traiettoria.

2. Si determina il vettore indicante la direzione mediante differenza fra il riferimento attuale e quello all'istante precedente. Tale vettore permette poi un'immediata stima dell'angolo. Questa soluzione fornisce però risultati non prevedibili nel caso in cui il *leader* sia fermo poichè il vettore direzione è nullo. Supponendo però che il *leader* si fermi solo quando raggiunge il suo obiettivo, questa situazione può essere facilmente evitata. D'altro canto, essa presenta il vantaggio di generare riferimenti rispetto alla traiettoria ideale del *leader* e non a quella effettiva. Ad esempio, nel caso in cui il *leader* abbia un moto oscillatorio attorno alla traiettoria di riferimento, si ottiene comunque una formazione non oscillante.

La generazione della traiettoria del *leader* resta inalterata rispetto al problema precedente, mentre per i *follower* è sufficiente includere dei vincoli di *collision avoidance* nel rispetto delle precedenze. Si è scelto di utilizzare vincoli di distanza fra gli agenti in modo da garantire

una distanza minima fra ogni coppia di robot. Con riferimento alle precedenze in Fig. 5.3, il *follower* F_1 sarà soggetto ad un vincolo di distanza rispetto al solo *leader*, mentre F_2 rispetto a F_1 ed al *leader*. Il problema di ottimizzazione per F_1 è quindi:

$$\begin{aligned} \min_{\bar{y}_{k+N}^{[F_1]}} & \gamma \left\| \bar{y}_{k+N}^{[F_1]} - \tilde{y}_{k+N-1}^{[F_1]} \right\|^2 + \left\| \bar{y}_{k+N}^{[F_1]} - y_{setpoint}^{[F_1]} \right\|_T^2 \\ & \left\| \bar{y}_{k+N}^{[F_1]} - \tilde{y}_{k+N-1}^{[L]} \right\|_2 \geq d_{min} \\ & \bar{y}_{k+N}^{[F_1]} - \tilde{y}_{k+N-1}^{[F_1]} \in \beta_\varepsilon(0) \end{aligned} \tag{5.5}$$

mentre per il secondo *follower*:

$$\begin{aligned} \min_{\bar{y}_{k+N}^{[F_2]}} & \gamma \left\| \bar{y}_{k+N}^{[F_2]} - \tilde{y}_{k+N-1}^{[F_2]} \right\|^2 + \left\| \bar{y}_{k+N}^{[F_2]} - y_{setpoint}^{[F_2]} \right\|_T^2 \\ & \left\| \bar{y}_{k+N}^{[F_2]} - \tilde{y}_{k+N-1}^{[L]} \right\|_2 \geq d_{min} \\ & \left\| \bar{y}_{k+N}^{[F_2]} - \tilde{y}_{k+N-1}^{[F_1]} \right\|_2 \geq d_{min} \\ & \bar{y}_{k+N}^{[F_2]} - \tilde{y}_{k+N-1}^{[F_2]} \in \beta_\varepsilon(0) \end{aligned} \tag{5.6}$$

in cui d_{min} è la distanza minima, legata al set di incertezza $\mathcal{Y}$, definita nella sezione precedente.

5.3.3 Obstacle avoidance

Si vuole ora trattare il problema di controllare un agente affinchè eviti degli ostacoli, la cui posizione e dimensione sono note a priori. Poichè gli ostacoli posso essere visti come altri agenti, la cui posizione è nota e fissa, il problema dell'*obstacle avoidance* può essere risolto semplicemente imponendo un vincolo di distanza minima fra ciascun agente e ciascun ostacolo, analogamente a quanto fatto per risolvere il problema del *collision avoidance* nelle precedenti sezioni. Si consideri dunque un insieme di N_{obst} ostacoli definiti attraverso la posizione nel piano di lavoro e il rispettivo raggio. Ricordando la definizione della distanza minima d_{min} presentata nella Sezione 5.3, è quindi sufficiente aggiungere i seguenti vincoli

$$\left\| \bar{y}_{k+N}^{[i]} - y_j^O \right\|_2 \geq R_j^O + d_{min}, \quad \forall j = 1, \ldots, N_{obst} \tag{5.7}$$

al problema di ottimizzazione di ciascun agente.

Capitolo 6

Implementazione degli algoritmi di controllo

Nel presente capitolo verranno discussi gli aspetti implementativi degli algoritmi presentati nei Capitoli 4 e 5. Inizialmente si tratterà l'implementazione del problema di ottimizzazione legato al controllo *Tube-Based MPC* per poi passare al calcolo e la scelta di tutti i set arbitrari discussi.

6.1 MPC Robusto

Si consideri il controllo *Tube-Based MPC* applicato all'inseguimento di traiettorie delle uscite, analizzato nella Sezione 4.4.2. Il problema di ottimizzazione (4.51) relativo a questa soluzione di controllo, non può essere direttamente implementato in risolutori di problemi di ottimizzazione lineare quadratica, ma necessita di alcune elaborazioni, necessarie per esplicitare cifra di costo e vincolo espliciti rispetto ad un unico vettore di ottimizzazione. In particolare, il risolutore utilizzato sarà la funzione `quadprog` disponibile nell'ambiente MATLAB che richiede problemi formulati come segue:

$$\begin{cases} \min\limits_{\xi} J = \xi^\top S\xi + 2g^\top \xi \\ \qquad H\xi \leq L \end{cases} \tag{6.1}$$

in cui ξ è il vettore di ottimizzazione, mentre S e g sono noti.

6.1.1 Cifra di Costo

La cifra di costo quadratica che compare nella formulazione (4.51) dipende dai valori futuri dello stato nominale, che vengono predetti attraverso il modello del sistema nominale. Si deve quindi riformulare la cifra di merito in funzione di quantità note e di quantità rispetto alle quali si vuole ottimizzare. Nel caso considerato, all'istante k sono noti i riferimenti in tutto l'orizzonte di predizione ($\tilde{x}_{k:k+N}$ e $\tilde{u}_{k:k+N-1}$) e si vuole ottimizzare rispetto allo stato iniziale $\hat{x}_k$ e al vettore di ingresso nominale $\hat{u}_{k:k+N-1}$. In altre parole, si vuole riscrivere la cifra di merito

$$J = \sum_{h=0}^{N-1} \left\|\hat{x}_{k+h} - \tilde{x}_{k+h}\right\|_Q^2 + \left\|\hat{u}_{k+h} - \tilde{u}_{k+h}\right\|_R^2 + \left\|\hat{x}_{k+N} - \tilde{x}_{k+N}\right\|_P^2$$

nella forma (6.1).

Si consideri il termine $||\hat{x}_{k+h} - \tilde{x}_{k+h}||_Q^2$. Si vuole ricavare la predizione di $\hat{x}_{k+h} - \tilde{x}_{k+h}$ a partire dall'istante k. La dinamica di tale termine è:

$$\begin{cases} \hat{x}_{k+1} = A\hat{x}_k + B\hat{u}_k \\ \tilde{x}_{k+1} = A\tilde{x}_k + B\tilde{u}_k \end{cases} \Rightarrow (\hat{x}_{k+1} - \tilde{x}_{k+1}) = A(\hat{x}_k - \tilde{x}_k) + B(\hat{u}_k - \tilde{u}_k)$$

Iterando l'equazione appena trovata, si ricava la predizione cercata:

$$\begin{aligned} (\hat{x}_k - \tilde{x}_k) &= I(\hat{x}_k - \tilde{x}_k) \\ (\hat{x}_{k+1} - \tilde{x}_{k+1}) &= A(\hat{x}_k - \tilde{x}_k) + B(\hat{u}_k - \tilde{u}_k) \\ (\hat{x}_{k+2} - \tilde{x}_{k+2}) &= A^2(\hat{x}_k - \tilde{x}_k) + AB(\hat{u}_k - \tilde{u}_k) + B(\hat{u}_{k+1} - \tilde{u}_{k+1}) \\ &\vdots \\ (\hat{x}_{k+N} - \tilde{x}_{k+N}) &= A^N(\hat{x}_k - \tilde{x}_k) + A^{N-1}B(\hat{u}_k - \tilde{u}_k) + \ldots + B(\hat{u}_{k+N-1} - \tilde{u}_{k+N-1}) \end{aligned} \tag{6.2}$$

che può essere riscritta in forma matriciale come

$$\underbrace{\begin{bmatrix} \hat{x}_k - \tilde{x}_k \\ \hat{x}_{k+1} - \tilde{x}_{k+1} \\ \vdots \\ \hat{x}_{k+N} - \tilde{x}_{k+N} \end{bmatrix}}_{\hat{\mathcal{X}}_k - \tilde{\mathcal{X}}_k} = \underbrace{\begin{bmatrix} I \\ A \\ \vdots \\ A^N \end{bmatrix}}_{\mathcal{A}} (\hat{x}_k - \tilde{x}_k) + \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 \\ B & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & \cdots & B \end{bmatrix}}_{\mathcal{B}} \underbrace{\begin{bmatrix} \hat{u}_k - \tilde{u}_k \\ \hat{u}_{k+1} - \tilde{u}_{k+1} \\ \vdots \\ \hat{u}_{k+N-1} - \tilde{u}_{k+N-1} \end{bmatrix}}_{\hat{\mathcal{U}}_k - \tilde{\mathcal{U}}_k} \tag{6.3}$$

Definendo il vettore:

$$\xi_k = \begin{pmatrix} \hat{x}_k - \tilde{x}_k \\ \hat{\mathcal{U}}_k - \tilde{\mathcal{U}}_k \end{pmatrix} \tag{6.4}$$

si può scrivere la predizione come:

$$\hat{\mathcal{X}}_k - \tilde{\mathcal{X}}_k = \begin{bmatrix}\mathcal{A} & \mathcal{B}\end{bmatrix} \begin{bmatrix}\hat{x}_k - \tilde{x}_k \\ \hat{\mathcal{U}}_k - \tilde{\mathcal{U}}_k\end{bmatrix} = \begin{bmatrix}\mathcal{A} & \mathcal{B}\end{bmatrix} \xi_k \tag{6.5}$$

A questo punto la cifra di costo si riscrive, sfruttando le forme matriciali trovate, come:

$$J = \left\|\hat{\mathcal{X}}_k - \tilde{\mathcal{X}}_k\right\|_{\mathcal{Q}}^2 + \left\|\hat{\mathcal{U}}_k - \tilde{\mathcal{U}}_k\right\|_{\mathcal{R}}^2 \tag{6.6}$$

in cui $\mathcal{Q} = diag(Q, \ldots, Q, P)$ e $\mathcal{R} = diag(R, \ldots, R)$. In particolare $\mathcal{Q}$ è una matrice diagonale a blocchi con $N+1$ blocchi e $\mathcal{R}$ con N. È ora possibile esplicitare la cifra di merito rispetto al vettore ξ_k. Utilizzando (6.5) e (6.4) si ottiene:

$$\begin{aligned}
J &= \left\|\begin{bmatrix}\mathcal{A} & \mathcal{B}\end{bmatrix} \xi_k\right\|_{\mathcal{Q}}^2 + \left\|\begin{bmatrix}0 & I\end{bmatrix} \xi_k\right\|_{\mathcal{R}}^2 \\
&= \left[\begin{pmatrix}\mathcal{A} & \mathcal{B}\end{pmatrix} \xi_k\right]^\top \mathcal{Q} \left[\begin{pmatrix}\mathcal{A} & \mathcal{B}\end{pmatrix} \xi_k\right] + \left[\begin{pmatrix}0 & I\end{pmatrix} \xi_k\right]^\top \mathcal{R} \left[\begin{pmatrix}0 & I\end{pmatrix} \xi_k\right] \\
&= \left[\begin{pmatrix}\mathcal{A} & \mathcal{B} \\ 0 & I\end{pmatrix} \xi_k\right]^\top \begin{bmatrix}\mathcal{Q} & 0 \\ 0 & \mathcal{R}\end{bmatrix} \left[\begin{pmatrix}\mathcal{A} & \mathcal{B} \\ 0 & I\end{pmatrix} \xi_k\right] \\
&= \left\|\left[\begin{pmatrix}\mathcal{A} & \mathcal{B} \\ 0 & I\end{pmatrix} \xi_k\right]\right\|_{\begin{bmatrix}\mathcal{Q} & 0 \\ 0 & \mathcal{R}\end{bmatrix}}^2 = \left\|\mathbb{A}\xi_k\right\|_{\mathbb{Q}}^2
\end{aligned}$$

Si noti inoltre che il vettore ξ_k è parzialmente noto poichè dipende dai riferimenti di stato e ingresso. In particolare si ha:

$$\xi_k = \begin{pmatrix}\hat{x}_k - \tilde{x}_k \\ \hat{\mathcal{U}}_k - \tilde{\mathcal{U}}_k\end{pmatrix} = \begin{pmatrix}\hat{x}_k \\ \hat{\mathcal{U}}_k\end{pmatrix} + \begin{pmatrix}\tilde{x}_k \\ \tilde{\mathcal{U}}_k\end{pmatrix} = \hat{\xi}_k + \tilde{\xi}_k$$

in cui $\tilde{\xi}_k$ è completamente definito all'istante k, mentre $\hat{\xi}_k$ è il vettore delle variabili di ottimizzazione. Trascurando momentaneamente i vincoli, il problema di ottimizzazione diventa semplicemente:

$$\min_{\hat{\xi}_k} \left\|\mathbb{A}(\hat{\xi}_k - \tilde{\xi}_k)\right\|_{\mathbb{Q}}^2 \tag{6.7}$$

Espandendo la norma si ottiene:

$$\begin{aligned}
J &= \left\|\mathbb{A}(\hat{\xi}_k - \tilde{\xi}_k)\right\|_{\mathbb{Q}}^2 = (\hat{\xi}_k - \tilde{\xi}_k)^\top \mathbb{A}^\top \mathbb{Q} \mathbb{A} (\hat{\xi}_k - \tilde{\xi}_k) \\
&= \hat{\xi}_k^\top \mathbb{A}^\top \mathbb{Q} \mathbb{A} \hat{\xi}_k - 2\tilde{\xi}_k^\top \mathbb{A}^\top \mathbb{Q} \mathbb{A} \hat{\xi}_k + \tilde{\xi}_k^\top \mathbb{A}^\top \mathbb{Q} \mathbb{A} \tilde{\xi}_k
\end{aligned}$$

Si noti che l'ultimo termine non dipende dal vettore di ottimizzazione $\hat{\xi}_k$ e non influisce quindi sul valore ottimo di tale vettore e può essere ignorato. Rispetto alla forma generale (6.1) si ha quindi:

$$\begin{cases} S = \mathbb{A}^\top \mathbb{Q} \mathbb{A} \\ g = -\mathbb{A}^\top \mathbb{Q}^\top \mathbb{A} \tilde{\xi}_k^\top \end{cases}$$

6.1.2 Formalizzazione dei vincoli

I vincoli da imporre nell'ottimizzazione devono essere espressi rispetto al vettore $\hat{\xi}_k$. Affinchè il problema rientri nella classe dei problemi di ottimizzazione lineare quadratica, per cui esistono efficienti algoritmi risolutivi, è necessario imporre vincoli lineari, cioè esprimibili attraverso insiemi caratterizzati da un sistema di disequazioni lineari, nella forma:

$$x \in X \text{ con } X = \{x : Hx \leq L\}$$

Grazie alla linearità del problema considerato, questo risultato si ottiene scegliendo i set arbitrari poliedrici. Infatti, tutti gli insiemi ottenuti successivamente sono risultato di operazioni lineari che non alterano le loro proprietà. Verranno ora considerati e riformulati opportunamente i vincoli (4.46), (4.47), (4.49) e (4.50).
Il primo considerato è (4.46). Tale vincolo dipende dalla scelta del set arbitrario $\mathbb{W}$, definito in (4.37), in cui il disturbo assume valori; scegliendo tale insieme poliedrico, il set $\mathcal{E}$ risulterà anch'esso tale poichè ottenuto mediante operazioni lineari, ed è quindi esprimibile nella forma:

$$\mathcal{E} = \{x : H_\varepsilon x \leq L_\varepsilon\}$$

Perciò il vincolo da imporre è:

$$H_\varepsilon(x_k - \hat{x}_k) \leq L_\varepsilon \;\Rightarrow\; H_\varepsilon x_k - H_\varepsilon \begin{bmatrix} I & 0 \end{bmatrix} \hat{\xi}_k \leq L_\varepsilon$$

Si noti che tale vincolo varia ad ogni instante di campionamento in funzione del valore misurato dello stato reale x_k. Di qui in avanti si manterrà la matrice H_ε costante, variando quindi il termine L_ε. Utilizzando questa convenzione si ottiene il vincolo:

$$-H_\varepsilon \begin{bmatrix} I & 0 \end{bmatrix} \hat{\xi}_k \leq L_\varepsilon - H_\varepsilon x_k \;\Rightarrow\; H_1 \hat{\xi}_k \leq L_{1,k} \tag{6.8}$$

Si consideri ora (4.47). Scegliendo l'insieme di ammissibilità $\mathbb{X}$, per lo stato reale poliedrico, si ottiene un set per lo stato nominale nella forma:

$$\hat{\mathbb{X}} = \{x : H_{\hat{x}} x \leq L_{\hat{x}}\}$$

Tale vincolo deve essere rispettato in tutto l'orizzonte di predizione e va quindi applicato alla predizione dello stato nominale. Analogamente a (6.2), iterando l'equazione di stato del sistema nominale per $N-1$ passi si ottiene:

$$\underbrace{\begin{bmatrix} \hat{x}_k \\ \hat{x}_{k+1} \\ \vdots \\ \hat{x}_{k+N-1} \end{bmatrix}}_{\hat{\mathcal{X}}_k} = \underbrace{\begin{bmatrix} I \\ A \\ \vdots \\ A^{N-1} \end{bmatrix}}_{\bar{\mathcal{A}}} \hat{x}_k + \underbrace{\begin{bmatrix} 0 & \cdots & 0 & 0 \\ B & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \\ A^{N-2}B & \cdots & B & 0 \end{bmatrix}}_{\bar{\mathcal{B}}} \underbrace{\begin{bmatrix} \hat{u}_k \\ \hat{u}_{k+1} \\ \vdots \\ \hat{u}_{k+N-1} \end{bmatrix}}_{\hat{\mathcal{U}}_k}$$

in cui, nella matrice $\bar{\mathcal{B}}$ si è aggiunta una colonna di zeri in modo che premoltiplichi il vettore completo di ingressi da ottimizzare. Poichè ogni elemento di $\hat{\bar{\mathcal{X}}}_k$ deve appartenere ad $\hat{\mathbb{X}}$, si può scrivere:

$$\underbrace{\begin{bmatrix} H_{\hat{\mathbb{X}}} & & \\ & \ddots & \\ & & H_{\hat{\mathbb{X}}} \end{bmatrix}}_{\bar{H}_{\hat{\mathbb{X}}}} \hat{\bar{\mathcal{X}}}_k \leq \underbrace{\begin{bmatrix} L_{\hat{\mathbb{X}}} \\ \vdots \\ L_{\hat{\mathbb{X}}} \end{bmatrix}}_{\bar{L}_{\hat{\mathbb{X}}}} \Rightarrow \bar{H}_{\hat{\mathbb{X}}} \Big(\bar{\mathcal{A}} \hat{x}_k + \bar{\mathcal{B}} \hat{\mathcal{U}}_k \Big) \leq \bar{L}_{\hat{\mathbb{X}}}$$

che espresso rispetto al vettore di ottimizzazione $\hat{\xi}_k$ diventa:

$$\bar{H}_{\hat{\mathbb{X}}} \Big(\bar{\mathcal{A}} \begin{bmatrix} I & 0 \end{bmatrix} + \bar{\mathcal{B}} \begin{bmatrix} 0 & I \end{bmatrix} \Big) \hat{\xi}_k \leq \bar{L}_{\hat{\mathbb{X}}} \;\Rightarrow\; H_2 \hat{\xi}_k \leq L_2 \tag{6.9}$$

La formalizzazione del vincolo (4.48) risulta invece più immediata, poichè si tratta di un vincolo imposto direttamente sul vettore di ottimizzazione. Scelto un set $\mathbb{U}$ poliedrico, l'insieme $\hat{\mathbb{U}}$ è esprimibile come:

$$\hat{\mathbb{U}} = \{x : H_{\hat{\mathbb{U}}} x \leq L_{\hat{\mathbb{U}}}\}$$

Ancora una volta, tale vincolo deve essere imposto su tutto l'orizzonte di predizione. Perciò si ha:

$$\begin{bmatrix} H_{\hat{\mathbb{U}}} & & \\ & \ddots & \\ & & H_{\hat{\mathbb{U}}} \end{bmatrix} \begin{bmatrix} \hat{u}_k \\ \vdots \\ \hat{u}_{k+N-1} \end{bmatrix} \leq \begin{bmatrix} L_{\hat{\mathbb{U}}} \\ \vdots \\ L_{\hat{\mathbb{U}}} \end{bmatrix} \;\Rightarrow\; \bar{H}_{\hat{\mathbb{U}}} \hat{\mathcal{U}}_k \leq \bar{L}_{\hat{\mathbb{U}}}$$

che espresso rispetto al vettore di ottimizzazione diventa

$$\bar{H}_{\hat{\mathbb{U}}} \begin{bmatrix} 0 & I \end{bmatrix} \hat{\xi}_k \leq \bar{L}_{\hat{\mathbb{U}}} \;\Rightarrow\; H_3 \hat{\xi}_k \leq L_3 \tag{6.10}$$

Il successivo vincolo è (4.49) che va, ancora una volta, imposto sullo stato nominale in tutto l'orizzonte predittivo. Scegliendo un set Δ^z esprimibile nella forma:

$$\Delta^z = \{y : H_z y \leq L_z\}$$

ed applicando il procedimento precedente, si ottiene

$$\begin{bmatrix} H_z C & & \\ & \ddots & \\ & & H_z C \end{bmatrix} \begin{bmatrix} \hat{x}_k - \tilde{x}_k \\ \vdots \\ \hat{x}_{k+N-1} - \tilde{x}_{k+N-1} \end{bmatrix} \leq \begin{bmatrix} L_z \\ \vdots \\ L_z \end{bmatrix}$$

in cui C è la trasformazione di uscita. Ricordando la definizione del vettore $\hat{\mathcal{X}}_k - \tilde{\mathcal{X}}_k$ in (6.3), si noti che nel vincolo appena descritto manca l'ultima componente $\hat{x}_{k+N} - \tilde{x}_{k+N}$, la cui presenza permetterebbe di scrivere facilmente il vincolo in funzione del vettore $\hat{\xi}_k$. A tal proposito includendo anche il vincolo (4.50), si ha:

$$\begin{bmatrix} H_z C & & & \\ & \ddots & & \\ & & H_z C & \\ & & & H_\varepsilon \end{bmatrix} \begin{bmatrix} \hat{x}_k - \tilde{x}_k \\ \vdots \\ \hat{x}_{k+N-1} - \tilde{x}_{k+N-1} \\ \hat{x}_{k+N} - \tilde{x}_{k+N} \end{bmatrix} \leq \begin{bmatrix} L_z \\ \vdots \\ L_z \\ L_\varepsilon \end{bmatrix} \Rightarrow \bar{H}_{z\varepsilon}\big(\hat{\mathcal{X}}_k - \tilde{\mathcal{X}}_k\big) \leq \bar{L}_{z\varepsilon}$$

Utilizzando la relazione (6.5), il vincolo, espresso in funzione del vettore di ottimizzazione, diventa:

$$\bar{H}_{z\varepsilon} \begin{bmatrix} \mathcal{A} & \mathcal{B} \end{bmatrix} \hat{\xi}_k \leq L_{z\varepsilon} + \bar{H}_{z\varepsilon} \begin{bmatrix} \mathcal{A} & \mathcal{B} \end{bmatrix} \tilde{\xi}_k \;\Rightarrow\; H_{45} \hat{\xi}_k \leq L_{45,k} \tag{6.11}$$

A questo punto è possibile unire tutti i vincoli in un'unica matrice che verrà poi utilizzata, in fase di implementazione, dal risolutore:

$$\begin{bmatrix} H_1 \\ H_2 \\ H_3 \\ H_{45} \end{bmatrix} \hat{\xi}_k \leq \begin{bmatrix} L_{1,k} \\ L_2 \\ L_3 \\ L_{45,k} \end{bmatrix} \;\Rightarrow\; \mathsf{H} \hat{\xi}_k \leq \mathsf{L}_k \tag{6.12}$$

6.1.3 Scelta dei parametri

Si vuole ora discutere la scelta dei parametri della legge di controllo robusta. In particolare, è necessario definire il guadagno K della legge (4.39) e le matrici di costo Q, R e P per lo stato e gli ingressi, relative al problema di ottimizzazione (4.51).
La scelta della matrice K deve essere tale che il sistema nominale in

anello chiuso sia asintoticamente stabile, cioè che la matrice $(A + BK)$ sia *Schur*. Fra le varie tecniche presenti in letteratura si è utilizzato il controllo lineare quadratico (LQ) coerentemente con l'analisi di stabilità del controllo predittivo discussa nella Sezione 4.2.1. In seguito a varie prove sperimentali si sono scelti i pesi Q e R, rispettivamente sullo stato e sull'ingresso, pari a:

$$Q = I_4 \qquad R = 20I_2 \tag{6.13}$$

che rappresentano un buon compromesso fra velocità di risposta del robot e moderazione dell'azione di controllo. Si ricorda infatti che più "grande" è R rispetto a Q, maggiore sarà la moderazione dell'azione di controllo poichè l'utilizzo degli ingressi è dispendioso in termini di funzione obiettivo. Attraverso la funzione `[K, P] = dlqr(A, B, Q, R)`, in cui A e B sono rispettivamente la matrice dinamica e quella d'ingresso definite in (2.6), si ottiene la matrice di guadagni cercata, oltre alla soluzione P dell'equazione algebrica di Riccati, utile per impostare il peso dello stato alla fine dell'orizzonte di predizione, come discusso nella citata sezione.

6.1.4 Scelta dei set arbitrari

Nella descrizione della soluzione di controllo *Tube-Based MPC* sono stati introdotti diversi set arbitrari, che verranno ora scelti e analizzati. Come visto, con l'obiettivo di ottenere un problema di ottimizzazione di complessità ridotta, soggetto a soli vincoli lineari, tali set vanno scelti nel dominio degli insiemi poliedrici.

- Nel sistema reale (4.37) relativo ad un agente, si è introdotto un disturbo $w_k \in \mathbb{W}$ fittizio e limitato, volto a rappresentare incertezze nel modello stesso oltre ad errori di attuazione e misura. Prove sperimentali hanno evidenziato errori nella misura sempre inferiori a 1cm e si è quindi scelto di utilizzare un insieme poliedrico del tipo:

  $$\mathbb{W} = \{(x_1, x_2) : -1 \leq x_1 \leq 1, -1 \leq x_2 \leq 1\}$$

  Noto tale set è possibile calcolare $\mathcal{E}$, come in (4.41), definendo quindi il vincolo (4.46).

- Si consideri ora il vincolo sullo stato (4.47). È inizialmente necessario definire la regione di ammissibilità $\mathbb{X}$ per le variabili di stato

dell'agente. Ricordando, da (2.6), che lo stato dell'agente consiste nella sua posizione e velocità lungo le due direzioni del piano di lavoro, ciò equivale ad imporre dei limiti sulla dimensione del piano di lavoro stesso e sulla velocità dell'agente. Una possibile scelta è quindi:

$$\mathbb{X} = \begin{cases} x_{min,1} \leq x_1 \leq x_{max,1} \\ x_{min,3} \leq x_3 \leq x_{max,3} \\ v_{min} \leq x_2, x_4 \leq v_{max} \end{cases}$$

in cui $x_{min,1}$ e $x_{max,1}$ indicano le dimensioni del piano di lavoro nella prima direzione e $x_{min,3}$ e $x_{max,3}$ nella seconda, mentre v_{min} e v_{max} sono le velocità minime e massime volute. Si noti che la scelta di questi vincoli può essere evitata poichè la posizione dell'agente può essere efficacemente limitata direttamente nel livello di generazione della traiettoria e la velocità è implicitamente legata alla dimensione della bolla $\beta_\varepsilon(0)$ utilizzata in tale problema. Di conseguenza, questo vincolo può essere visto come un'ulteriore robustezza rispetto a tali limitazioni. Nell'implementazione realizzata si è scelto di porre vincoli meno stringenti di quelli nei livelli superiori ed in particolare si è posto $v_{max} = -v_{min} = 10cm/s$. Scelto l'insieme $\mathbb{X}$ si calcola quindi $\hat{\mathbb{X}}$ come in (4.47).

- Analogamente, per il vincolo (4.48), si sceglie l'insieme di ammissibilità per le variabili di ingresso, che equivale e porre dei limiti all'accelerazione dell'agente. Una possibilità è:

$$\mathbb{U} = \{(a_x, a_y) : a_{min} \leq a_x, a_y \leq a_{max}\}$$

in cui a_{min} e a_{max} rappresentano i limiti desiderati. Nella successiva implementazione si è scelto di non applicare alcun vincolo di questo tipo. Si noti infatti che, anche in assenza di un vincolo esplicito, l'ingresso dell'agente, cioè la sua accelerazione, sarà comunque limitata grazie ai vincoli impliciti sulla velocità dovuti alla limitatezza della distanza fra due punti successivi della traiettoria di riferimento, come precedentemente discusso. Il calcolo dell'insieme $\hat{\mathbb{U}}$, in base a questa scelta, non è quindi necessario.

- Si consideri ora il vincolo (4.49). Si deve definire il set Δ^z che limita lo scostamento fra le uscite del sistema nominale e quelle relative alla traiettoria che lo stato dovrà inseguire. La scelta di

questo insieme presenta un *trade-off* fra lo scostamento e l'ammissibilità del problema: scegliendo un insieme troppo piccolo, si ottiene una ridotta discrepanza fra le uscite, ma allo stesso tempo il problema potrebbe non essere risolvibile poichè non è detto che l'agente sia effettivamente in grado di seguire la traiettoria di riferimento. Varie prove sperimentali hanno portato alla seguente scelta:

$$\Delta^z = \{(y_1, y_2) : -1 \leq y_1 \leq 1, -1 \leq y_2 \leq 1\}$$

accettando quindi uno scostamento di 1cm fra le uscite nominali e quelle di riferimento.

6.2 Generazione della traiettoria

I problemi di ottimizzazione presentati nella Sezione 5.3 presentano dei vincoli non lineari e non possono essere risolti utilizzando la funzione `quadprog`. Si è quindi optato per un risolutore di problemi di programmazione non lineare, che nell'ambiente MATLAB, è disponibile attraverso la funzione `fmincon`.

6.2.1 Scelta del vincolo di distanza fra due punti successivi

Si vuole ora analizzare e discutere la scelta del set $\beta_\varepsilon(0)$, relativo al vincolo:

$$\bar{y}_{k+N} - \tilde{y}_{k+N-1} \in \beta_\varepsilon(0) \tag{6.14}$$

presente in tutti i problemi di ottimizzazione relativi alla generazione della traiettoria considerati. Tale vincolo, come descritto nella Sezione 4.5, rappresenta un compromesso fra velocità e incertezza nell'effettiva traiettoria seguita dall'agente. L'insieme scelto deve avere la proprietà di non presentare direzioni preferenziali rispetto al problema di ottimizzazione in cui è posto. Si noti infatti che i problemi considerati generano un nuovo punto della traiettoria minimizzando la distanza rispetto all'obiettivo e perciò eventuali direzioni che permettono una maggiore riduzione di tale distanza saranno preferite. A titolo di esempio si consideri un insieme quadrato, di lato r, definito come:

$$\beta_\varepsilon(0) = \{(y_1, y_2) : -r \leq y_1 \leq r, -r \leq y_2 \leq r\}, \quad r > 0$$

In base a quando detto, le diagonali di tale set presentano delle direzioni preferenziali che porteranno quindi ad una traiettoria generalmente

oscillante. Il set ideale è quindi un cerchio di raggio r, che non è però poliedrico, ma può essere efficacemente approssimato con un poligono inscritto nella circonferenza, dotato di un numero di lati sufficientemente elevato.
Nella successiva implementazione si è scelto di utilizzare un poligono con 100 lati, inscritto in una circonferenza di raggio 1.5cm.

6.2.2 Formalizzazione del problema con obiettivo fisso

Si vuole ora formalizzare il problema di ottimizzazione (5.1), riformulando opportunamente la cifra di costo ed i vincoli.

Cifra di costo

Poichè la cifra di costo è quadratica è ancora possibile esprimerla nella forma (6.1). Trascurando l'apice $^{[F_1]}$, utilizzato per indicare lo specifico agente, si ha quindi:

$$\begin{aligned} J = & \gamma(\bar{y}_{k+N} - \tilde{y}_{k+N-1})^\top(\bar{y}_{k+N} - \tilde{y}_{k+N-1}) + \\ & + (\bar{y}_{k+N} - y_{setpoint})^\top T(\bar{y}_{k+N} - y_{setpoint}) + \delta^\top D\delta \\ = & \bar{y}_{k+N}^\top(\gamma I_2 + T)\bar{y}_{k+N} + \delta^\top D\delta - 2(\gamma\tilde{y}_{k+N-1}^\top - y_{setpont}^\top T^\top)\bar{y}_{k+N} \\ = & \begin{pmatrix} \bar{y}_{k+N}^\top & \delta^\top \end{pmatrix} \begin{pmatrix} I_2 & I_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \gamma I_2 & 0 & 0 \\ 0 & T & 0 \\ 0 & 0 & D \end{pmatrix} \begin{pmatrix} I_2 & 0 \\ I_2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \bar{y}_{k+N} \\ \delta \end{pmatrix} + \\ & + 2 \begin{pmatrix} -\gamma\tilde{y}_{k+N-1} - Ty_{setpoint} \\ 0 \end{pmatrix}^\top \begin{pmatrix} \bar{y}_{k+N} \\ \delta \end{pmatrix} \end{aligned} \tag{6.15}$$

Chiamando $\eta_a = \begin{pmatrix} \bar{y}_{k+N}^\top & \delta^\top \end{pmatrix}^\top$ il vettore di ottimizzazione, la cifra di merito diventa:

$$J = \eta_a^\top S\eta_a + 2g^\top\eta_a \tag{6.16}$$

in cui si è posto:

$$\begin{aligned} S &= \begin{pmatrix} I_2 & I_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \gamma I_2 & 0 & 0 \\ 0 & T & 0 \\ 0 & 0 & D \end{pmatrix} \begin{pmatrix} I_2 & 0 \\ I_2 & 0 \\ 0 & 1 \end{pmatrix} \\ g &= \begin{pmatrix} -\gamma\tilde{y}_{k+N-1} - Ty_{setpoint} \\ 0 \end{pmatrix} \end{aligned} \tag{6.17}$$

ottenendo quindi la forma (6.1) cercata.

Vincoli

Il problema considerato è soggetto a due vincoli. Il primo riguardo alla distanza fra due punti successivi della traiettoria ed il secondo relativo alla distanza fra due agenti:

1. La distanza fra due punti successivi è limitata dalla relazione:
$$\bar{y}_{k+N}^{[F_1]} - \tilde{y}_{k+N-1}^{[F_1]} \in \beta_\varepsilon(0)$$
in cui, come discusso, l'insieme $\beta_\varepsilon(0)$ è poliedrico e quindi esprimibile nella forma:
$$\beta_\varepsilon(0) = \{y : H_\beta y \leq L_\beta\}$$
Analogamente a quando svolto in (6.8), il vincolo può essere scritto rispetto al vettore di ottimizzazione η_a come segue:
$$\begin{aligned} H_\beta(\bar{y}_{t+N}^{[F_1]} - \tilde{y}_{t+N-1}^{[F_1]}) \leq L_\beta \;&\Rightarrow\; H_\beta(I_2 \quad 0)\eta_a \leq L_\beta + H_\beta \tilde{y}_{t+N-1}^{[F_1]} \\ &\Rightarrow\; H_a \eta_a \leq L_a \end{aligned} \tag{6.18}$$
2. Il vincolo dedicato al mantenimento della formazione è:
$$\left\| \bar{y}_{k+N}^{[F_1]} - \tilde{y}_{k+N-1}^{[L]} \right\|_2 = h_{1L} + \delta$$
che non permette una riformulazione matriciale poichè non lineare. Può però essere riscritto in termini del vettore di ottimizzazione η_a come segue:
$$\left\| \begin{bmatrix} I_2 & 0 \end{bmatrix} \eta_a - \tilde{y}_{k+N-1}^{[L]} \right\|_2 - h_{1L} + \begin{bmatrix} 0 & 1 \end{bmatrix} \eta_a = 0 \tag{6.19}$$
che permette una facile implementazione nel risolutore utilizzato.

Il problema di ottimizzazione (5.2), relativo al secondo *follower*, è del tutto analogo e non ne verranno quindi discussi gli aspetti implementativi.

6.2.3 Formalizzazione del problema con setpoint variabile

Si consideri ora la soluzione al problema del mantenimento di una formazione discusso nella Sezione 5.3.2, ed in particolare, si vuole formalizzare il problema di ottimizzazione (5.5).

Cifra di costo

La cifra di costo del problema (5.5) è uguale al problema (5.1) precedentemente trattato, a meno del termine δ relativo all'errore di formazione. Di conseguenza, il procedimento per la formalizzazione, analogo a quanto svolto in (6.17) porta, trascurando l'apice $^{[F_1]}$, al seguente risultato:

$$\begin{aligned} J &= \gamma(\bar{y}_{k+N} - \tilde{y}_{k+N-1})^\top(\bar{y}_{k+N} - \tilde{y}_{k+N-1}) + (\bar{y}_{k+N} - y_{setpoint})^\top T(\bar{y}_{k+N} - y_{setpoint}) \\ &= \bar{y}_{k+N}^\top(\gamma I_2 + T)\bar{y}_{k+N} - 2(\gamma\tilde{y}_{k+N-1}^\top - y_{setpont}^\top T^\top)\bar{y}_{k+N} \\ &= \bar{y}_{k+N}^\top \begin{pmatrix} I_2 & I_2 \end{pmatrix} \begin{pmatrix} \gamma I_2 & 0 \\ 0 & T \end{pmatrix} \begin{pmatrix} I_2 \\ I_2 \end{pmatrix} \bar{y}_{k+N} + 2\left(-\gamma\tilde{y}_{k+N-1} - T y_{setpoint}\right)^\top \bar{y}_{k+N} \end{aligned} \tag{6.20}$$

Che può essere riformulato come:

$$J = \bar{y}_{k+N}^\top S \bar{y}_{k+N} + 2g^\top \bar{y}_{k+N} \tag{6.21}$$

in cui si è posto:

$$\begin{aligned} S &= \begin{pmatrix} I_2 & I_2 \end{pmatrix} \begin{pmatrix} \gamma I_2 & 0 \\ 0 & T \end{pmatrix} \begin{pmatrix} I_2 \\ I_2 \end{pmatrix} \\ g &= -\gamma\tilde{y}_{k+N-1} - T y_{setpoint} \end{aligned} \tag{6.22}$$

ottenendo la forma (6.1), come voluto.

Vincoli

I vincoli a cui questo problema è soggetto, sono molto simili a quelli appena analizzati per la soluzione con obiettivo fisso. La loro formalizzazione non verrà quindi ulteriormente discussa, poichè immediata a partire dai calcoli precedentemente effettuati.

6.3 Generazione delle traiettorie di riferimento per lo stato e per l'ingresso

L'implementazione del secondo livello dell'architettura di controllo proposta, dedicato alla generazione delle traiettorie di riferimento per lo stato e per l'ingresso corrispondenti alla traiettoria di riferimento per le uscite generata al livello superiore, richiede solo il calcolo delle matrici

K_x e K_e definite nel sistema (4.25). Come anticipato, attraverso la riscrittura (4.27) della matrice di dinamica $\mathcal{F}$ è possibile determinare le matrici cercate attraverso efficienti algoritmi, purchè la coppia $(\bar{A}, \bar{B})$ sia osservabile.
Nel problema considerato, poichè la coppia $(\bar{A}, \bar{B})$ è osservabile, si è scelto di utilizzare il metodo dell'assegnamento degli autovalori, disponibile attraverso la funzione `place` nell'ambiente MATLAB. La scelta degli autovalori è arbitraria e si richiede solo che la dinamica del sistema risultante sia sufficientemente veloce se comparata con quella dell'agente. Si è scelto di porre tutti gli autovalori in $z = 0.3$ ottenendo, come presentato successivamente, soddisfacenti risultati. Si ricorda che l'assegnamento di tutti gli autovalori nella stessa posizione può portare alla non presenza di una soluzione, ma questo problema è facilmente risolvibile ponendo valori molto vicini a quello desiderato, purchè non coincidenti.

6.4 Inizializzazione degli algoritmi

Il problema di ottimizzazione (4.51), relativo al livello di controllo robusto, richiede i valori della traiettoria di riferimento per lo stato e per le uscite in tutto l'orizzonte di predizione. È quindi necessario inizializzare il sistema (4.25) in modo da disporre già all'istante iniziale della traiettoria richiesta dal livello sottostante. A tal proposito si supponga di partire da condizioni iniziali in cui gli agenti sono fermi in una posizione $y_0^{[i]}$ nota. Detto N l'orizzonte di predizione, una possibile soluzione, trascurando l'apice $^{[i]}$, è la seguente:

1. Si pone una traiettoria di riferimento per le uscite pari alla posizione iniziale per tutto l'orizzonte N:

$$\tilde{y}_{0:N} = (y_0, \ldots, y_0)$$

2. Si utilizza l'equazione di regime (4.28) per il sistema (4.25) per ottenere il corrispondente punto di regime della traiettoria di riferimento per lo stato e per gli ingressi:

$$\tilde{\chi}_0^{SS} = (I - \mathcal{F})^{-1} \mathcal{G} \tilde{y}_0$$

e si pone quindi il valore dello stato $\tilde{x}_0 = [I_4 \quad 0]\tilde{\chi}_0^{SS}$ e $\tilde{u}_0 = [K_x \quad K_e]\tilde{\chi}_0^{SS}$ per tutto l'orizzonte di predizione:

$$\tilde{x}_{0:N} = (\tilde{x}_0, \ldots, \tilde{x}_0)$$
$$\tilde{u}_{0:N-1} = (\tilde{u}_0, \ldots, \tilde{u}_0)$$

A questo punto le traiettorie di riferimento per tutte le grandezze sono note in tutto l'orizzonte di predizione ed è quindi possibile applicare gli algoritmi direttamente dall'istante iniziale.

Capitolo 7

Problema del Contenimento

Nel presente capitolo si analizzerà il problema del contenimento, o *Containment Control Problem*, per un sistema multi-agente. In particolare, esso verrà risolto mediante l'applicazione della tecnica di controllo robusto oggetto di questa tesi, che permette di disaccoppiare il problema della generazione della traiettoria da quello dell'inseguimento. Per questo motivo, nel prosieguo non verranno ulteriormente discusse le problematiche relative all'inseguimento robusto del riferimento, ma ci si focalizzerà solo sulla sua generazione.

7.1 Introduzione

Il problema del contenimento, trattato ad esempio in [5] e [4], consiste nel controllare un insieme di agenti, detti *follower*, affinchè la loro traiettoria resti sempre confinata all'interno della regione dello spazio definita dalle posizioni di un secondo insieme di agenti, detti *leader*. Intuitivamente, si vuole quindi che i *follower* stiano sempre contenuti "all'interno" di una regione definita dai *leader*, i quali si muovono liberamente nel piano di lavoro secondo obiettivi stabiliti. Inoltre, si considererà un'interazione solo locale fra gli agenti, intendendo che, in generale, ciascun agente non conosce la configurazione completa del sistema, ma solo quella relativa agli agenti definiti come vicini nel rispetto della rete di comunicazione presente. Per questo motivo, tale problema rientra nella classe dei problemi, detti "di consenso", che riguardano l'accordo fra più agenti rispetto a certi obiettivi attraverso lo scambio di informazioni locali. Per ulteriori informazioni riguardo a recenti studi su algoritmi per il consenso si rimanda, ad esempio,

a [24] o [25]. A questa problematica verrà unito anche il problema di *sensing*, nel quale i *follower* sono equipaggiati di sensori che devono essere disposti in modo efficiente all'interno della regione da misurare. Il problema risultate viene detto *mixed sensing-containment problem* e può essere scomposto nelle seguenti attività:

1. I *leader* si muovono verso posizioni prestabilite che delimitano una regione dello spazio che deve essere opportunamente occupata ed ispezionata da parte dei *follower*, garantendone il loro contenimento.

2. Quando i *leader* raggiungono i propri obiettivi, cioè i vertici della regione di interesse, restano fermi attendendo che i *follower* si dispongano nel modo ottimale ed eseguano le misure.

3. Nel caso in cui le regioni siano molteplici, si ripete il procedimento dal passo 1 finchè non sono terminate le regioni da visitare.

La soluzione proposta consiste in un algoritmo di controllo ibrido basato su un opportuno automa a stati finiti che, come sarà analizzato successivamente, garantisce l'ottenimento di quanto voluto. Questa tecnica è basata sul lavoro svolto in [10], a cui però si è sostituito il controllo di posizione utilizzato con quello robusto discusso in questo lavoro.

7.2 Definizioni

Verranno ora fornite alcune definizioni utili per la comprensione e la definizione del problema in questione. In particolare, si definiranno dei concetti della teoria dei grafi, utilizzati poi nella costruzione del grafo volto a rappresentare la rete di comunicazione fra i *leader* e i *follower* e, successivamente, si presenterà il concetto geometrico di *convex hull*, utilizzato per definire la regione di contenimento.

7.2.1 Teoria dei grafi

Si considerino n agenti indicizzati dagli elementi dell'insieme $\mathcal{N} = \{1, \ldots, n\}$. Il grafo di comunicazione può essere rappresentato da un grafo pesato $G = (\mathcal{N}, \mathcal{E}, w_G)$, in cui $\mathcal{E} \subseteq \{(i,j) : i,j \in \mathcal{N}, i \neq j\}$ è l'insieme degli archi e la funzione $w_G : \mathcal{E} \mapsto \mathbb{R}_{>0}$ associa ad ogni arco $(i,j) \in \mathcal{E}$ un peso strettamente positivo w_{ji}, il quale descrive il fatto

che possano essere trasmessi dati dal nodo i al nodo j.
Il grafo G è detto *bidirezionale* se $\forall(i,j) \in \mathcal{E} \Leftrightarrow (j,i) \in \mathcal{E}$, mentre è detto *non orientato* se vale inoltre $w_{ij} = w_{ji}$.
Un nodo $i \in \mathcal{N}$ è *connesso* ad un nodo $j \in \mathcal{N} \setminus \{i\}$ se esiste un cammino da i a j nel grafo, nel rispetto dell'orientamento degli archi. Analogamente, il grafo G si dice *connesso* se $\forall(i,j) \in \mathcal{N} \times \mathcal{N}$, i è connesso a j. Per quanto riguarda una coppia di nodi, se $(j,i) \in \mathcal{E}$, allora il nodo j è un *vicino* di i. Nel contesto di questo lavoro, ciò significa che l'agente j è in grado di trasmettere istantaneamente il proprio stato ad i. L'insieme di vicini del nodo $i \in \mathcal{N}$ viene indicato con $\mathcal{N}^{(i)}(G) = \{j \in \mathcal{N} : (j,i) \in \mathcal{E}\}$, la cui cardinalità è denotata con $|\mathcal{N}^{(i)}|$. Infine, se tutti i nodi sono vicini, e cioè se $\forall i,j \in \mathcal{N}, (i,j) \in \mathcal{E}$, il grafo si dice *completo*.

7.2.2 Rete di Comunicazione

Si consideri un sistema multi agente composto da n_l *leader* e n_f *follower*, con $n_l + n_f = n$. Corrispondentemente a tale partizione si definiscono gli insiemi $\mathcal{N}_l = \{l_i, i = 1, \ldots, n_l\}$ e $\mathcal{N}_f = \{f_i, i = 1, \ldots, n_f\}$ con $\mathcal{N} = \mathcal{N}_l \cup \mathcal{N}_f$ e $\mathcal{N}_l \cap \mathcal{N}_f = \emptyset$. Da ora in poi si utilizzerà la notazione l_i per indicare l'i-esimo *leader* e, analogamente, f_i per l'i-esimo *follower*.
Si vuole modellare la rete di comunicazione fra gli agenti mediante un grafo pesato e non orientato in cui gli agenti sono rappresentati dai nodi e lo scambio di informazioni mediante la presenza di un arco. A tal proposito di definiscono i seguenti tre grafi:

$$\begin{aligned} G_l &= (\mathcal{N}_l, \mathcal{E}_l, w_l), \quad \mathcal{E}_l \subseteq \{(i,j) : i,j \in \mathcal{N}_l, i \neq j\} \\ G_f &= (\mathcal{N}_f, \mathcal{E}_f, w_f), \quad \mathcal{E}_f \subseteq \{(i,j) : i,j \in \mathcal{N}_f, i \neq j\} \\ G_{lf} &= (\mathcal{N}, \mathcal{E}_{lf}, w_{lf}), \quad \mathcal{E}_{lf} \subseteq \{(i,j) : i \in \mathcal{N}_l, j \in \mathcal{N}_f\} \end{aligned}$$

in cui G_l descrive la comunicazione fra i *leader*, G_f quella fra i *follower* e G_{lf} lo scambio di informazioni fra il gruppo di *leader* e quello di *follower*.
Nelle successive analisi si considerano le seguenti assunzioni:

- G_l è un grafo completo. Ciò significa che è possibile lo scambio di informazioni fra tutti gli agenti in $\mathcal{N}_l$.
- Il grafo G_{lf} è tale per cui $\forall i \in \mathcal{N}_f$, $\exists j \in \mathcal{N}_l$ tale che $(j,i) \in \mathcal{E}_{lf}$. In altre parole, si richiede che ogni *follower* riceva lo stato di almeno un *leader*. Si noti che questa assunzione porta ad avere un insieme

di vicini del generico *follower* f_i, pari a $\mathcal{N}^{(i)}(G_{lf}) = \{l_j \in \mathcal{N}_l : (j,i) \in \mathcal{E}_{lf}\}$, la cui cardinalità è sempre maggiore uguale ad uno.

A partire da questa partizione del grafo originario G, si definisce la seguente *matrice del grafo* $K_{G_{lf}} \in \mathbb{R}^{|\mathcal{N}_f| \times |\mathcal{N}_l|}$, in cui ciascun elemento k_{ij} è definito come:

$$k_{ij} = \begin{cases} \dfrac{w_{ij}}{\sum_{h \in \mathcal{N}^{(i)}(G_{lf})} w_{ih}} & j \in \mathcal{N}^{(i)}(G_{lf}) \\ 0 & j \notin \mathcal{N}^{(i)}(G_{lf}) \end{cases}$$

dove si ricordi che $\mathcal{N}^{(i)}(G_{lf})$ denota l'insieme di vicini del nodo i nel rispetto del grafo G_{lf}. Si noti che la matrice risultate è stocastica a destra, o *row-stochastic*, cioè tale per cui la somma degli elementi di ogni riga è 1. Infatti, osservando che gli elementi corrispondenti a nodi non vicini al nodo i sono nulli, la somma dell'i-esima riga è:

$$k_i = \sum_{j \in \mathcal{N}_l} k_{ij} = \frac{\sum_{j \in \mathcal{N}_l} w_{ij}}{\sum_{h \in \mathcal{N}^{(i)}(G_{lf})} w_{ih}} = \frac{\sum_{h \in \mathcal{N}^{(i)}(G_{lf})} w_{ih}}{\sum_{h \in \mathcal{N}^{(i)}(G_{lf})} w_{ih}} = 1 \tag{7.1}$$

Questa proprietà verrà utilizzata nel seguito per garantire il contenimento dei *follower*.

7.2.3 Convex Hull

Si consideri un insieme C, definito in uno spazio vettoriale reale $V \subseteq \mathbb{R}^p$. Esso è detto *convesso* se $\forall x, y \in C$, il punto $w = (1-z)x + zy \in C$ per ogni $z \in [0,1]$. Si definisce quindi il *convex hull* di un set di punti $X = \{x_1, \ldots, x_q\}$ come l'insieme convesso minimo contenente tutti i punti di X e viene indicato con $\mathbf{Co}\{x_1, \ldots, x_q\}$ oppure con $\mathbf{Co}\{X\}$. In Fig. 7.1 è illustrato un esempio di *convex hull* per un insieme di 10 punti nel piano. L'operatore $\mathbf{Co}\{\cdot\}$ gode della proprietà di decrescente monotonia, cioè, dati due set di punti X_1 e X_2 tali che $X_1 \subseteq X_2$, si ottiene $\mathbf{Co}\{X_1\} \subseteq \mathbf{Co}\{X_2\}$. Questa caratteristica sarà utile nelle successive analisi.

7.2.4 Distanza da un insieme convesso

Si definisce ora una misura della distanza che verrà poi utilizzata per determinare il contenimento dei *follower*. Dato un insieme Ω convesso, si definisce la seguente misura:

$$d(x, \Omega) = \zeta_\Omega(x) \min_{y \in \delta\Omega} ||x - y||_2 \tag{7.2}$$

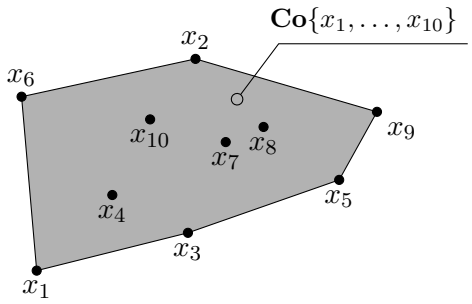


Figura 7.1: Esempio Convex Hull

in cui

$$\zeta_\Omega(x) = \begin{cases} -1 & x \in \Omega \\ 1 & x \notin \Omega \end{cases}$$

e $\delta\Omega$ indica il bordo di Ω. Si noti che tale misura risulta negativa se il punto x è all'interno di Ω e positiva altrimenti.

7.3 Formalizzazione del problema di contenimento

Si vuole ora formalizzare il problema di *containment* precedentemente descritto, utilizzando i concetti appena presentati. In particolare, si utilizzerà la definizione di *convex hull*, unita alla metrica introdotta, per formalizzare il concetto di *containment*, nel rispetto della rete di comunicazione presente fra gli agenti e modellata attraverso i grafi discussi nella precedente sezione.

7.3.1 Contenimento

Come accennato, il problema del contenimento consiste nel guidare gli agenti, detti *follower*, affinchè siano sempre contenuti nella regione dello spazio descritta dai *leader*. In particolare, tale regione è rappresentata dal *convex hull*, detto Ω_l, generato dalle posizioni di questi ultimi. Questa scelta, molto comune in letteratura, è dovuta alla semplicità di rappresentazione degli insiemi ottenuti, poichè essi risultano sempre poliedrici qualora il numero di *leader* sia finito.
Con l'obiettivo di formalizzare il concetto di contenimento, si consideri la seguente definizione, presente in [10]:

Definizione 2 (ε-containment)**.** *I follower si dicono ε-contenuti in Ω_l se per un dato $\varepsilon > 0$ e $\forall f_i \in \mathcal{N}_f$, vale $d(y^{[f_i]}, \Omega_l) < \varepsilon$.*

in cui $y^{[f_i]}$ indica la posizione del *follower* f_i e $d(\cdot,\cdot)$ è la misura di distanza definita in (7.2). Tale definizione permette di formulare matematicamente il concetto voluto, ma poichè non è garantito che il grafo G_{lf} sia completo, i *follower* potrebbero non essere a conoscenza dell'insieme Ω_l non permettendo quindi un'analisi locale riguardo il proprio contenimento. A tal proposito, per ogni *follower* $i \in \mathcal{N}_f$ si definisce il *convex hull* relativo ai soli *leader* in $\mathcal{N}^{(i)}(G_{lf})$, cioè quelli di cui conosce la posizione, indicato con $\Omega_l(i)$. Ora è possibile rilassare il concetto di ε-*containment* introducendo la seguente definizione:

Definizione 3 (ε-containment locale)**.** *Il follower f_i si dice ε-contenuto in $\Omega_l(i)$ se per un dato $\varepsilon > 0$ vale $d(y^{[f_i]}, \Omega_l(i)) < \varepsilon$.*

Ogni *follower* è quindi in grado di valutare autonomamente se è localmente contenuto e comunicare quest'informazione ai *leader*. Si noti inoltre, che l'ε-contenimento locale di tutti i *follower* implica quello in senso generale. Infatti, poichè $\mathcal{N}^{(i)}(G_{lf}) \subseteq \mathcal{N}_l$ e l'operatore $\mathbf{Co}\{\cdot\}$ gode della proprietà di monotonia decrescente, valgono i seguenti due risultati:

$$\begin{cases} \Omega_l(i) \subseteq \Omega_l \\ \bigcup\limits_{i \in \mathcal{N}_f} \Omega_l(i) = \Omega_l \end{cases}$$

che provano quanto detto.

7.3.2 Definizione del problema

Le definizioni introdotte permettono, come voluto, di esprimere formalmente il problema qui discusso.
A tal proposito, si considerino m regioni $\mathcal{D}_1, \ldots, \mathcal{D}_m$, convesse e poliedriche, ciascuna costituita da n_l vertici indicati con $z_j(\mathcal{D}_k)$, $j = 1, \ldots, n_l$. All'interno di una data regione D_k, si definiscono n_f punti, denotati con $s_j(\mathcal{D}_k)$, $j = 1, \ldots, n_f$, in cui i *follower* dovranno posizionarsi per effettuare le misure. Il problema di controllo consiste quindi:

I. Nel portare i *leader* in corrispondenza dei vertici $z_j(\mathcal{D}_k)$ delle regioni $\mathcal{D}_k$, $k = 1, \ldots, m$, garantendo allo stesso tempo che i *follower* restino all'interno del loro *convex hull* in ogni istante, secondo la definizione di ε-contenimento.

II. Successivamente al raggiungimento delle posizioni $z_j(\mathcal{D}_k)$, portare i *follower* nelle posizioni di misura $s_j(\mathcal{D}_k)$.

Più formalmente, i passi da compiere sono:

Algoritmo 1 Controllo ibrido

1. Indicando con k l'indice della regione corrente, si pone $k = 1$;
2. Si guidano i leader verso i vertici $z_j(\mathcal{D}_k)$ della regione corrente garantendo che i follower siano sempre ε-contenuti nel convex-hull generato dalle posizioni dei leader;
3. Si guidano i follower verso i rispettivi punti di misurazione $s_j(\mathcal{D}_k)$ della regione corrente, mentre i leader attendono che le operazioni di misura siano portate a termine;
4. Se le regioni non sono terminate, si passa alla regione successiva, ponendo $k = k + 1$ e tornando al punto 2.

La definizione completa del problema richiede inoltre una mappatura fra i *leader* e i vertici delle regioni fissate, in modo che sia noto in quale vertice si dovrà posizionare ciascun agente. Si definisce quindi una funzione

$$z^{[l_i]} = \mathcal{S}_l(l_i, \mathcal{D}_k)$$

che assegni un vertice ad ogni *leader* in base ad un opportuno criterio. Il vertice assegnato al *leader* l_i verrà indicato con $z^{[l_i]}$, compatibilmente con la simbologia utilizzata nel Capitolo 5. Analogamente, poichè anche i *follower* sono indistinguibili fra loro, è necessaria una seconda mappatura

$$s^{[f_i]} = \mathcal{S}_f(f_i, \mathcal{D}_k)$$

che associ a ciascun *follower* la relativa posizione di misura. La scelta di tali posizioni all'interno della regione, può essere effettuata mediante algoritmi di posizionamento ottimale dei sensori, volti a massimizzare l'area osservata dai sensori. Per dettagli riguardo tecniche di questo tipo si rimanda, ad esempio, a [33].

7.4 Architettura di Controllo

Si propone ora un'architettura di controllo ibrida, costituita da controllori locali di posizione e da un automa a stati finiti che governa le varie fasi della dinamica del problema in considerazione.
I controllori locali di posizione utilizzano la tecnica di controllo predittivo a tre livelli presentata nei capitoli precedenti che permette di disaccoppiare il problema di generazione della traiettoria da quello dell'inseguimento, assicurando però che l'agente si trovi in un intorno, limitato e noto a priori, della traiettoria di riferimento. Nella presente discussione non si considererà quindi il problema dell'inseguimento, ma ci si focalizzerà solo sulla generazione della traiettoria di riferimento. A tal proposito si formalizzerà il problema del *containment* attraverso il problema di ottimizzazione (4.52) che permette, fra l'altro, di aggiungere vincoli di *collision avoidance*, come precedentemente analizzato.

7.4.1 Controllo ibrido dei *leader*

La strategia di controllo utilizzata per muovere i *leader* nel piano di lavoro è basata su un automa a tre stati discreti chiamati GO, STOP e TARGET, illustrato in Fig. 7.2.

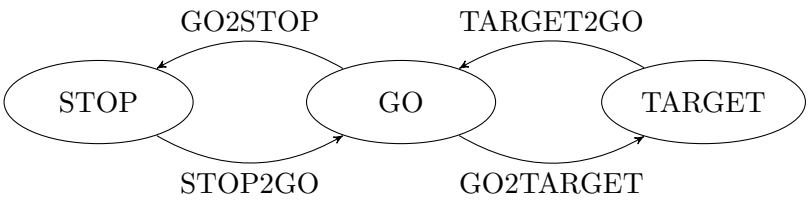


Figura 7.2: Automa *leader*

Si considerino ora gli stati discreti: per ciascuno di essi verrà chiarito il significato unito alla specifica strategia di generazione della traiettoria utilizzata.

- **GO**: I *leader* si muovono verso i vertici della regione da visitare. Si tratta quindi del semplice problema di raggiungere un punto nel piano già discusso nella Sezione 4.5. Per ogni *leader* l_i, si utilizza dunque il problema (4.52) ponendo il *setpoint*, chiamato $y_{setpoint}^{[l_i]}$ pari a $z^{[l_i]}$:

$$y_{setpoint}^{[l_i]} = z^{[l_i]}$$

- **STOP**: I *leader* si arrestano nella posizione attuale, anche se diversa da $z^{[l_i]}$. Come verrà chiarito nel seguito, il sistema si porta in questo stato quando almeno un *follower* non è localmente ε-contenuto. L'idea di questo stato è infatti quella di fermare i *leader* affinchè i *follower* siano in grado di ripristinare il contenimento. Nel rispetto della metodologia di controllo utilizzata, considerando il problema di ottimizzazione (4.52), è possibile ottenere questo risultato ponendo il *setpoint* pari alla posizione attuale:

  $$y_{setpoint}^{[l_i]} = y^{[l_i]}$$

  Una soluzione alternativa consiste nel rimuovere completamente il controllo di posizione impostando una velocità di movimento nulla, ma la prima tecnica è preferita poichè non si hanno commutazioni istantanee nella velocità ed è di più semplice implementazione anche per quanto riguarda la ripartenza, poichè sarà sufficiente importare il nuovo *setpoint* senza dover reinserire l'anello di posizione. Inoltre questo permette di ottenere il risultato desiderato senza dover interagire direttamente con l'agente, ma operando nel solo livello di generazione della traiettoria.

- **TARGET**: I *leader* si trovano in questo stato una volta raggiunto il vertice assegnato della regione $\mathcal{D}_k$. Analogamente allo stato STOP, i *leader* devono arrestarsi nella posizione attuale. In altre parole, il comportamento è del tutto analogo a quello in STOP. La differenza fra questi due stati è dovuta a differenti regole di transizione e ad un diverso comportamento dei *follower*, come discusso nel seguito.

7.4.2 Controllo ibrido dei *follower*

Analogamente a quanto presentato per i *leader*, anche la strategia di controllo dei *follower* è basata su un automa a tre stati, ora chiamati CONTAINMENT, FOLLOW e SENSING come illustrato in Fig. 7.3.
Si noti che l'automa dei *follower* è sincronizzato con quello dei *leader* in quando essi sono legati dalle stesse transizioni, analizzate nel seguito.
Si consideri ora il significato dei diversi stati:

- **FOLLOW**: questo stato è associato allo stato GO nell'automa dei *leader*. Di conseguenza, i *follower* devono essere controllati affinchè si mantengano all'interno del *convex hull* generato dai

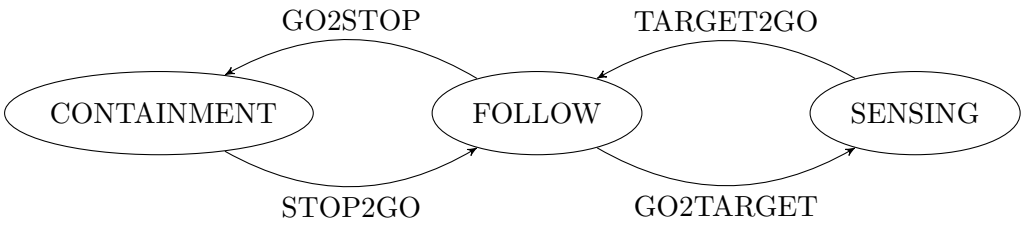


Figura 7.3: Automa *follower*

leader di cui conosce la posizione o, in altre parole, si vuole che essi siano localmente ε-contenuti. Per raggiungere questo obiettivo si utilizza, ancora una volta, il problema di generazione della traiettoria (4.52) in cui, per il *follower* f_i, si imposta il seguente *setpoint*, variabile ad ogni istante:

$$y_{setpoint}^{[f_i]} = \left(K_{G_{lf},i} \otimes I_2\right) \begin{bmatrix} \tilde{y}^{[l_1]} \\ \vdots \\ \tilde{y}^{[l_1]} \end{bmatrix} \tag{7.3}$$

in cui $K_{G_{lf},i}$ è l'i-esima riga della matrice $K_{G_{lf}}$, I_2 è la matrice identità di dimensione 2×2 e $\otimes$ denota il prodotto di Kronecker[1]. Con l'obiettivo di studiare le proprietà del *setpoint* risultante, si ricorda che la matrice $K_{G_{lf}}$ è stocastica a destra, cioè la somma degli elementi di ciascuna riga è sempre pari ad uno. È inoltre facile notare che tale proprietà è ereditata dalla matrice (di dimensione $2 \times 2n_l$) $K_{G_{lf},i} \otimes I_2$. Dunque il prodotto (7.3) rappresenta una combinazione convessa delle posizioni di riferimento dei *leader*, infatti:

$$y_{setpoint}^{[f_i]} = k_{i1}\tilde{y}^{[l_1]} + \cdots + k_{in_l}\tilde{y}^{[l_{n_l}]}$$

in cui vale (7.1). Ciò significa che il *setpoint* calcolato si trova certamente all'interno del *convex hull* generato dai *setpoint* delle posizioni dei *leader*. Inoltre, dalla definizione di tale matrice, si ha che l'elemento k_{ij}, corrispondente alla coppia (f_i, l_j), è nullo se il *follower* f_i non è connesso al *leader* l_j e non ne conosce quindi la posizione. Unendo le due proprietà appena discusse, si può concludere che la posizione di riferimento per f_i, calcolata utilizzando

[1] Si definisce il prodotto di Kronecker $C = A \otimes B$ di due matrici $A = \{a_{ij}\}$ e B come la matrice in cui l'elemento (blocco) ij-esimo è $a_{ij}B$

(7.3), si trova all'interno del *convex hull* relativo solo ai suoi *leader* vicini, precedentemente denotato con $\Omega_l(i)$. In conclusione, ogni *follower* viene guidato all'interno della regione $\Omega_l(i)$, nella quale è in grado di valutare autonomamente il proprio ε-contenimento locale.

- **CONTAINMENT**: poichè questo stato è associato allo stato STOP relativo ai *leader*, esso è attivo quando almeno un *follower* non è più localmente contenuto. Come detto prima, questo stato è utilizzato per ripristinare il contenimento perso, e quindi è sufficiente un riferimento analogo a quello relativo allo stato FOLLOW. Infatti, poichè i *leader* sono nello stato di STOP e quindi fermi, ciascun *follower* f_i sarà in grado di rientrare nel corrispondente *convex hull* locale $\Omega_l(i)$ ripristinando dunque il contenimento globale.

- **SENSING**: questo stato è associato allo stato TARGET nell'automa dei *leader*. Perciò questi ultimi si trovano in corrispondenza dei vertici della regione corrente assicurando che i *follower* siano contenuti in tale regione. A questo punto è quindi possibile avviare la fase in cui i *follower* vengono guidati verso le rispettive posizioni di *sensing*, indicate con $s^{[f_i]}$. Utilizzando sempre il problema di generazione della traiettoria (4.52), è quindi sufficiente porre
$$y_{setpoint}^{[f_i]} = s^{[f_i]}$$

7.4.3 Transizioni di stato

Si considerino ora le transizioni fra gli stati degli automi dei *leader* e dei *follower*, rispettivamente illustrati nelle Figg. 7.2 e 7.3. L'analisi degli stati di entrambi gli automi è stata eseguita interamente nel livello di generazione della traiettoria poichè si sono sempre utilizzati i riferimenti di posizione anzichè l'effettiva posizione degli agenti coinvolti. Questa scelta è dovuta alla volontà di non accoppiare i livelli dell'architettura di controllo predittivo robusto presentata nel Capitolo 4. D'altro canto, lo studio delle transizioni richiede l'utilizzo delle posizioni effettive. Ad esempio, per determinare quando i *leader* hanno raggiunto i rispettivi vertici delle regioni, è necessario utilizzare la posizione reale poichè, utilizzando quella di riferimento, non si può garantire che ciascun *leader* sia effettivamente nell'assegnato vertice. Per

questo motivo quindi si utilizzeranno le posizioni degli agenti, indicate con y.

- **GO2TARGET**: questa transizione avviene quando i *leader* raggiungono i vertici della regione corrente. Il test da eseguire è quindi:
$$y^{[l_i]} = z^{[l_i]}, \quad \forall l_i \in \mathcal{N}_l \tag{7.4}$$
- **TARGET2GO**: essa scatta quando i *follower* hanno raggiunto le posizioni di rilevazione. Infatti, poichè si è assunto che le misure siano istantanee, il raggiungimento di tali posizioni è sufficiente per ritenere completa l'operazione di *sensing* e si può quindi passare alla regione successiva. Il test da eseguire è:
$$y^{[f_i]} = s^{[f_i]}, \quad \forall f_i \in \mathcal{N}_f \tag{7.5}$$
- **GO2STOP**: questa transizione corrisponde alla perdita del contenimento locale di almeno un *follower*. Perciò la condizione da verificare è:
$$\exists f_i \in \mathcal{N}_f : d(y^{[f_i]}, \Omega(i)) \geq \varepsilon \tag{7.6}$$
- **STOP2GO**: essa avviene quando si recupera il contenimento di tutti i *follower*. Il test da effettuare è dunque:
$$d(y^{[f_i]}, \Omega(i)) < \varepsilon, \quad \forall f_i \in \mathcal{N}_f \tag{7.7}$$

Si osservi che in un contesto reale i test (7.4) e (7.5) possono essere eseguiti limitando la distanza fra la posizione dell'agente e il relativo obiettivo. Ad esempio, (7.4) può essere riformulato come:
$$\left\| y^{[l_i]} - z^{[l_i]} \right\|_2 \leq \mu, \quad \forall l_i \in \mathcal{N}_l \tag{7.8}$$
con $\mu > 0$.

7.5 Proprietà della strategia di controllo proposta

Si vogliono ora discutere ed analizzare le proprietà della soluzione di controllo ibrida proposta. Affinchè l'algoritmo sia adatto allo scopo per cui è stato realizzato, deve garantire il contenimento dei *follower* e la convergenza degli agenti verso le posizioni desiderate. Inoltre, è importante valutare che l'automa dei *leader* non resti indefinitamente nello stato di STOP poichè questo porterebbe all'arresto degli agenti senza possibilità di recupero. Tale proprietà è detta *liveness*.

7.5.1 Contenimento

Si consideri inizialmente il problema di garantire il contenimento dei *follower*. In [10] è stato dimostrato il seguente risultato, che è possibile considerare valido anche nel caso qui trattato:

Teorema 1 ([10])**.** *La strategia di controllo predittivo ibrido proposta garantisce che, se* $\exists \bar{k} \geq 0$ *tale che* $d(y^{[f_i]}, \Omega) \geq \varepsilon$ *per un qualche follower* $f_i \in \mathcal{N}_f$*, allora* $\exists T > 0$ *tale che al tempo* $\bar{k} + T$ *si abbia* $d(y^{[f_i]}, \Omega) < \varepsilon$*.*

In altre parole, se l'ε-contenimento viene perso all'istante $\bar{k}$, esso verrà recuperato in un tempo T. Per dimostrare tale risultato è necessario valutare le due condizioni in cui il contenimento può essere perso:

a) Si consideri il caso in cui i *leader* siano nello stato GO e i *follower* nel corrispondente FOLLOW. Si supponga che all'istante $\bar{k}$ venga perso l'ε-contenimento. In base alla definizione della transizione GO2STOP, essa scatterà ed i rispettivi automi si porteranno negli stati STOP e CONTAINMENT. A questo punto i *follower* inseguono un riferimento fisso, definito in (7.3) che, come precedentemente dimostrato, ha la proprietà di essere all'interno del *convex hull* definito dai *leader*. Grazie all'algoritmo di controllo robusto di posizione utilizzato per guidare i *follower*, si ha la certezza che essi arrivino nella desiderata posizione di riferimento in un tempo finito, ripristinando quindi il contenimento. Si ricorda però che la strategia di controllo utilizzata garantisce solo che l'agente sotto controllo stia in un intorno limitato della traiettoria di riferimento: questo implica che bisogna garantire che le posizioni inseguite dai *follower* siano all'interno del *convex hull* in misura sufficiente per cui si abbia la certezza che il *follower* vi sia contenuto indipendentemente dalla sua effettiva posizione. In altre parole, punti sul bordo dell'insieme o comunque troppo vicini agli estremi non permettono di garantire la proprietà cercata. Ricordando l'espressione (7.3), che definisce questi punti, si può concludere che è necessario selezionare accuratamente i pesi degli archi nel grafo G_{lf} posizionando i riferimenti nel rispetto di quanto appena discusso. Si ricorda però che non si richiede il contenimento in senso geometrico, ma nel senso dato nelle definizioni precedenti, cioè si considerano contenuti tutti i punti la cui distanza dal *convex hull* è inferiore ad ε. Nella Fig. 7.4 è illustrato un esempio di quanto appena esposto: la regione di incertezza intorno alla posizione del *follower* f_1 è completamente contenuta in

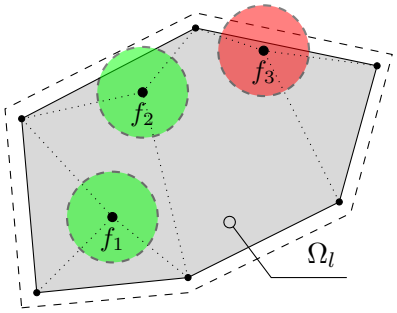


Figura 7.4: Esempio scelta pesi archi in $\mathcal{E}_{lf}$ rispetto alla dimensione ε

Ω_l e si ha quindi la garanzia che esso, in un tempo finito, recuperi il contenimento; per quanto riguarda il *follower* f_2, i punti di tale regione sono tutti a distanza inferiore di ε dal *convex hull*, ottenendo quindi la garanzia voluta; infine, per il *follower* f_3 non è possibile garantire nulla poichè la regione di incertezza contiene dei punti la cui distanza da Ω_l è superiore a ε. La scelta di ε deve quindi essere fatta considerando sia l'entità dell'incertezza nella posizione dei *follower* che i pesi attribuiti agli archi di $\mathcal{E}_{lf}$. Lo stesso discorso vale per l'effettiva posizione dei *leader* rispetto al vertice della regione in cui, nominalmente si dovrebbero trovare, ma, in questo caso, è possibile utilizzare la posizione del vertice stesso anzichè quella effettiva dell'agente, chiedendo implicitamente che il contenimento dei *follower* sia espresso rispetto alla posizione nominale e non a quella effettiva.

b) Si consideri ora il caso in cui i *leader* si trovino nello stato TARGET ed i *follower* in SENSING. Durante il movimento di quest'ultimi verso le posizioni di misura si potrebbe perdere il contenimento, ma assumendo che tali posizioni siano all'interno di Ω_l in maniera sufficiente rispetto alla dimensione della regione di incertezza, si può concludere che, in un tempo finito, il contenimento viene ripristinato.

7.5.2 Liveness

La proprietà di *liveness* richiede una permanenza non infinita nello stato di STOP per l'automa dei *leader*. Essa si ottiene direttamente dalla proprietà di contenimento appena dimostrata poichè quest'ultima garantisce che, in caso di perdita del contenimento, esso sia recuperato

in tempo finito. Analogamente, in caso di transizione nello stato STOP, la permanenza è limitata.

7.5.3 Convergenza

Le proprietà di convergenza degli agenti nelle posizioni di riferimento desiderate sono proprie dell'algoritmo di controllo robusto utilizzato e sono state dimostrate nel Capitolo 4. In particolare, come già detto, esse permettono di definire una regione di incertezza attorno alla traiettoria di riferimento, nella quale si ha la certezza di trovare l'agente.

Si noti infine che nessuna di queste proprietà può essere garantita nel caso si impongano vincoli nella generazione della traiettoria quali *obstacle* o *collision avoidance* poichè non si ha più la garanzia che l'agente converga nella posizione desiderata. Questo aspetto non è strettamente relativo al problema di contenimento trattato, ma è di carattere generale, come sarà analizzato nel successivo capitolo, dedicato proprio a trovare una soluzione a questo problema, mediante l'utilizzo di informazioni locali sull'ambiente circostante.

7.6 Mappatura delle posizioni di misura

Si propone ora una semplice metodologia per ottenere un'efficiente mappatura delle le posizioni di misura. Si noti che quando scatta la transizione GO2TARGET, i *follower* si trovano nella posizione generata con l'espressione (7.3), la quale non è legata in alcun modo alle posizioni di misura. Per questo motivo l'idea è di assegnare queste ultime in modo da minimizzare la distanza totale che i *follower* dovranno percorrere per raggiungere i rispettivi obiettivi. Tale problema di ottimizzazione può essere formulato come segue:

$$\begin{aligned} \min_{\alpha_{ij}} & \sum_{i \in \mathcal{N}_f} \sum_{j \in \mathcal{N}_f} \alpha_{ij} d_{ij} \\ & \alpha_{ij} \in \{0,1\}, \quad \forall i,j \in \mathcal{N}_f \\ & \sum_{j \in N_f} \alpha_{ij} = 1, \quad \forall i \in \mathcal{N}_f \end{aligned} \tag{7.9}$$

in cui d_{ij} è la distanza del *follower* f_i dalla j-esima posizione di misura al momento della transizione GO2TARGET. Si tratta di un problema

di programmazione lineare intera che consiste nel cercare la permutazione fra le coppie (*follower*, posizione di misura) che minimizza la distanza totale.
Si noti come lo stesso algoritmo possa essere utilizzato anche per eseguire la mappatura fra i *leader* ed i vertici della regione.

7.7 Prove in Simulazione

Nella presente sezione verranno presentati i risultati di una prova eseguita in ambiente MATLAB, in cui si è utilizzata la strategia di controllo fin qui discussa. Per le prove sperimentali, realizzate mediante l'utilizzo dell'apparato descritto nel Capitolo 2, si rimanda al Capitolo 9, in cui sono raccolti tutti i risultati sperimentali dei diversi algoritmi discussi in questa tesi. La simulazione che segue è stata eseguita utilizzando 4 *leader* e 3 *follower* connessi come rappresentato nel grafo in Fig. 7.5. Si noti che le due assunzioni presentate nella Sezione 7.2.2

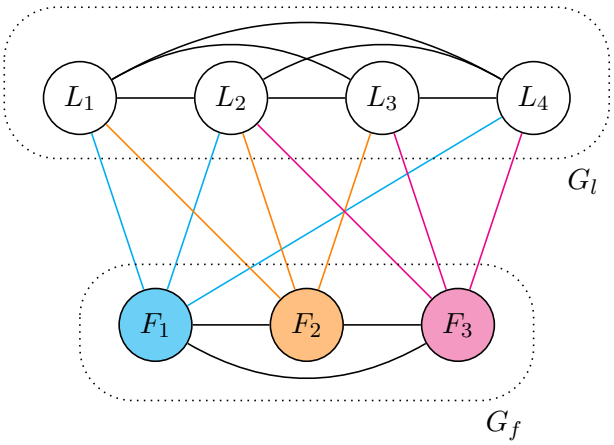


Figura 7.5: Simulazione - Grafo comunicazione *leader-follower*

sono rispettate poichè il grafo G_l è completo e ogni *follower* è connesso ad almeno un *leader*. Poichè tutti i pesi sono unitari, la matrice $K_{G_{lf}}$ risulta:

$$K_{G_{lf}} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \tag{7.10}$$

che, come dimostrato in precedenza, è stocastica a destra.
Si sono definite inoltre quattro regioni da visitare ed i rispettivi punti di misura, illustrate in Fig. 7.6. La scelta di quest'ultimi è avvenuta in

modo manuale, senza l'utilizzo di algoritmi dedicati al posizionamento ottimale dei sensori, in quanto non obiettivo di questo lavoro. La

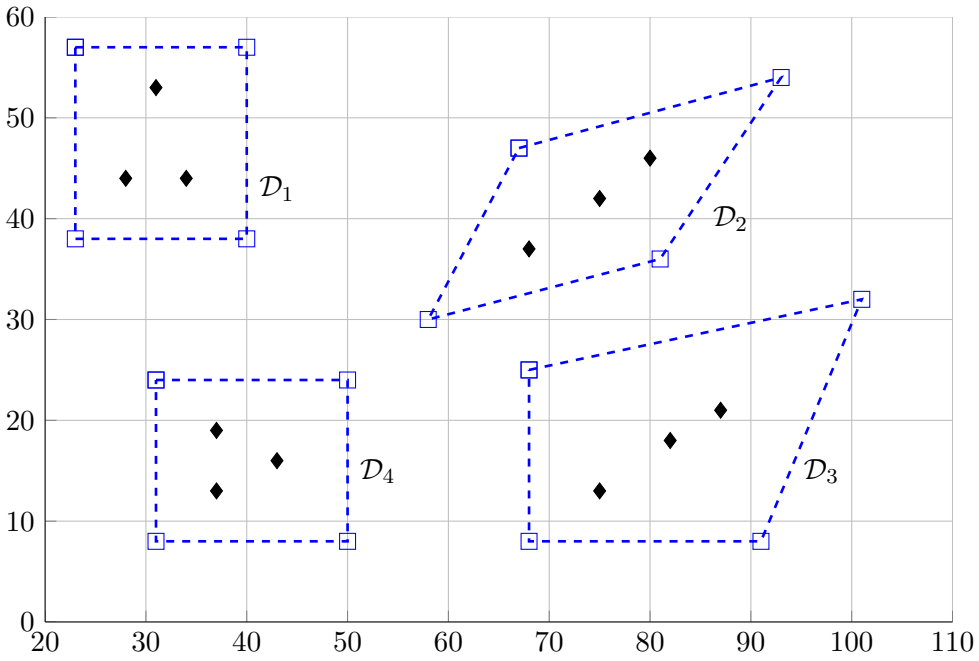


Figura 7.6: Simulazione - Regioni da visitare e relativi punti di misura

mappatura dei *leader* rispetto ai vertici delle regioni è stata anch'essa definita manualmente, associando i *leader* in senso orario, partendo dal vertice in alto a sinistra di ogni regione. Diversamente, la mappatura dei *follower* è stata eseguita attraverso il problema di ottimizzazione (7.9). Infine, per determinare quando gli agenti si trovano nelle rispettive posizioni di riferimento si è utilizzata la distanza fra la posizione effettiva e quella desiderata, accettando distanze entro $1cm$. Inoltre, si è posto $\varepsilon = 1$ nella valutazione del contenimento.
Nel problema di ottimizzazione dedicato alla generazione della traiettoria di ogni agente si è implementato un vincolo di *collision avoidance*, come descritto nella Sezione 5.3.3, mantenendo quindi i vari agenti ad una distanza minima fra di loro.

Posizione inziale e movimentazione verso la regione $\mathcal{D}_1$

Nella Fig. 7.7a è illustrata la configurazione iniziale scelta per gli agenti. Si noti che essa, anche se non richiesto, verifica il contenimento

iniziale dei *follower*. Perciò lo stato iniziale degli automi è rispettivamente GO e FOLLOW e dunque i *leader* si dirigono verso i vertici della regione $\mathcal{D}_1$. La successiva Fig. 7.7b rappresenta lo stato degli agenti nell'istante in cui scatta la transizione GO2TARGET, cioè quando i tutti *leader* hanno raggiunto i vertici. In questo istante viene risolto il problema di ottimizzazione (7.9) per assegnare a ciascun *follower* una posizione di misura. Gli automi passano quindi nello stato TARGET e SENSING e i *follower* si muovono verso le posizioni di *sensing* stabilite, come illustrato in Fig. 7.7c. Infine nella Fig. 7.7d è riportato un ingrandimento della configurazione degli agenti nello stesso istante di tempo. La regione $\mathcal{D}_1$ è quindi conclusa e si può passare alla regione successiva.

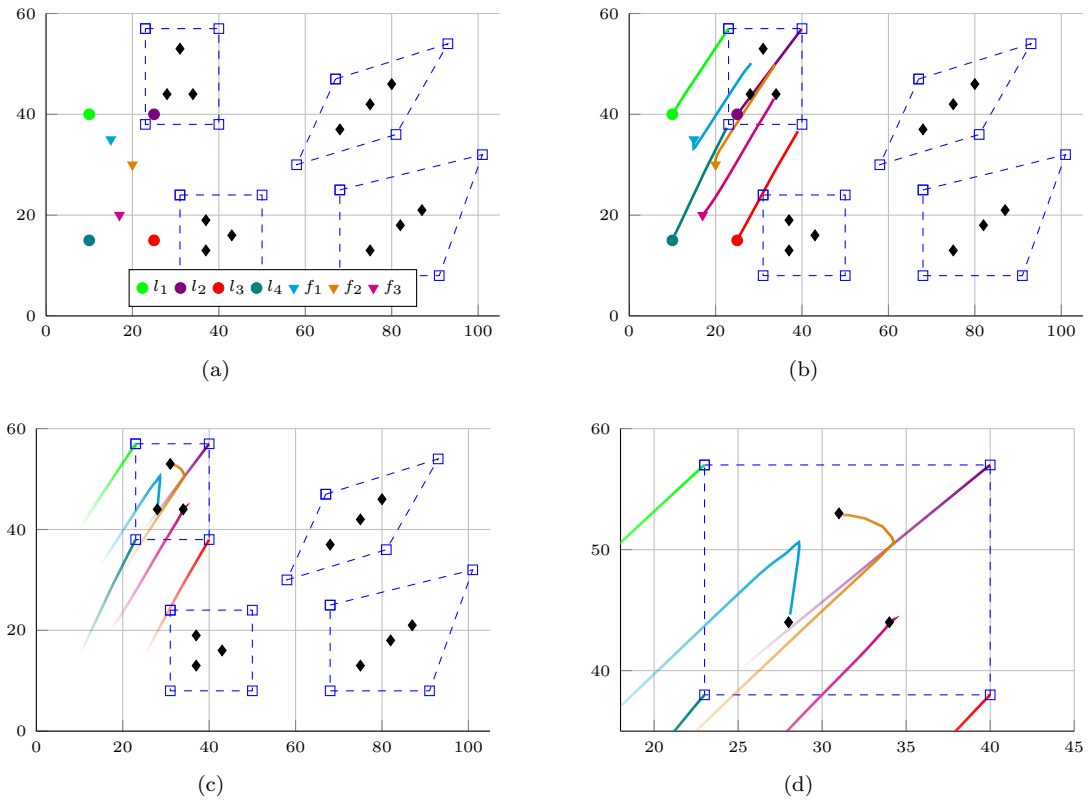


Figura 7.7: Simulazione - Configurazione inziale e traiettorie per la regione $\mathcal{D}_1$

Movimentazione verso le altre regioni

Nella Fig. 7.8 sono riportate le traiettorie percorse dagli agenti nella movimentazione dalla regione $\mathcal{D}_1$ alla regione $\mathcal{D}_2$. In particolare nella Fig. 7.8a è riportata la configurazione al momento della transizione GO2TARGET e in 7.8b quella relativa al completamento della regione corrente, cioè quando avviene TARGET2GO.
In modo del tutto analogo, nelle Figg. 7.9 e 7.10 sono riportate le traiettorie percorse dagli agenti fra ogni coppia di regioni successive. In particolare, analogamente alla Fig. 7.8, (a) rappresenta la configurazione al momento della transizione GO2TARGET e (b) al momento di TARGET2GO.

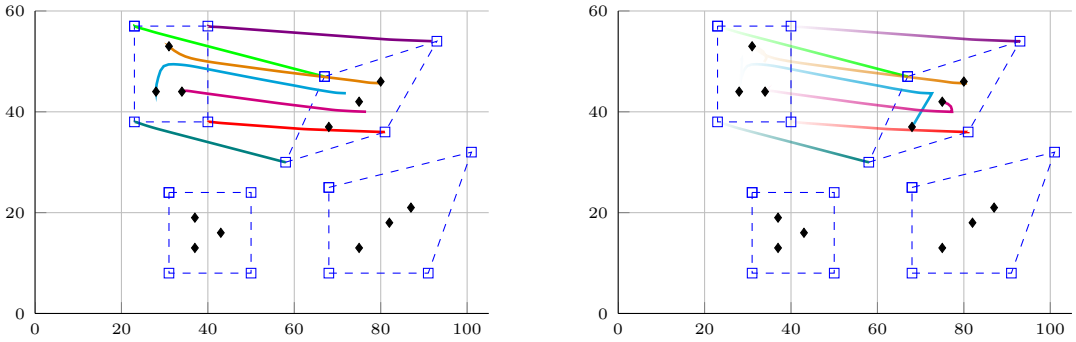


Figura 7.8: Simulazione - Traittorie da $\mathcal{D}_1$ a $\mathcal{D}_2$

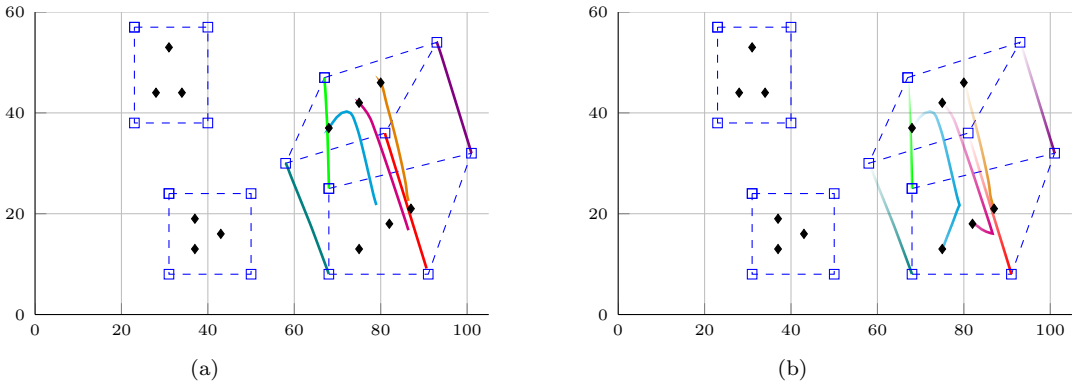


Figura 7.9: Simulazione - Traittorie da $\mathcal{D}_2$ a $\mathcal{D}_3$

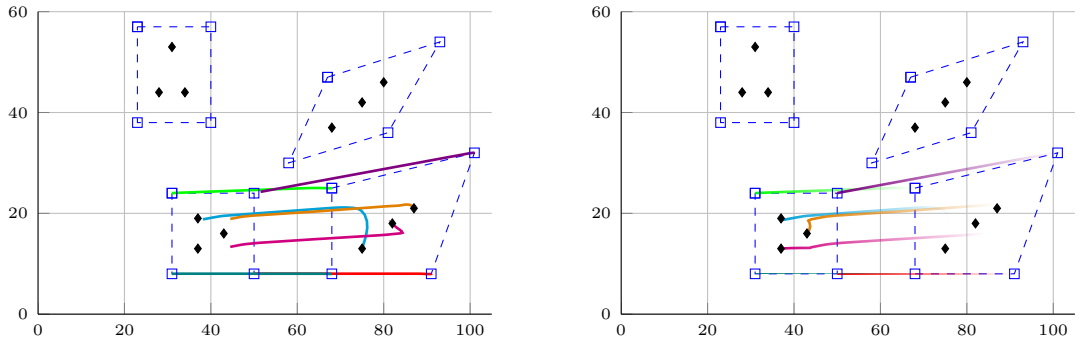


Figura 7.10: Simulazione - Traittorie da $\mathcal{D}_3$ a $\mathcal{D}_4$

7.8 Conclusioni

I risultati illustrati nelle figure precedenti portano a concludere che l'algoritmo di controllo ibrido presentato risolve in modo soddisfacente il problema del contenimento e rilevazione oggetto di questo capitolo. In particolare si notano i seguenti aspetti positivi:

- Il problema viene risolto solo nel livello di generazione della traiettoria senza considerare in alcun modo il problema dell'inseguimento dei riferimenti, rendendone quindi più facile la trattazione, l'implementazione e le relative analisi. Infatti, si ricorda che la generazione è disaccoppiata dalla dinamica dell'agente utilizzato, poichè quest'ultima è interamente gestita nei livelli di controllo inferiori dedicati, appunto, all'inseguimento.

- La tecnica di controllo robusto utilizzata per l'inseguimento permette di conoscere a priori un intorno della traiettoria di riferimento nel quale si è certi di trovare l'agente. Questo aspetto permette di scegliere in modo accurato il valore di ε utilizzato per determinare l'ε-contenimento dei *follower*, cosicchè quest'ultimo sia verificato indipendentemente dall'effettiva posizione dei *leader* e dei *follower* all'interno dei rispettivi intorni di incertezza.

D'altro canto l'algoritmo di controllo richiede di risolvere due problemi di ottimizzazione per agente ad ogni istante di tempo. Tale operazione potrebbe non essere localmente risolvibile in tempi accettabili, qualora le capacità computazionali distribuite siano ridotte.

Capitolo 8

Problema del Patrolling e della Navigazione in Ambiente Sconosciuto

Questo capitolo si pone l'obiettivo di studiare e risolvere il problema della navigazione di un agente in un ambiente ignoto, a partire dalla tecnica di controllo predittivo robusto oggetto di questa tesi. Nei capitoli precedenti si è analizzato il problema del raggiungimento di un obiettivo in un ambiente a priori completamente noto. In opposizione, si vuole ora trattare lo stesso problema nella caso in cui gli agenti siano equipaggiati di sensori che permettano di osservare solo una piccola porzione dell'ambiente. Ogni agente dovrà quindi prendere decisioni riguardo la traiettoria da inseguire sulla base di tali, limitate, informazioni. La soluzione di questo problema verrà elaborata in due fasi: inizialmente si considererà il solo problema del *patrolling*, o circumnavigazione di un ostacolo, per poi utilizzare tale risultato nel problema completo della navigazione in un ambiente sconosciuto. Per ulteriori dettagli e soluzioni già presenti in letteratura si rimanda, ad esempio, a [21] e [20].

8.1 Introduzione

Nei problemi di navigazione con *obstacle avoidance* analizzati nei capitoli precedenti non era possibile garantire che l'agente convergesse verso l'obiettivo. Ciò è dovuto all'eventuale presenza di vincoli che impediscono all'agente di muoversi verso il proprio obiettivo o, in termini

del problema di ottimizzazione, a causa di vincoli (per esempio non convessi) che non permettono di muoversi in una direzione che porti a una decrescita della cifra di costo. Si consideri, a titolo di esempio, la Fig. 8.1: la direzione opposta al gradiente, $-\nabla J$, della funzione di costo è in netto contrasto con i vincoli imposti al problema di ottimizzazione. Infatti, le direzioni ammissibili di spostamento dell'agente portano ad un aumento del valore ottimale della cifra di merito. Di conseguenza, la posizione in cui si trova l'agente è quella ottimale nel rispetto dei vincoli imposti. Una situazione di questo tipo è detta *deadlock*. Obiettivo di questo capitolo è quello superare questo problema

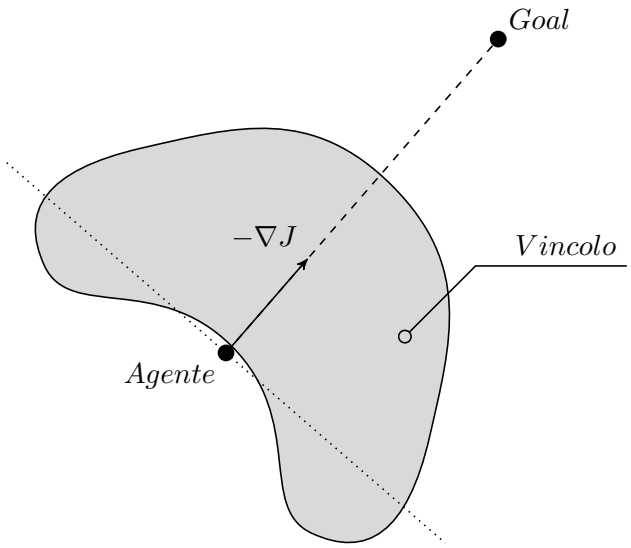


Figura 8.1: Esempio *deadlock* nella generazione vincolata della traiettoria

introducendo nell'agente differenti comportamenti in modo che esso sia in grado di distinguere quando il *goal* non è raggiungibile ed attuare di conseguenza strategie di movimentazione alternative, in base alle limitate conoscenze dell'ambiente.

Una possibile soluzione al problema della navigazione consiste nell'introdurre due comportamenti:

1. Movimentazione verso l'obiettivo, supponendo che la direzione del *goal* sia nota

2. *Patrolling* dell'ostacolo, sulla base delle informazioni locali disponibili

Le transizioni fra i due comportamenti sono legate alla presenza di un ostacolo nelle vicinanze dell'agente. Algoritmi di questo tipo, molto comuni in letteratura [18, 34], sono denominati *Bug's Algorithms* poichè replicano il comportamento naturale degli insetti, i quali si dirigono verso l'obiettivo fino al raggiungimento di un ostacolo, che viene quindi circumnavigato fino al suo superamento per poi procedere nuovamente verso il *goal*. Per una più approfondita analisi di queste tecniche si rimanda a [11].

L'idea è quindi quella di applicare una soluzione analoga sfruttando però la tecnica di controllo predittivo robusto presentata nei capitoli precedenti.

8.2 Patrolling

Il problema del *patrolling* consiste nel muovere un agente attorno al bordo di un ostacolo, in modo tale che sia mantenuta una certa distanza minima da esso. A titolo di esempio, si consideri la Fig. 8.2, in cui è rappresentato un ostacolo e la relativa traiettoria di circumnavigazione che l'agente deve inseguire. Sfruttando la tecnica di controllo predditti-

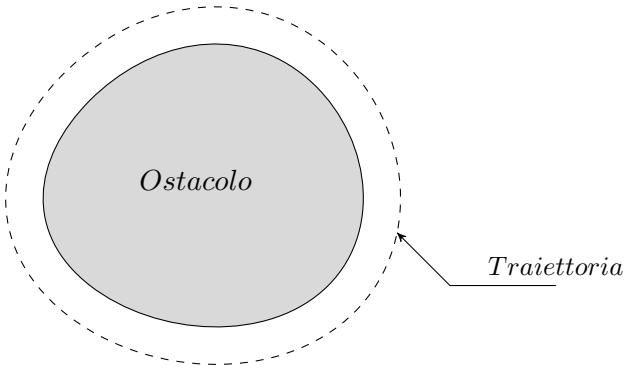


Figura 8.2: Esempio *patrolling*

vo robusto introdotta nel Capitolo 4, è possibile risolvere tale problema unicamente nel livello di generazione della traiettoria, avendo poi la certezza che la posizione reale dell'agente sia in un intorno di dimensione nota della traiettoria di riferimento. Come già visto nei problemi precedentemente affrontati, questo disaccoppiamento permette anche di fornire una soluzione generale, indipendente dalla dinamica dell'agente utilizzato poichè essa è gestita nei livelli inferiori di controllo robusto.

8.2.1 Rappresentazione dell'ambiente

Si introduce ora una rappresentazione dell'ambiente, che verrà utilizzata nel prosieguo del capitolo. Si supponga che l'agente sia equipaggiato con sensori che permettono di rilevare la presenza di un ostacolo in una regione limitata nell'intorno dell'agente stesso, definita attraverso i due parametri:

- d_{vis}: distanza massima entro la quale si rileva l'ostacolo
- α_{vis}: angolo di visuale dell'agente

rappresentata in Fig. 8.3. Si suppone inoltre che il risultato della rileva-

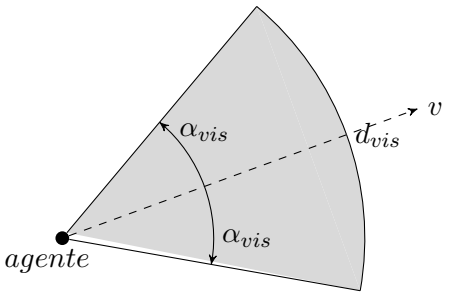


Figura 8.3: Definizione campo visivo agente

zione sia un certo numero di punti dei quali si conoscono le coordinate. Questo implica che i sensori devono fornire sia la direzione che la distanza del punto misurato in modo tale che, a partire dalla posizione dell'agente, sia possibile determinare la posizione dei punti visibili dell'ostacolo nel piano di lavoro. In Fig. 8.4 è illustrato un esempio del concetto appena introdotto, in cui l'agente conosce i punti dell'ostacolo evidenziati, nel rispetto del proprio campo visivo. A partire da tali punti sarà possibile costruire un livello di generazione della traiettoria per ottenere il risultato desiderato.

8.2.2 Generazione della traiettoria per il Patrolling

A partire dalla rappresentazione dell'ambiente introdotta, si propone ora un problema di ottimizzazione per la generazione della traiettoria adatto al problema in esame. Nel rispetto della simbologia utilizzata

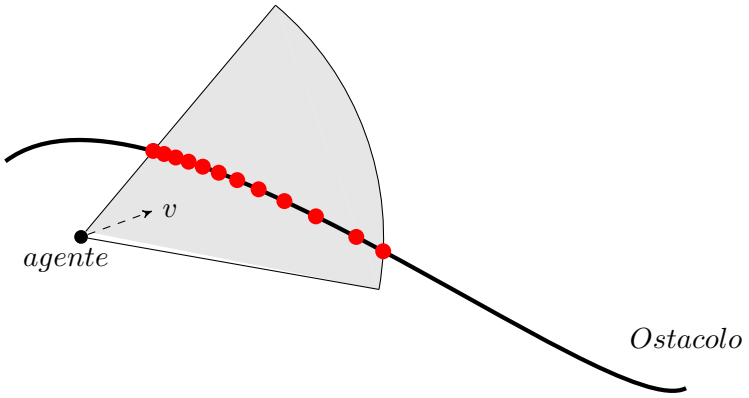


Figura 8.4: Esempio punti visibili di un ostacolo

nei precedenti capitoli, si indicherà con $\bar{y}_{k+N}$ il nuovo punto della traiettoria generato all'istante k a partire dal punto precedente, indicato con $\tilde{y}_{k+N-1}$. Inoltre, $\Upsilon_k = \{y_{k,h}^{[o]}, h = 1, \ldots, N_o\}$ rappresenta l'insieme dei punti visibili dell'ostacolo all'istante k, in cui, senza perdita di generalità, si suppone che siano successivi, a partire dal più vicino all'agente.

Come precedentemente introdotto, la traiettoria generata deve essere tale da mantenere l'agente ad una distanza minima, chiamata d_{obst}, dall'ostacolo, utilizzando solo le informazioni derivate dai punti visibili nell'insieme Υ_k. Questo risultato può essere ottenuto imponendo dei vincoli di distanza al problema di ottimizzazione, che assicurino che il nuovo punto $\bar{y}_{k+N}$ sia sufficientemente distante da ciascun punto visibile. In altre parole, si può pensare di porre i seguenti vincoli:

$$\left\| \bar{y}_{k+N} - y_{k,h}^{[o]} \right\|_2 \geq d_{obst}, \quad \forall h = 1, \ldots, N_o \tag{8.1}$$

Questa soluzione non è però adeguata al problema in questione. Infatti, anche se si ha la certezza che l'agente sia sufficientemente distante dall'ostacolo, non è possibile prevedere a che distanza esso si collochi realmente in quanto è libero di allontanarsi senza limiti. Per ovviare a tale problema è possibile sfruttare la tecnica presentata nella Sezione 5.3.1, che consiste nel considerare la distanza minima d_{obst} come una distanza di riferimento da mantenere dall'ostacolo e risolvere il problema di ottimizzazione minimizzando lo scostamento da tale riferimento.

In altre parole, si consideri il seguente problema:

$$\begin{aligned} \min_{\bar{y}_{k+N},\delta_h} & \sum_{h=1}^{N_o} \delta_h^2 \\ & \delta_h \geq 0, \quad \forall h = 1,\dots,N_o \\ & \left\| \bar{y}_{k+N} - y_{k,h}^{[o]} \right\|_2 = d_{obst} + \delta_h, \quad \forall h = 1,\dots,N_o \end{aligned} \tag{8.2}$$

in cui δ_h indica l'errore in distanza fra $\bar{y}_{k+N}$ e $y_{k,h}^{[o]}$, rispetto al riferimento d_{obst}. Tale problema di ottimizzazione genera una traiettoria, la cui distanza dall'ostacolo è vicina a d_{obst}, garantendo allo stesso tempo che essa non sia minore. Si ricorda inoltre che, per garantire il corretto funzionamento dei livelli di controllo robusto, è necessario imporre una distanza massima fra due punti successivi della traiettoria. Di conseguenza, è necessario aggiungere al problema (8.2), il seguente vincolo, introdotto e discusso nella Sezione 4.5:

$$\bar{y}_{k+N} - \tilde{y}_{k+N-1} \in \beta_\varepsilon(0)$$

in cui $\beta_\varepsilon(0)$ è una bolla di raggio ε centrata nell'origine. Con riferimento alla Fig. 8.5, il problema di ottimizzazione determina il nuovo punto $\bar{y}_{k+N}$, in modo tale che la lunghezza di ciascun segmento che collega i punti visibili a tale punto, in blu nella figura, sia più vicino possibile alla distanza desiderata d_{obst}. Il problema (8.2) non garantisce però che

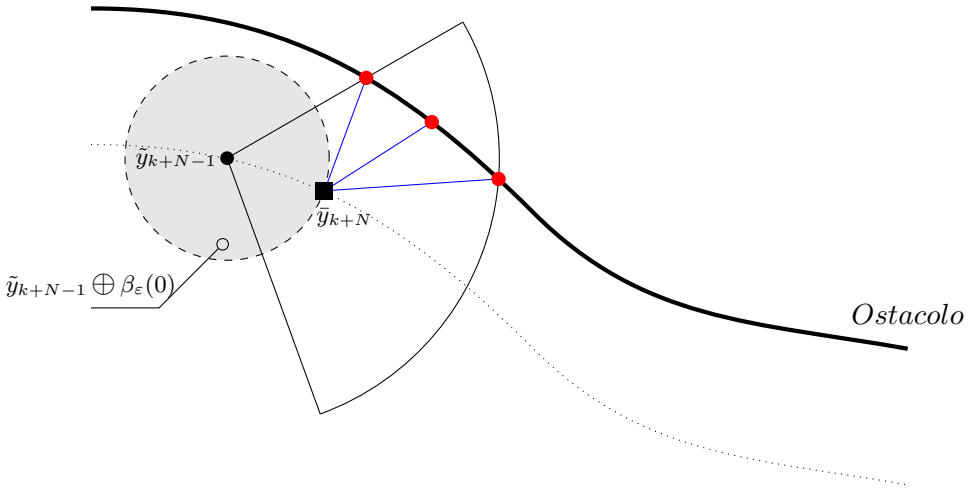


Figura 8.5: Esempio generazione della traiettoria per il patrolling

l'agente prosegua il suo moto attorno l'ostacolo. Infatti, nè nei vincoli, nè nella cifra di merito è presente un termine che imponga una certa distanza minima fra due punti successivi della traiettoria. Un'immediata soluzione a questo problema consiste nel porre un vincolo esplicito per soddisfare tale requisito:

$$\left\|\bar{y}_{k+N} - \tilde{y}_{k+N-1}\right\| \geq d_{pat}$$

in cui d_{pat} rappresenta la minima distanza desiderata. Alternativamente, una reinterpretazione di questo problema permette di ottenere un risultato analogo senza la necessità di introdurre nuovi vincoli. Infatti, per garantire che l'agente circumnavighi l'ostacolo è sufficiente fare in modo che esso si muova verso l'ultimo punto visibile, cosicchè, al passo successivo sia avanzato di una quantità sufficiente per vedere nuovi punti. In termini del problema di ottimizzazione presentato, questo risultato può essere ottenuto attribuendo pesi crescenti agli scostamenti δ_h. Trascurando i vincoli del problema, esso può quindi essere riformulato come segue:

$$\min_{\bar{y}_{k+N},\delta_h} \sum_{h=1}^{N_o} \lambda^h \delta_h^2, \quad \lambda > 1$$

Così facendo, il termine di errore δ_h relativo ai punti visibili più distanti ha un peso maggiore nella cifra di costo e verrà quindi minimizzato con maggiore priorità rispetto a quelli dei punti più vicini, forzando quindi l'agente a proseguire lungo il bordo dell'ostacolo.
In conclusione il problema di ottimizzazione completo è:

$$\begin{aligned} \min_{\bar{y}_{k+N},\delta_h} & \sum_{h=1}^{N_o} \lambda^h \delta_h^2, \quad \lambda > 1 \\ & \delta_h \geq 0, \ \ \forall h = 1, \ldots, N_o \\ & \left\|\bar{y}_{k+N} - y_{k,h}^{[o]}\right\|_2 = d_{obst} + \delta_h, \ \ \forall h = 1, \ldots, N_o \\ & \bar{y}_{k+N} - \tilde{y}_{k+N-1} \in \beta_\varepsilon(0) \end{aligned} \tag{8.3}$$

8.2.3 Gestione di un punto cieco

La strategia di generazione della traiettoria presentata si basa sull'assunzione che, ad ogni iterazione, l'insieme Υ_k non sia vuoto, o, in altre parole, che l'agente sia in grado di rilevare dei punti dell'ostacolo. Nel

caso ciò non sia verificato, il problema di ottimizzazione (8.3) non è ben posto, in quanto non è definita la funzione di costo. Tale configurazione si realizza quando l'agente si avvicina ad un punto cieco, come illustrato in Fig. 8.6. Una soluzione di carattere generale di questa problematica

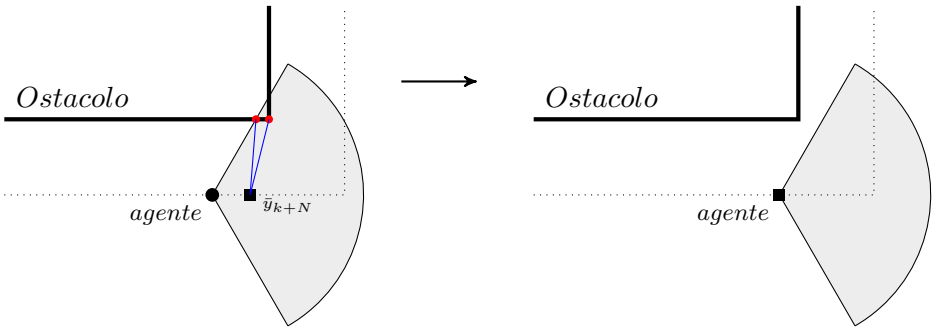


Figura 8.6: Esempio problema della generazione della traiettoria in un punto cieco

è ottenuta mediante l'applicazione di un controllo ibrido attraverso un automa a stati finiti. L'idea alla base di questa tecnica consiste nell'implementazione di due diversi comportamenti, uno dedicato al *patrolling* ed un secondo destinato alla gestione *ad hoc* del problema in esame. Con riferimento all'automa rappresentato in Fig. 8.7, si definiscono le

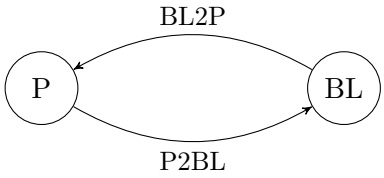


Figura 8.7: Automa per il problema del *patrolling*

seguenti transizioni:

- **P2BL**: rappresenta la commutazione fra il classico problema di *patrolling* e il caso particolare in cui non si hanno punti visibili. Dunque, la condizione di scatto può essere formulata come:

$$|\Upsilon_k| = 0$$

- **BL2P**: rappresenta la transizione opposta che deve quindi scattare quando il punto cieco è superato, o, in altre parole, quando si hanno un numero, $N_b > 0$, sufficiente di punti visibili per poter

riprendere la generazione della traiettoria mediante il problema (8.3). La condizione è quindi:

$$|\Upsilon_k| \geq N_b$$

A questo punto risulta immediato definire il significato dei singoli stati:

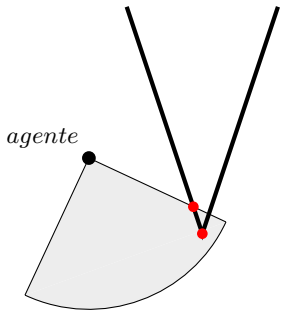
- **P**: quanto l'agente si trova in questo stato, detto di **P**atrolling, la generazione della traiettoria viene eseguita mediante il problema (8.3). La presenza di almeno un punto visibile sarà garantita dallo scatto delle transizioni.

- **BL**: l'obiettivo di questo stato è gestire la situazione in cui non si hanno punti visibili, cioè quando l'agente si trova in un punto cieco (**BL**ind). Supponendo che la transizione BL2P scatti all'istante k, l'insieme Υ_{k-1} dei punti visibili all'istante precedente, ha sicuramente cardinalità maggiore di uno, grazie alla struttura dell'automa presentato. In tal caso, per procedere nel pattugliamento del perimetro prescelto:

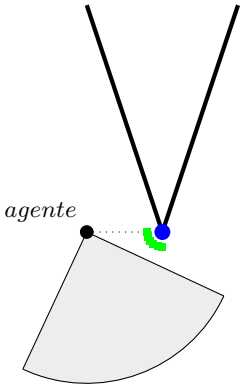
  1. Si considera l'ultimo punto visibile in Υ_{k-1}, indicato con $y_{old}^{[o]}$
  2. Si genera una successione di punti, $\bar{\Upsilon}_k$, rappresentante un arco di circonferenza centrato in $y_{old}^{[o]}$. L'angolo iniziale di tale arco è pari a quello del vettore congiungente il punto $y_{old}^{[o]}$ con la posizione attuale dell'agente, mentre la sua ampiezza è pari a α_{BL}.
  3. Si risolve il problema di ottimizzazione (8.3) per la generazione di un nuovo punto della traiettoria di riferimento, ponendo $\bar{\Upsilon}_k$ come insieme dei punti visibili
  4. Si ripete dal punto 2, fino allo scatto della transizione BL2P, cioè fino a quando l'agente non individua un numero sufficiente di punti visibili

  In Fig. 8.8 è illustrata la dinamica della soluzione presentata. In particolare, in Fig. 8.8a l'agente si sta avvicinando ad un punto cieco, raggiunto poi in Fig. 8.8b. A questo punto, scatta la transizione P2BL poichè l'agente non individua nessun punto e viene avviata la procedura descritta. Quando l'agente, in Fig. 8.8d, rileva un numero sufficiente di punti, scatta la transizione BL2P

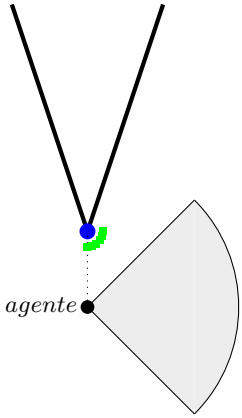
ritornando al classico problema di *patrolling*. Si noti come la generazione dell'arco rispetti la tecnica descritta al punto 2) del precedente algoritmo poichè l'angolo iniziale è legato alla posizione dell'agente.



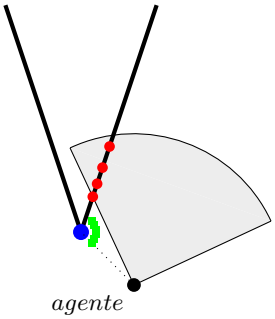
(a) Avvicinamento ad un punto cieco



(b) Scatto di PAT2BL, calcolo di $y_{old}^{[o]}$ e generazione dell'arco di punti fittizi con $\alpha_{BL} = \pi/2$



(c) Generazione dei punti fittizi in base alla posizione dell'agente



(d) Scatto transizione BL2PAT poichè si ha un sufficiente numero di punti visibili

Figura 8.8: Esempio di gestione di un punto cieco

8.2.4 Scelta della distanza minima

La scelta della distanza minima a cui l'agente deve mantenersi dall'ostacolo va eseguita nel rispetto dell'incertezza nella posizione effettiva

legata all'algoritmo di controllo robusto utilizzato. Come discusso nel Capitolo 4, la tecnica di controllo MPC *Tube-Based*, unita alla struttura gerarchica per la generazione ed inseguimento di una traiettoria per le uscite, permette di conoscere a priori una regione, definita attorno alla traiettoria di riferimento, la cui dimensione d_{min} è finita e nota, in cui si ha la certezza di trovare l'agente, indipendentemente dai disturbi agenti sul sistema, purchè essi rispettino le ipotesi presentate nella Sezione 4.3. Di conseguenza, la distanza minima, d_{obst}, per il problema del *patrolling* deve essere tale da garantire che, indipendentemente da dove l'agente effettivamente si trovi, quest'ultimo non collida con l'ostacolo adiacente o, in altre parole, deve valere:

$$d_{obst} > d_{min}$$

8.2.5 Prove in Simulazione

La tecnica introdotta per la soluzione del problema del *patrolling* è stata implementata in ambiente MATLAB con l'obiettivo di verficarne la validità. Si presentano ora i risultati di alcune prove, in cui si evidenziano le proprietà della soluzione proposta. In Appendice sono allegati gli algoritmi elaborati per la computazione dei punti visibili a partire dalla posizione e orientamento dell'agente. I parametri scelti nelle successive simulazioni sono:

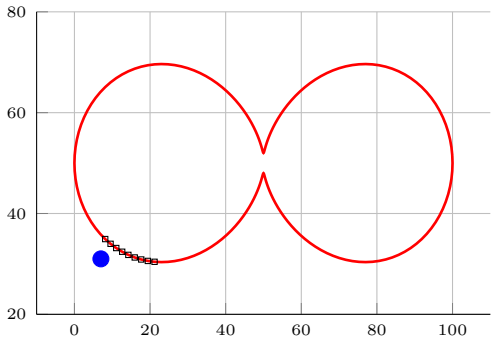
- Distanza a cui l'agente si deve mantenere dall'ostacolo: $d_{obst} = 3cm$
- Angolo di visuale: $\alpha_{vis} = \pi/2rad/s$
- Distanza massima di visuale: $d_{vis} = 15cm$
- Ampiezza arco di punti fittizi per il superamento di un punto cieco: $\alpha_{BL} = \pi/2rad/s$

Si noti che nelle prove in simulazione si assume che l'agente abbia raggio nullo. Tale ipotesi è ovviamente non valida nelle successive prove sperimentali.

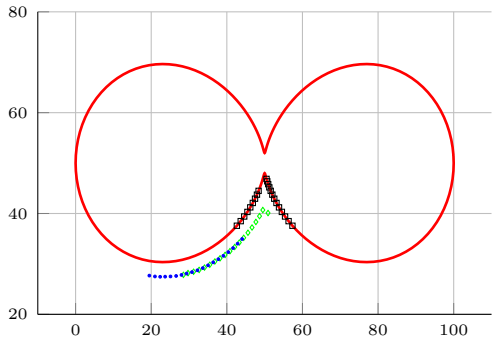
Patrolling di un ostacolo non convesso

La prima prova presentata consiste nel *patrolling* di un ostacolo non convesso illustrato in Fig. 8.9a dove è indicata, con un punto, la posizione iniziale dell'agente. Si noti che la scelta dello stato iniziale deve

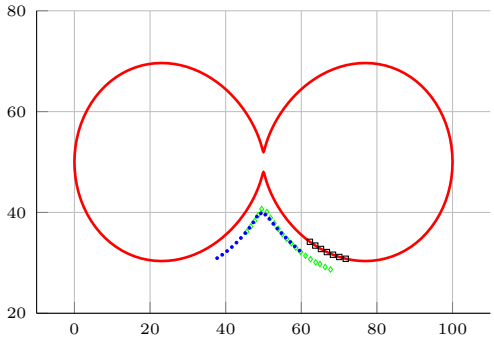
essere tale da garantire che l'insieme dei punti visibili non sia nullo. Infatti, in tale situazione, scatterebbe immediatamente la transizione P2BL senza però che sia definito l'insieme dei punti all'istante precedente, rendendo quindi privo di senso lo stato BL. Questo problema verrà risolto introducendo il concetto di *goal* che l'agente deve raggiungere, nella soluzione del problema completo della navigazione in un ambiente sconosciuto, discusso successivamente. Nella successiva Fig. 8.9b, l'agente si trova nelle vicinanze della concavità. Si ricorda che, grazie alla cifra di costo del problema di ottimizzazione (8.3), l'agente tende sempre a dirigersi verso il punto visibile più distante. Dunque, poichè nella configurazione rappresentata, esso vede già il fronte opposto dell'oggetto, la concavità verrà superata come illustrato in Fig. 8.9c.



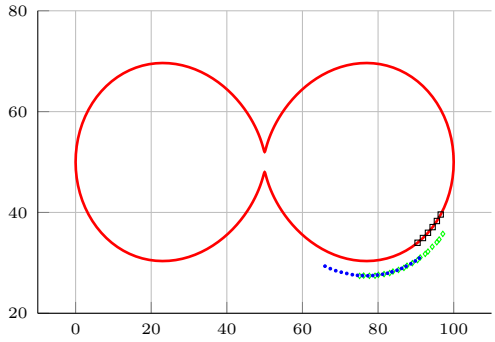
(a) Posizione e punti visibili iniziali



(b) Avvicinamento ad una cuspide e visibilità del fronte fronte opposto



(c) Superamento della cuspide

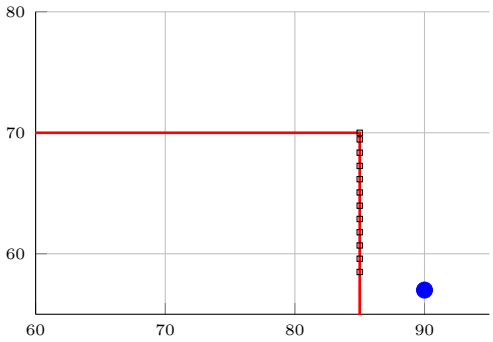


(d) Proseguimento del *patrolling* dell'ostacolo

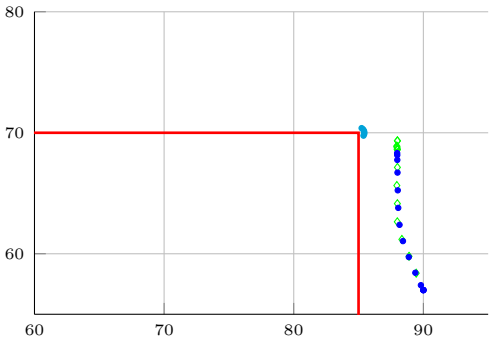
Figura 8.9: Patrolling di un ostacolo non convesso

Patrolling di un ostacolo con punto cieco

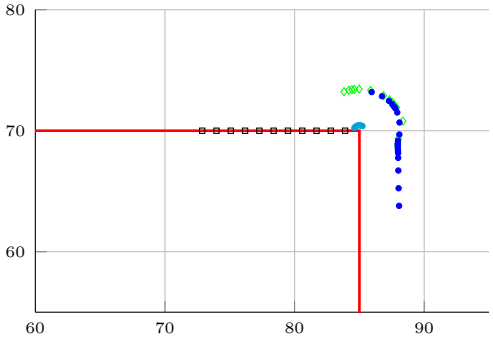
Si illustra ora la traiettoria generata nel caso in cui la struttura dell'ostacolo determini la presenza di un punto cieco. Si consideri la Fig. 8.10 in cui è presentata la traiettoria inseguita dall'agente, unita ai punti visualizzati in diversi istanti di tempo. In particolare, a partire dalla configurazione iniziale di Fig. 8.10a, l'agente si dirige verso l'angolo dell'ostacolo perdendone la visibilità e determinando quindi lo scatto della transizione P2BL, come illustrato in Fig. 8.10b. A questo punto viene generato l'arco fittizio e, in accordo con l'algoritmo introdotto, la traiettoria generata circumnaviga il punto cieco fino all'avvenuto superamento, Fig. 8.10c, che corrisponde alla recuperata visibilità. L'agente può ora proseguire con il classico *patrolling* dell'ostacolo in quando rileva sempre un numero sufficiente di punti.



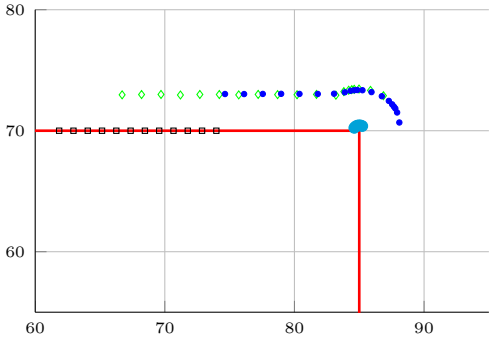
(a) Posizione e punti visibili iniziali



(b) Scatto della transizione PAT2BL e generazione dell'arco fittizio



(c) Scatto della transizione BL2PAT



(d) Proseguimento del *patrolling* dell'ostacolo

Figura 8.10: Patrolling di un ostacolo con punto cieco

8.3 Navigazione in Ambiente Sconosciuto

Il problema della navigazione in un ambiente sconosciuto consiste nel guidare un agente verso un punto noto dello spazio di lavoro, detto *goal*, sulla base di informazioni solo locali sull'ambiente circostante. Come introdotto inizialmente, tali informazioni sono legate alla visibilità di eventuali ostacoli prossimi all'agente che esso rileva attraverso opportuni sensori. Per rappresentare tali rilevazioni si utilizzerà la convenzione introdotta nella Sezione 8.2.1 per la soluzione del *patrolling* di un ostacolo cosicchè si possa risolvere il problema in esame a partire da quanto già elaborato in questo capitolo.

L'idea alla base della soluzione proposta è quella di introdurre diversi comportamenti nella generazione della traiettoria in base alla configurazione dell'ambiente e alla posizione dell'obiettivo. La modellizzazione di tali comportamenti sarà realizzata attraverso un automa a stati finiti, regolato da opportune transizioni che governano le commutazioni fra i diversi comportamenti dell'agente.

8.3.1 Automa per la Navigazione

Nella navigazione in un ambiente sconosciuto si vuole che l'agente abbia due macro-comportamenti:

1. Navigazione verso il *goal*, se non ci sono oggetti ad ostacolarne il movimento
2. *Patrolling* di un eventuale ostacolo che l'agente rileva nella via verso il goal

Di conseguenza, come anticipato, è possibile costruire un automa a partire da quello elaborato per il più semplice problema del *patrolling*, integrandolo con nuovi comportamenti volti al raggiungimento dell'obiettivo. A tal proposito si consideri l'automa in Fig. 8.11 che rappresenta, appunto, un'estensione di quello in Fig. 8.7 a cui è stato aggiunto uno stato, indicato con G, nel quale l'agente sarà diretto verso il *goal*. La generazione della traiettoria in questo stato può essere realizzata mediante il problema di ottimizzazione (4.52) poichè si tratta del semplice raggiungimento di un punto nello spazio di lavoro in un ambiente privo di ostacoli. Infatti, la presenza di questi ultimi viene gestita, come detto, mediante la loro circumnavigazione poichè è necessario utilizzare solo informazioni locali. Si considerino le transizioni

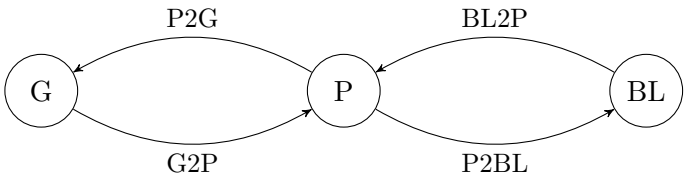


Figura 8.11: Automa a stati finiti per la Navigazione Autonoma

legate allo stato G:

- **P2G**: questa transizione porta l'agente nello stato G in cui, come detto, dovrà raggiungere il *goal*. Intuitivamente, quindi, essa deve scattare quando l'obiettivo è visibile nel rispetto delle sole informazioni locali relative agli ostacoli. La valutazione dell'eventuale visibilità del *goal* può essere eseguita sulla base di semplici considerazioni geometriche. Con riferimento alla Fig. 8.12, si può affermare che il goal è visibile se non è contenuto nel cono le cui generatrici sono le rette passanti per la posizione dell'agente e i punti visibili estremi di eventuali ostacoli. Si noti come la decisione sulla visibilità venga presa, come voluto, in base alle sole informazioni locali sull'ostacolo; infatti, se il *goal* non rientra in tale cono, non è detto che esso sia effettivamente raggiungibile poichè l'esatta conformazione dell'ostacolo non è nota. Ad ogni modo, grazie all'automa in Fig. 8.11, la presenza di ulteriori ostacoli riporterà l'agente nello stato di *patrolling* fino a quando il *goal* non tornerà visibile, evitando quindi collisioni.

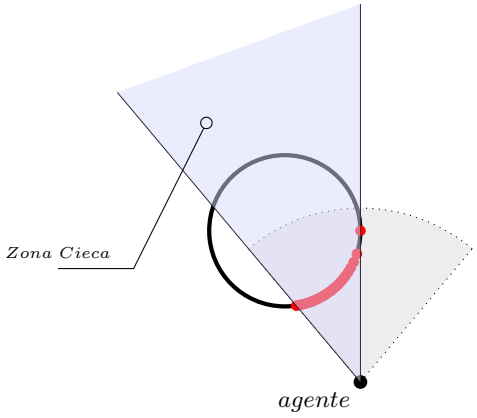


Figura 8.12: Rappresentazione della zona cieca per la determinazione della visibilità del *goal*

Considerando la natura del problema del *patrolling*, la soluzione presentata è valida solo nel caso in cui nell'ambiente ci sia un solo ostacolo o, più in generale, nel caso in cui l'agente non rilevi mai più impedimenti contemporaneamente. Infatti, in tali situazioni il problema del *patrolling* non è definito, in quanto non è chiaro quale ostacolo debba essere circumnavigato. A tal proposito è necessario definire delle regole che determinino quale ostacolo l'agente debba considerare. Ricordando che l'obiettivo ultimo è il raggiungimento del *goal*, una possibile scelta è quella di considerare solo l'ostacolo più vicino ad esso. Si noti che la tecnica presentata non garantisce in alcun modo che il *goal* sia raggiunto, poichè non si ha la certezza che l'agente sia in grado di trovare la corretta via per raggiungerlo. Ciò non deve stupire poichè, non vi è nessuna garanzia che, in mancanza di informazioni aggiuntive, la strada corretta verso il *goal* possa essere trovata persino da esseri umani. Ovviamente è possibile raffinare la tecnica di ricerca della via mediante opportuni algoritmi, qui non sviluppati, dedicati alla prova di vie alternative nel caso l'agente ritorni nello stesso punto. Dunque, l'idea di scegliere l'ostacolo più vicino al *goal* permette, in generale, di dirigere l'agente verso l'obiettivo con la speranza che possa trovare poi una via diretta per lo stesso.

La presenza di più ostacoli porta anche all'eventuale formazione di strettoie in cui l'agente non può passare soddisfacendo contemporaneamente i vincoli di distanza da ciascun ostacolo come, a titolo di esempio, è illustrato in Fig. 8.13. Come detto, il problema del *patrol-*

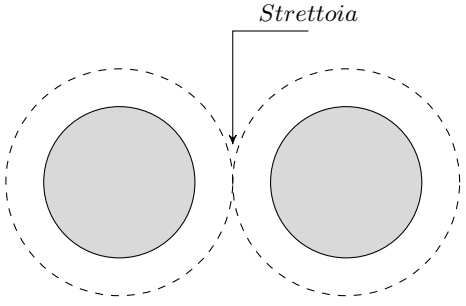


Figura 8.13: Esempio di strettoia

ling è definito rispetto a un solo ostacolo ed è quindi necessario scegliere quale circumnavigare, fra i due che generano la strettoia. Ad ogni modo, indipendentemente dalla scelta effettuata, non è possibile rispettare entrambi i vincoli di distanza. Di conseguenza, emerge la necessità di

trattare queste configurazioni mediante un'opportuna tecnica volta a superare la strettoia e riportare l'agente in una delle situazioni di cui si è già discussa una tecnica di soluzione. A tal proposito si aggiunge uno stato all'automa di Fig. 8.11, chiamato **B**ottle**N**eck il cui obiettivo è generare una traiettoria che riporti l'agente nello stato di *patrolling*. Per assegnare all'agente un valido comportamento da mantenere in questo stato, si consideri più in dettaglio il problema in esame, nel rispetto del comportamento istintivo di un individuo: quando una persona identifica la presenza di una strettoia tale da non riuscire a passare, cerca una via alternativa dirigendosi inizialmente verso il bordo di uno dei due ostacoli causa di tale configurazione, con l'idea di circumnavigare il problema. Si vuole dunque dotare l'agente di un comportamento analogo, che sia quindi in grado di rilevare una strettoia e trovare una conseguente via di fuga. Con questo fine, si consideri il nuovo automa di Fig. 8.14, in cui, rispetto al precedente, è stato introdotto il nuovo stato BN appena presentato. Trascurando gli archi e i nodi già

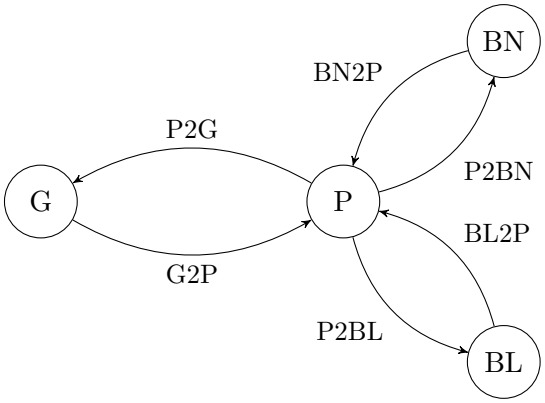


Figura 8.14: Automa completo per la navigazione in ambiente sconosciuto

introdotti, si considerino le transizioni associate allo stato BN:

- **P2BN**: questa transizione porta l'agente nello stato di *Bottleneck* e deve quindi scattare quando esso rileva la presenza di una strettoia tale da non riuscire a passare, soddisfacendo i vincoli di distanza rispetto a tutti gli ostacoli. Supponendo che l'agente sia in grado di distinguere i due ostacoli, sia $\Upsilon_k^{[o_1]}$ l'insieme dei punti visibili del primo ostacolo e $\Upsilon_k^{[o_2]}$ quelli del secondo. Dunque, la

condizione di scatto è:

$$d_s = \min \left\{ \left\| y_k^{[o_1]} - y_k^{[o_2]} \right\|, y_k^{[o_1]} \in \Upsilon_k^{[o_1]}, y_k^{[o_2]} \in \Upsilon_k^{[o_2]}, \right\} < 2d_{obst} \tag{8.4}$$

che corrisponde a verificare se esiste una coppia di punti visibili non appartenenti allo stesso ostacolo, la cui distanza è inferiore a $2d_{obst}$, dove, si ricorda che d_{obst} è la distanza che l'agente deve mantenere da ogni ostacolo.

- **BN2P**: questa transizione scatta quando l'agente ha superato efficacemente la strettoia avvicinandosi ad uno dei due ostacoli. Di conseguenza, tale transizione è analoga a BL2P.

A questo punto, successivamente alla descrizione delle transizioni che determinano la permanenza nello stato BN, si presenta l'algoritmo progettato per il superamento della strettoia. Per una maggiore comprensione dei passi che lo caratterizzano, si osservi la Fig. 8.15.

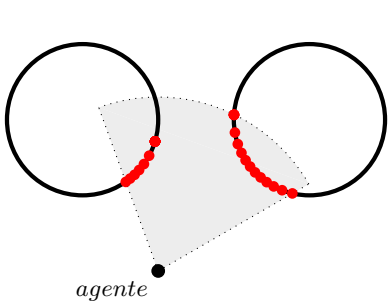
1. Si identifica la coppia di punti non appartenenti allo stesso ostacolo, la cui distanza è minima e si calcola corrispondente il punto medio della strettoia, indicato con P_{ms}. Con riferimento alla distanza d_s in (8.4), si ha quindi:

$$\begin{aligned} &P_{ms} = \frac{1}{2}(y_{min}^{[o_1]} + y_{min}^{[o_2]}), \\ &y_{min}^{[o_1]} \in \Upsilon_k^{[o_1]}, y_{min}^{[o_2]} \in \Upsilon_k^{[o_2]} : \left\| y_{min}^{[o_1]} - y_{min}^{[o_2]} \right\| = d_s \end{aligned} \tag{8.5}$$

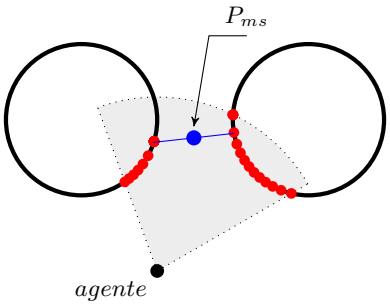
   in cui d_s è stata definita in (8.4).

2. Si identifica il punto visibile, chiamato P_{BN}, la cui somma delle distanze rispetto alla posizione dell'agente e al punto P_{ms} è maggiore.

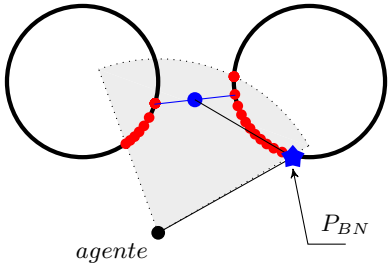
3. Si genera una trattoria che diriga l'agente verso il punto P_{BN}. Si osservi che tale generazione può essere ottenuta attraverso il problema di ottimizzazione (4.52) in quanto si tratta del semplice problema di raggiungimento di un punto nel piano di lavoro. Si noti che, grazie alla transizione BN2P, si ha la garanzia che l'agente non collida con l'ostacolo poichè, quando giunge in prossimità, l'agente viene riportato nello stato di *patrolling* mantenendosi alla distanza d_{obst} desiderata.



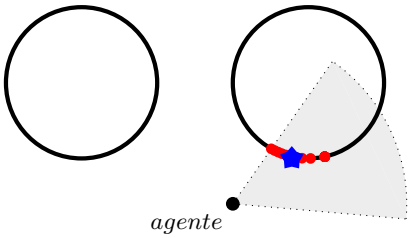
(a) Avvicinamento ad una strettoia e scatto di P2BL



(b) Calcolo del punto medio P_{ms} della strettoia



(c) Identificazione del punto più distante P_{BN}



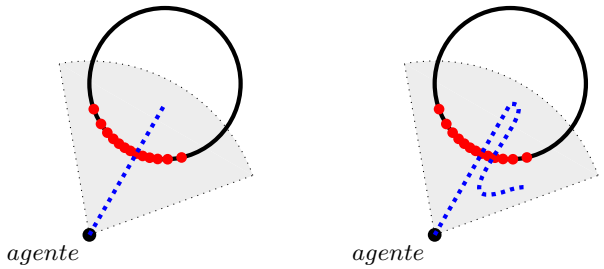
(d) Navigazione verso P_{BN}

Figura 8.15: Gestione di una strettoia

8.3.2 Predizione della traiettoria nella commutazione fra gli stati

A causa della natura ibrida del sistema di controllo proposto per la soluzione del problema della navigazione in ambiente sconosciuto, è necessario prestare attenzione alla combinazione di traiettorie predette attraverso differenti logiche implementate nei diversi stati. Infatti, proprio per la natura della tecnica di controllo predittivo, il livello di generazione della traiettoria elabora, ad ogni instante di tempo, un nuovo punto che l'agente raggiungerà nell'orizzonte di predizione. Questo porta ad un ritardo fra la posizione effettiva dell'agente e tale nuovo punto della traiettoria di riferimento. Nel problema in esame, questo aspetto va gestito con attenzione poichè può facilmente portare a collisioni dato che le transizioni fra gli stati dell'automa sono valuta-

te rispetto alla posizione effettiva dell'agente e non a quella predetta. Infatti, le decisioni di controllo devono essere prese in base al campo visivo dell'agente, il quale è, appunto, definito rispetto a tale posizione. Per chiarire meglio questa problematica, si consideri la Fig. 8.16: nella



(a) Scatto della transizione G2P

(b) Generazione della traiettoria a partire dalla predizione già elaborata nello stato precedente

Figura 8.16: Generazione della traiettoria nel cambio di stato senza reset della predizione

Fig. 8.16a l'agente, che si sta dirigendo verso il *goal*, si avvicina all'ostacolo fino allo scatto della transizione G2P, portandosi nello stato di *patrolling*. A questo punto, Fig. 8.16b, la generazione della traiettoria avviene mediante il problema di ottimizzazione (8.3). Come risulta evidente, poichè la nuova traiettoria viene "appesa" in coda a quella precedente, c'è la possibilità che l'agente collida con l'ostacolo. Per evitare questo fenomeno è necessario reinizializzare l'algoritmo di controllo predittivo rispetto alla posizione corrente dell'agente. In questo modo, verrà cancellata la traiettoria fin qui generata e i nuovi punti partiranno proprio dalla posizione attuale dell'agente. Il soddisfacente risultato di quest'operazione è riportato in Fig. 8.17. La reinizializzazione dell'algoritmo di controllo può essere eseguita con la tecnica descritta nella Sezione 6.4. Si noti che questo comporta un temporaneo arresto dell'agente poichè tale reset imposta la traiettoria di riferimento pari alla posizione attuale dell'agente per tutto l'orizzonte di predizione. Per ovviare a questo problema è possibile eseguire qualche iterazione del problema di ottimizzazione in modo da rigenerare una predizione tale da movimentare istantaneamente l'agente.

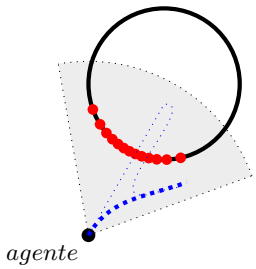


Figura 8.17: Generazione della traiettoria nel cambio di stato con reset della predizione

8.3.3 Prove in Simulazione

Si presentano ora delle prove eseguite, ancora una volta, in ambiente MATLAB in cui si vogliono verificare le prestazioni della tecnica presentata per la soluzione del problema in esame.

Nella prima prova, il cui risultato è riportato in Fig. 8.18, l'agente deve raggiungere il *goal* in un ambiente sconosciuto composto da ostacoli di forme diverse. In particolare si vogliono far notare i seguenti punti:

- L'agente raggiunge l'obiettivo imposto. Come discusso in precedenza, questo aspetto non è scontato poichè non è possibile garantire che l'agente riesca a trovare una via, ammesso che esista, per arrivare al *goal*.

- Nella configurazione riportata in Fig. 8.18d, l'agente rileva contemporaneamente due ostacoli differenti ed è necessario selezionare quello che verrà circumnavigato. In base a quando discusso in precedenza, viene scelto quello più vicino al *goal* poichè l'obiettivo ultimo è proprio il suo raggiungimento. Come voluto infatti, l'agente si muove verso tale ostacolo e quindi verso l'obiettivo, da cui riuscirà poi a trovare una strada verso il *goal*.

- Nelle Figg. 8.18b e 8.18f, è possibile notare la reinizializzazione dell'algoritmo di controllo e il corrispondente reset della traiettoria. In entrambi i casi infatti, la traiettoria generata prima dello scatto della transizione G2P guiderebbe l'agente verso una collisione con l'ostacolo. Grazie al reset invece, la nuova traiettoria per il *patrolling* viene generata a partire dalla posizione corrente

dell'agente, ignorando la vecchia predizione. In particolare, nelle figure è visibile il primo punto delle nuove traiettorie.

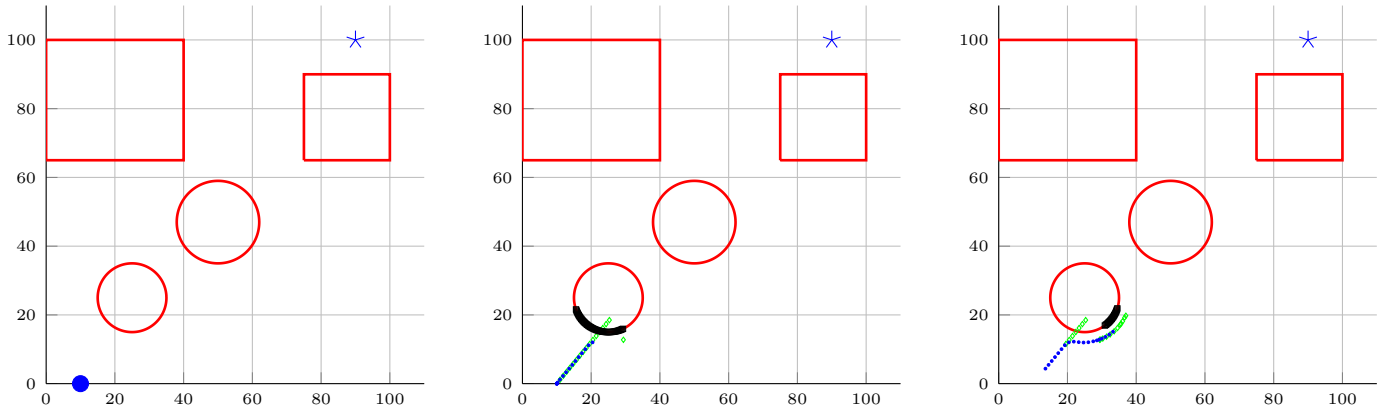
- Nelle Figg. 8.18g e 8.18h l'agente sta circumnavigando un ostacolo con punti ciechi. Per il superamento viene utilizzata la tecnica descritta nella Sezione 8.2.3, come visibile dagli archi generati nelle figure citate.

Si consideri ora un'ulteriore prova in cui si è voluto provare l'efficacia della tecnica presentata nel caso in cui l'ambiente sia composto da strettoie nelle quali l'agente non può passare soddisfacendo i vincoli di distanza. Le varie fasi della simulazione sono rappresentate in Fig. 8.19. Nelle figure sono anche riportati i punti P_{ms} e P_{BN} relativi allo stato BN, rispettivamente con una stellina nera e magenta. Si osservino in particolare i seguenti punti:

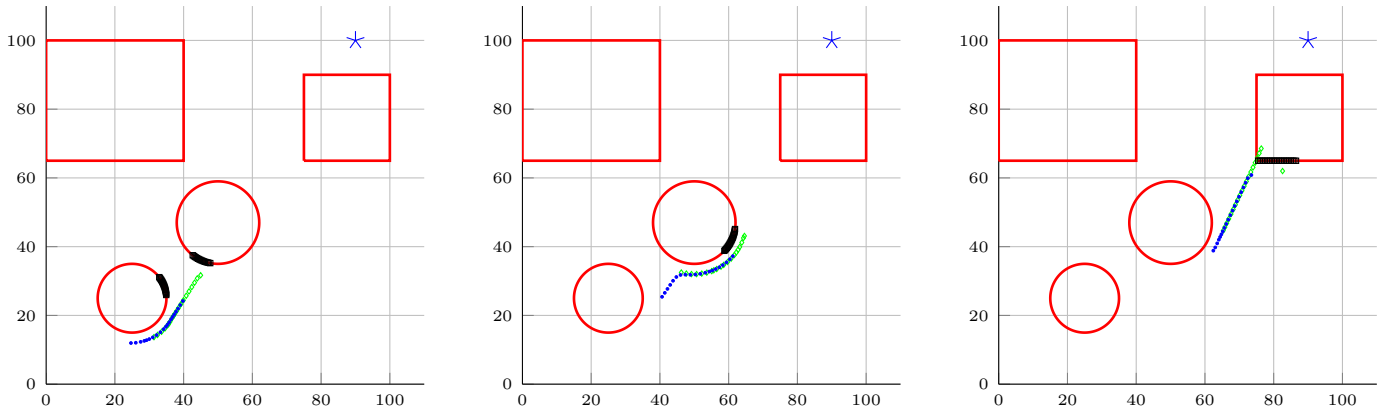
- Nella fase iniziale, l'agente circumnaviga l'ostacolo di sinistra poichè più vicino al *goal*.

- Le strettoie vengono rilevate più volte poichè si è scelto di non implementare nessun tipo di memoria. Dunque, quando l'agente identifica la presenza di una strettoia precedentemente rilevata ed evitata, esso la considera come se non fosse mai stata gestita. L'effetto negativo di questa scelta è evidente nelle Figg. 8.19b, 8.19d e 8.19f in cui scatta la transizione P2BN a causa sempre della strettoia di sinistra. D'altro canto, la non presenza di memoria rende più facile l'implementazione dell'algoritmo.

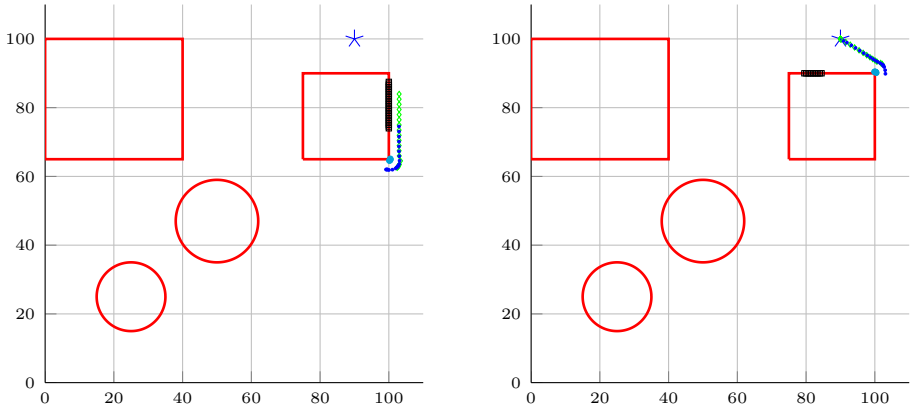
- La scelta di calcolare il punto di fuga, indicato con P_{BN} risulta efficace poichè porta l'agente, come voluto, distante dal punto in cui è stata rilevata la strettoia. Si noti però che la posizione di tale punto non è sempre ottimale. Infatti, con riferimento alla Fig. 8.19c, esso si trova proprio nell'altra strettoia. Questo aspetto è legato anche a quanto detto al punto precedente: dato che l'agente non è dotato di memoria, non c'è modo di valutare la "bontà" di tale punto.



(a) Configurazione iniziale (b) Scatto G2P (c) Scatto P2G

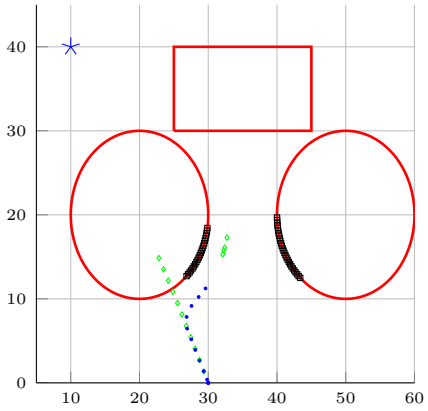


(d) Scatto G2P (e) Scatto G2P (f) Scatto P2G

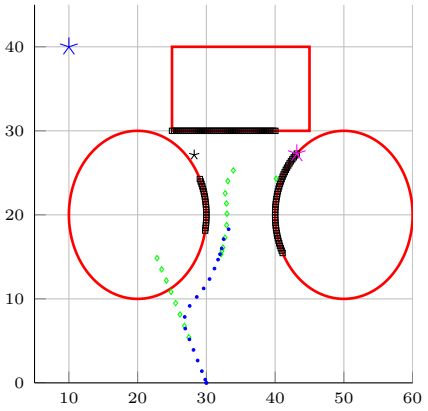


(g) Patrolling ostacolo con punto cieco (h) Raggiungimento del *goal*

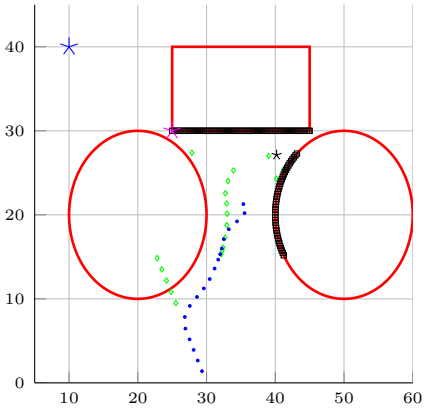
Figura 8.18: Prova navigazione in ambiente sconosciuto



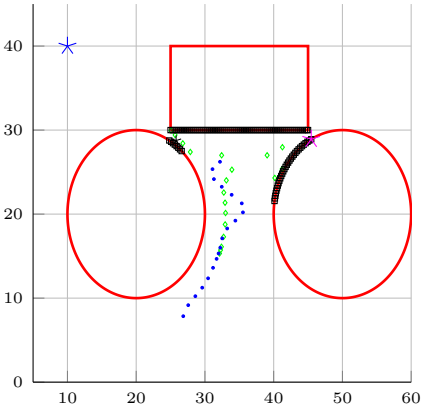
(a) Patrolling di un ostacolo



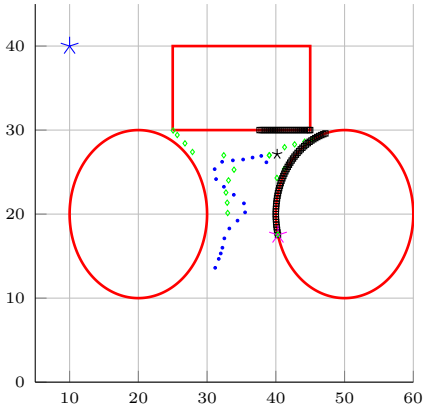
(b) Individuazione strettoia di sinistra e scatto P2BN



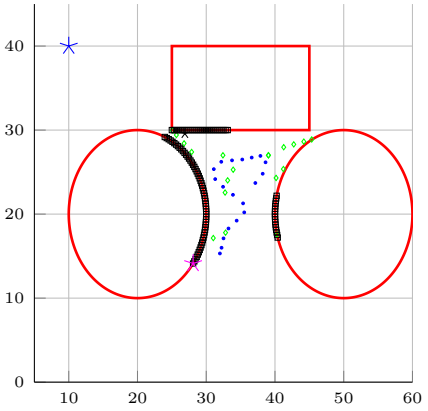
(c) Individuazione strettoia di destra e scatto P2BN



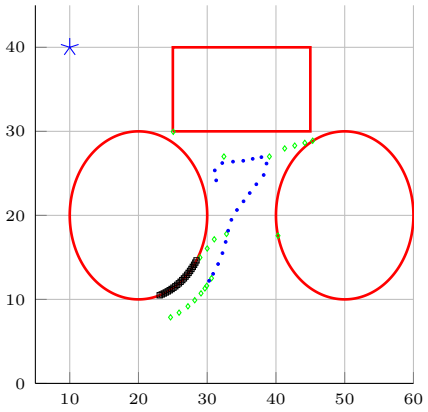
(d) Individuazione strettoia di sinistra e scatto P2BN



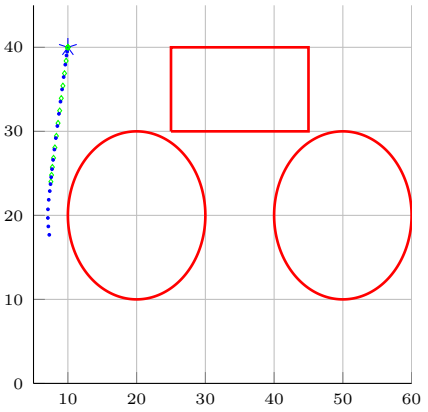
(e) Individuazione strettoia di destra e scatto P2BN



(f) Individuazione strettoia di sinistra e scatto P2BN



(g) Patrolling dell'ostacolo in uscita



(h) Raggiungimento del *goal*

Figura 8.19: Prova navigazione in ambiente sconosciuto con presenza di strettoie

Capitolo 9

Risultati Sperimentali

In questo capitolo verranno presentati i risultati delle varie prove sperimentali effettuate utilizzando il *setup* presentato nel Capitolo 2. Inizialmente si illustreranno le prove eseguite con un singolo agente con l'obiettivo di provare i risultati teorici ottenuti nel Capitolo 4, per poi passare ai test relativi agli algoritmi elaborati per la soluzione del problema del controllo di formazione, del contenimento e della navigazione in ambiente sconosciuto, descritti rispettivamente nei Capitoli 5, 7 e 8. Si ricorda che, coerentemente con la notazione introdotta nel Capitolo 5, con $\tilde{y}$ si indica la traiettoria di riferimento per le uscita e con y quella effettivamente percorsa dall'agente.

9.1 Prove con un singolo agente

Si consideri il semplice problema di far raggiungere ad un agente un obiettivo, in un ambiente privo di ostacoli, la cui soluzione è stata analizzata nella Sezione 4.5. Il risultato della prova sperimentale è illustrato in Fig. 9.1 e prova quanto dimostrato:

- La traiettoria effettiva del robot si trova in un intorno limitato e noto della traiettoria di riferimento generata dal sistema di controllo
- La distanza fra due punti successivi del riferimento è limitata dalla dimensione scelta di $\beta_\varepsilon(0)$. Nella simulazione si è utilizzata una bolla circolare di raggio $1.5cm$.

Si estenda ora il problema includendo un ostacolo circolare nel piano di lavoro. Come descritto nella Sezione 5.3.3, l'*obstacle avoidance*

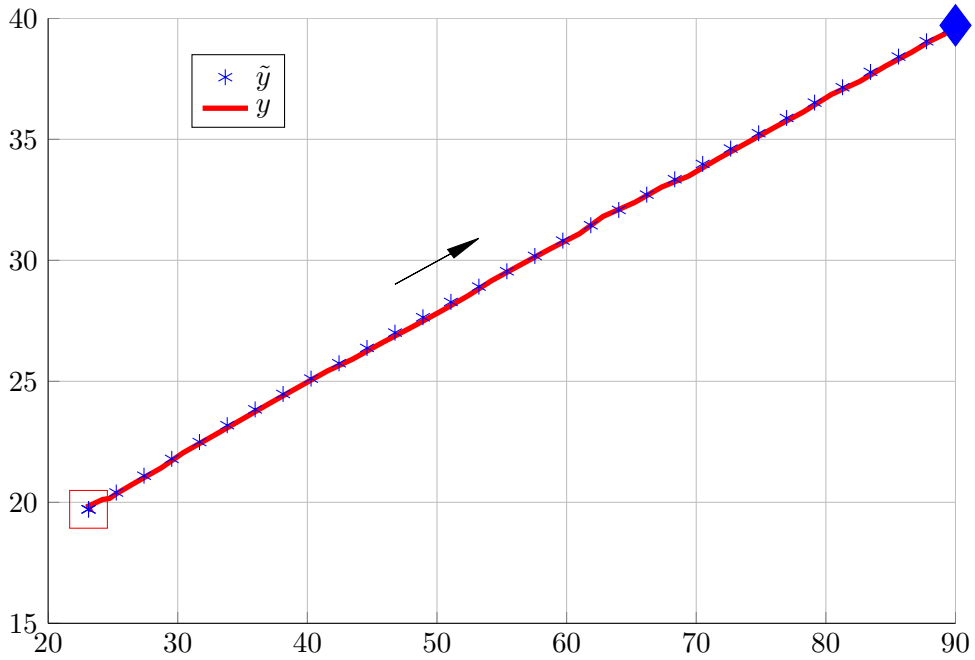


Figura 9.1: Prova sperimentale inseguimento robusto traiettoria

può essere risolta semplicemente implementando il vincolo (5.7), che mantiene il robot ad una certa distanza minima dall'ostacolo stesso. Il risultato, presentato in Fig. 9.2, oltre a confermare i risultati teorici relativi all'inseguimento della traiettoria generata, evidenzia come il robot eviti efficacemente l'ostacolo, mediante un percorso circolare attorno ad esso. La traiettoria generata è infatti circolare a causa della natura del vincolo in norma euclidea utilizzato, che porta ad una regione in cui è violato di natura, appunto, circolare.

9.2 Controllo di Formazione

Si presentano ora i risultati sperimentali relativi agli algoritmi proposti nel Capitolo 5 per il controllo di un insieme di agenti nel rispetto di una data formazione in base alla tecnica *leader-follower*. In particolare, si presenteranno i risultati ottenuti mediante le due soluzioni proposte che verranno poi confrontati con la tecnica di controllo descritta nel Capitolo 3.

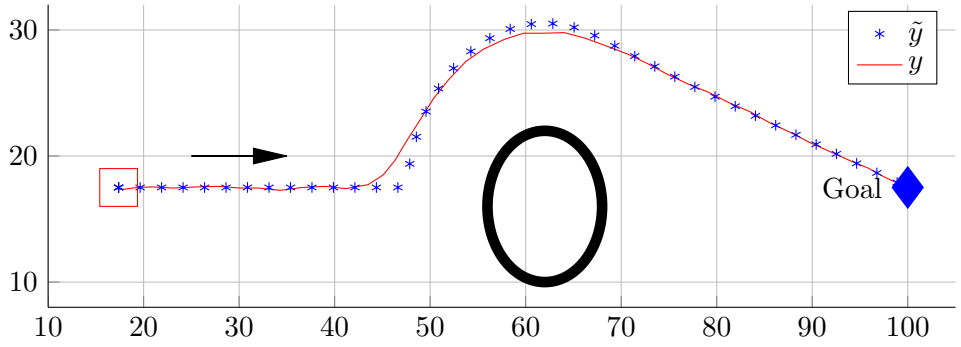


Figura 9.2: Prova sperimentale collision avoidance con un agente

9.2.1 Mantenimento della formazione e raggiungimento obiettivo

Si consideri inizialmente l'algoritmo di controllo introdotto nella Sezione 5.3.1. In Fig. 9.3 è illustrato il risultato della prova in cui si è scelta una formazione triangolare. In particolare si possono notare i seguenti punti:

- Nella fase iniziale i *follower* sono fuori formazione ed il *leader* rallenta aspettandoli fino a quando la formazione è raggiunta. Da questo momento in avanti tutti gli agenti procedono alla massima velocità. Questo risultato è stato ottenuto utilizzando l'accorgimento relativo alla variazione dinamica della dimensione della bolla $\beta_\varepsilon(0)$, discusso nella Sezione 5.3.1.

- Fintantochè il *leader* non raggiunge il suo goal, i *follower*, come voluto, lo scortano in formazione triangolare. Si noti però che la scelta di un vincolo sulla distanza fra gli agenti per il mantenimento della formazione porta ad una non univoca posizione per la formazione stessa. Infatti, imporre una distanza minima, ad esempio dal *leader*, equivale a definire una regione circolare centrata nella posizione del *leader* stesso, fuori dalla quale il vincolo è soddisfatto. In Fig. 9.4 è illustrato un esempio di due, fra le infinite, possibili configurazioni della formazione in esame. Con riferimento alla Fig. 9.3, si noti come questa problematica porti a delle rotazioni della formazione quando il *follower* F_1 si avvicina al suo obiettivo.

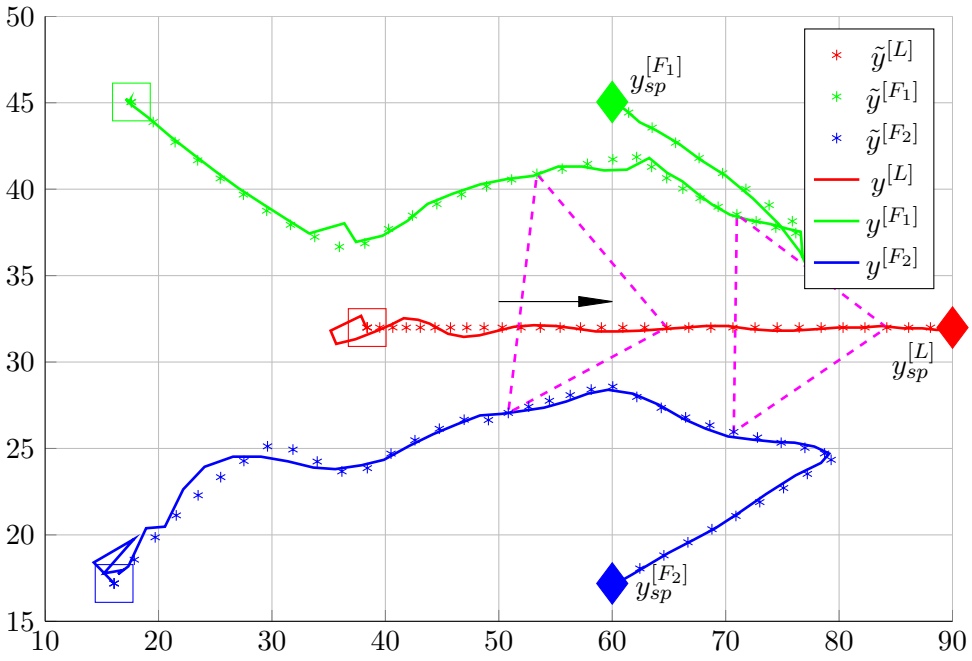


Figura 9.3: Simulazione formazione e goal individuale

- Quando il *leader* raggiunge il suo obiettivo, il costo dell'errore di formazione viene posto a zero e, come ci si aspetta, i *follower* si dirigono verso i propri obiettivi.

9.2.2 Mantenimento della formazione con setpoint variabile

Si consideri ora l'algoritmo presentato nella Sezione 5.3.2, in cui la formazione è mantenuta generando dei *setpoint* variabili ai *follower* in funzione della posizione del *leader*. Anche in questa soluzione si è implementata la variazione dinamica della dimensione della bolla $\beta_\varepsilon(0)$ cosicchè il *leader* rallenti quando i *follower* sono fuori formazione. La formazione di riferimento è, ancora una volta, triangolare in modo che sia possibile effettuare un confronto con i risultati precedenti. In Fig. 9.5 è illustrata una prova analoga a quella in Fig. 9.3, che evidenzia però un migliore risultato dato che la formazione è sempre correttamente mantenuta poichè univocamente determinata in quanto è direttamente calcolata dalla posizione di riferimento del *leader*. Si sono effettuate ulteriori prove in cui si è variata la configurazione iniziale degli agenti. In particolare, in Fig. 9.6 si sono posti i *follower*

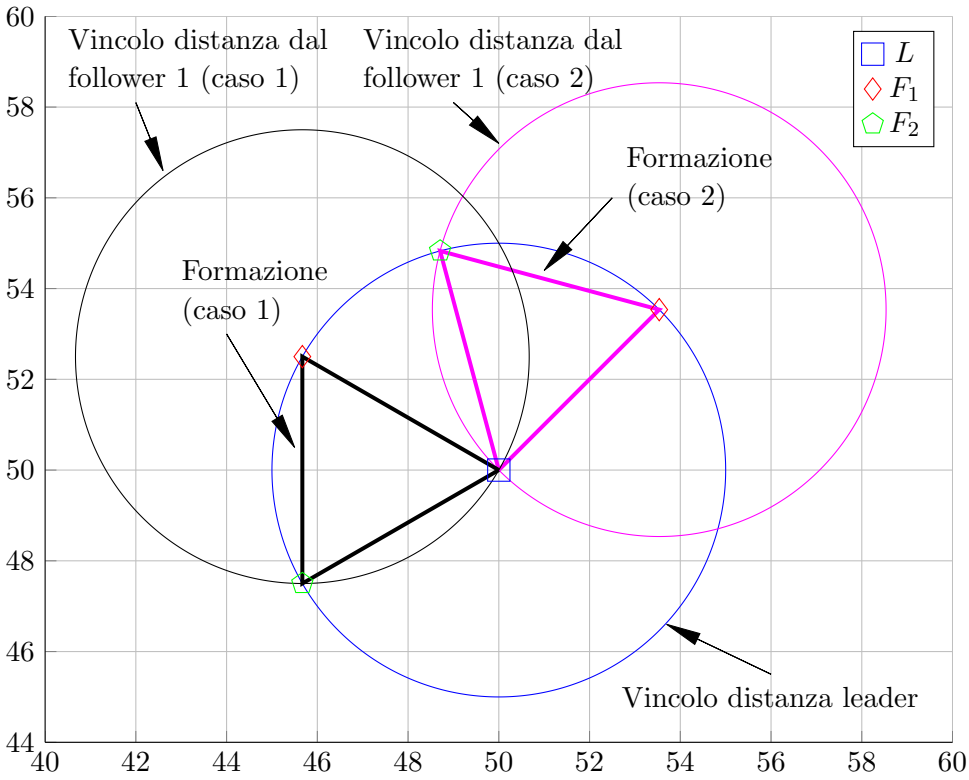


Figura 9.4: Problematica nell'utilizzo di vincoli in distanza

anteriormente rispetto al *leader* ed in Fig. 9.7, ne sono state invertite le posizioni iniziali. Si noti infine come nella Fig. 9.7, il *follower* F_2 sia costretto ad allontanarsi da F_1 a causa dei vincoli di precedenza e, appena F_1 è sufficientemente lontano, inverta la direzione di movimento per completare la formazione.

Obstacle Avoidance

In aggiunta al mantenimento della formazione, si è implementato anche l'*obstacle avoidance* utilizzando la tecnica descritta nella Sezione 5.3.3, includendo quindi un vincolo di distanza minima per ogni ostacolo nel problema di ottimizzazione di ogni agente. Il risultato della prova sperimentale è presentato in Fig. 9.8. Si noti che il *follower* F_2 è ora in grado sia di evitare gli ostacoli che di dare la precedenza al *follower* F_1. Questi comportamenti sono visibili nella citata figura, nell'intorno della posizione $(50, 25)$, in cui F_2 rallenta fino a fermarsi, per dare la

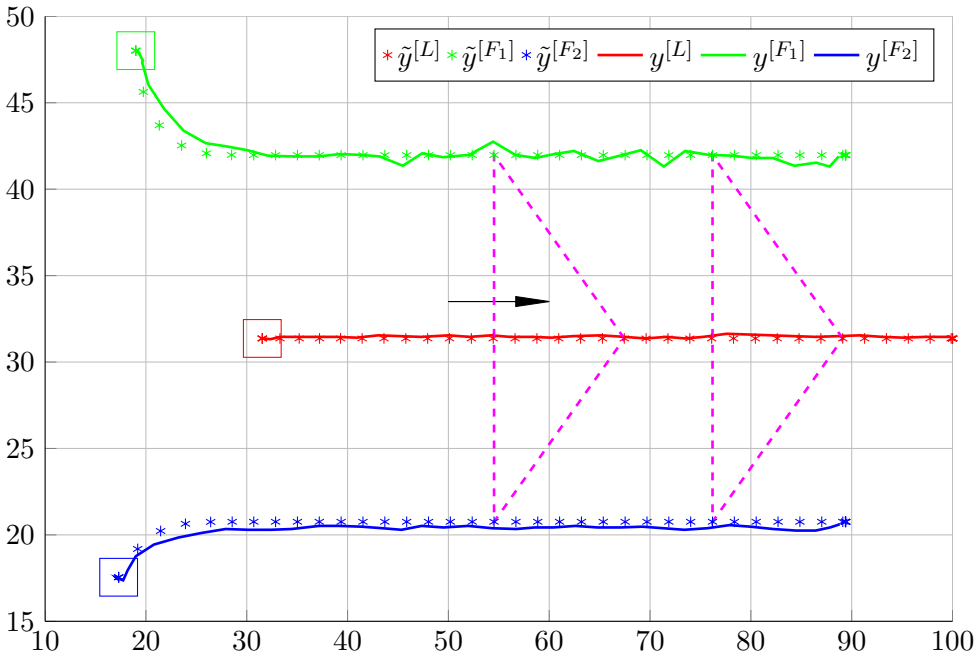


Figura 9.5: Formazione con setpoint variabile - Prova 1

precedenza a F_1, e continua il suo percorso solo successivamente al passaggio di quest'ultimo, rimanendo sempre ad una distanza minima dagli ostacoli.

9.2.3 Confronto con $SB_{ij}C$, $S_{ij}S_{jk}C$

Si vuole confrontare la soluzione di controllo fin qui discussa con la tecnica descritta nella Sezione 3.2. Nelle Figg. 9.9 e 9.10 sono presentate due diverse prove sperimentali, con l'obiettivo di evidenziare i diversi comportamenti dei due algoritmi. In particolare si notino i seguenti vantaggi e svantaggi nell'utilizzo dell'algoritmo presentato in questa tesi:

- Le traiettorie generate con MPC sono più prevedibili e permettono di determinare a priori le distanze minime che gli agenti manterranno durante il movimento.
- Le tecniche $SB_{ij}C$ e $S_{ij}S_{jk}C$ richiedono un'accurata selezione dei parametri degli algoritmi di controllo per evitare saturazioni delle variabili di attuazione, cioè delle velocità dei motori passo-passo

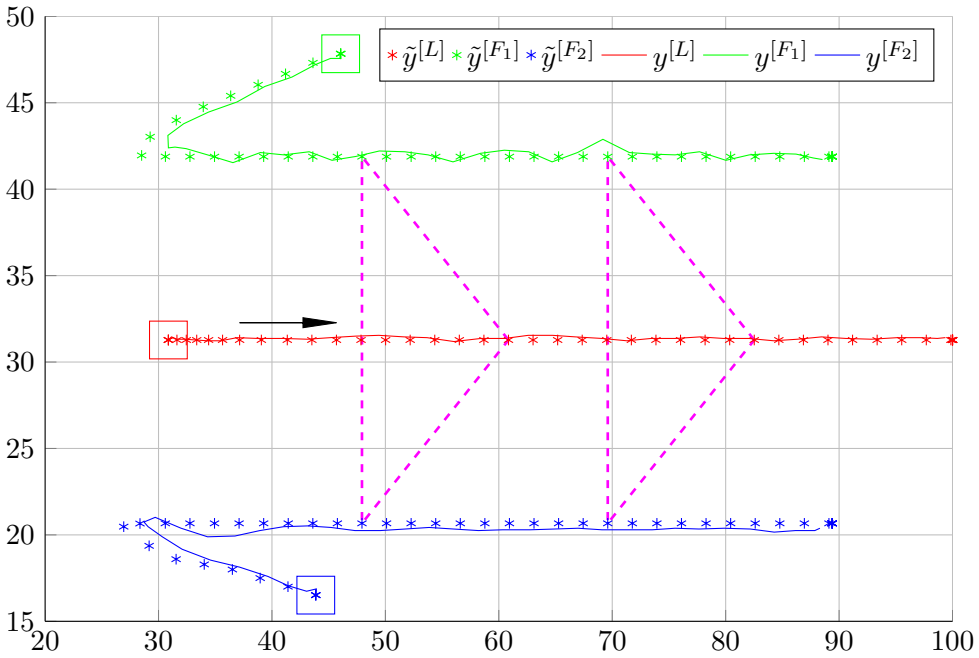


Figura 9.6: Formazione con setpoint variabile - Prova 2

che mobilitano gli agenti. La scelta di tali parametri dipende anche dalla configurazione iniziale in quanto le leggi di controllo (3.9) e (3.13) sono funzione della distanza fra gli agenti. Tale dipendenza è un aspetto critico in una generica situazione, poichè non permette una soluzione generale per il controllo. D'altro canto, l'utilizzo di problemi di ottimizzazione per la determinazione delle variabili di controllo permette di limitare tali variabili mediante vincoli espliciti che garantiscono che il problema della saturazione sia evitato. Si noti come, per agenti uniciclo, questo problema sia critico poichè in caso di saturazione il robot potrebbe roteare su sè stesso, portando ad instabilità. Infatti, utilizzando $SB_{ij}C$ o $S_{ij}S_{jk}C$, se l'agente ha questo comportamento, la sua distanza dall'obiettivo aumenta causando azioni di controllo ancora più elevate. Inoltre, l'applicazione della tecnica *Tube-Based MPC* garantisce robustezza del sistema di controllo anche a fronte di incertezze nel modello dell'agenti e di disturbi agenti sul robot.

- L'utilizzo delle tecniche $SB_{ij}C$, $S_{ij}S_{jk}C$ richiede l'impiego della strategia di commutazione descritta nella Sezione 3.2.3. Quando

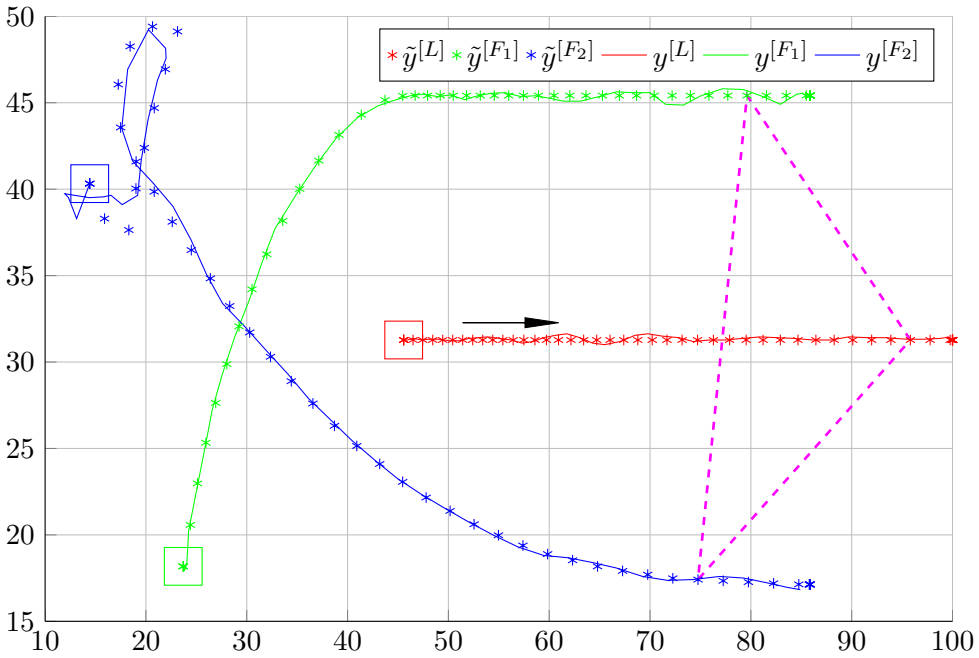


Figura 9.7: Formazione con setpoint variabile - Prova 3

avviene la commutazione fra i due algoritmi sono possibili discontinuità nell'azione di controllo, come visibile nell'intorno della posizione $(25, 28)$ della Fig. 9.10. Questa problematica non è presente nella soluzione di controllo proposta.

- L'algoritmo di controllo predittivo proposto richiede l'utilizzo dell'anello linearizzante, descritto nella Sezione 2.4, a causa della non linearità del modello dell'agente considerato. Tale anello risulta non definito per velocità lineare nulla e questo provoca comportamenti imprevedibili a velocità molto ridotte. Contrariamente, le tecniche $SB_{ij}C$ e $S_{ij}S_{jk}C$ sono progettate direttamente sul modello non lineare e non presentano quindi questa problematica.

- L'implementazione della tecnica MPC proposta richiede uno sforzo computazionale molto più elevato delle soluzioni $SB_{ij}C$ e $S_{ij}S_{jk}C$, che necessitano quindi di calcolatori più performanti che potrebbero non essere disponibili localmente negli agenti stessi. Si ricorda infatti che, ad ogni istante di tempo è necessario risolvere sia il problema di ottimizzazione relativo alla generazione della traiettoria che quello relativo al controllo robusto, oltre che

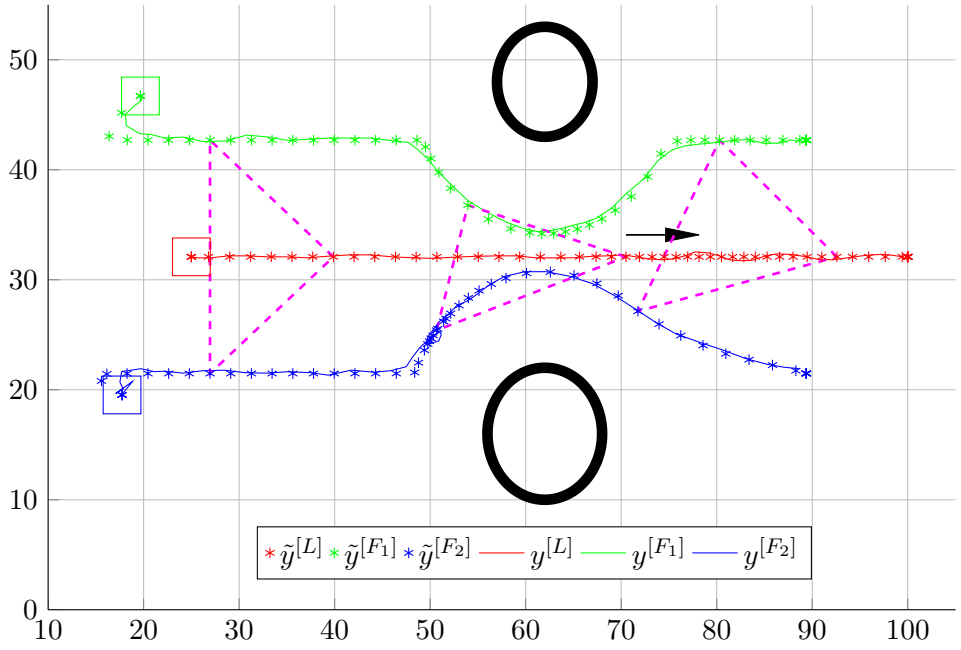


Figura 9.8: Prova sperimentale obstacle avoidance con formazione

ad implementare, a frequenze più elevate, l'anello linearizzante interno.

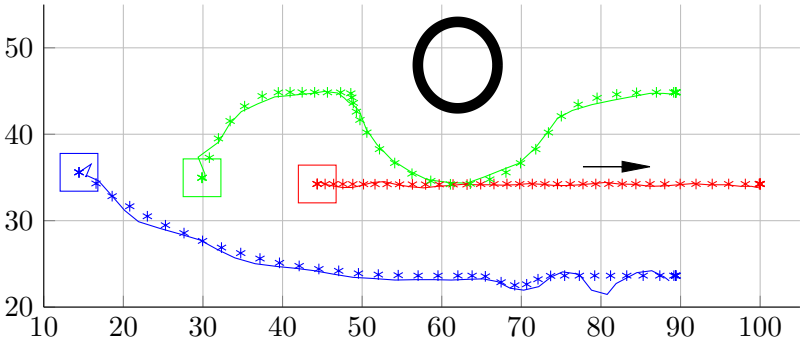


Figura 9.9: Comparazione - MPC

9.3 Problema del Contenimento

Si presentano ora i risultati ottenuti mediante la soluzione di controllo proposta nel Capitolo 7 per la soluzione del problema del contenimento.

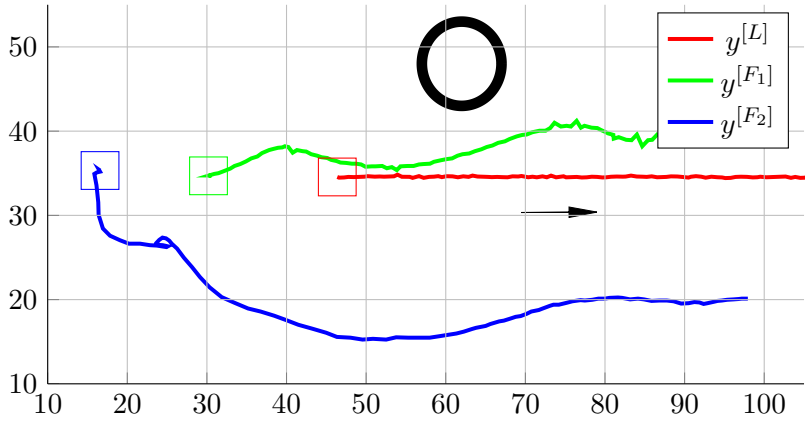


Figura 9.10: Comparazione - $SB_{ij}C, S_{ij}S_{jk}C$

Si sono utilizzati due robot *e-puck* come *follower* e quattro *leader* virtuali, la cui rete di comunicazione è rappresentata dal grafo in Fig. 9.11. La matrice $K_{G_{lf}}$ introdotta nella Sezione 7.2.2, che descrive

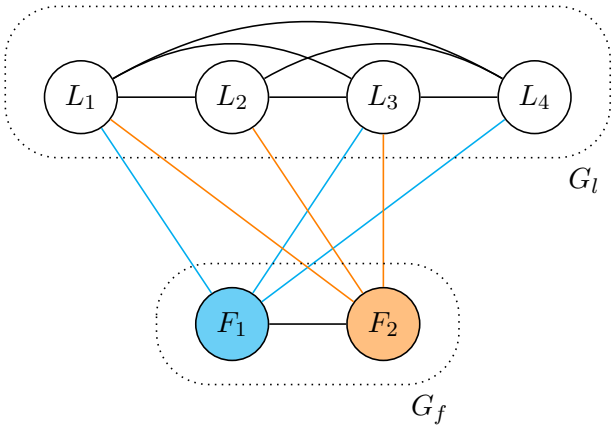


Figura 9.11: Prova Epuck - Grafo comunicazione *leader-follower*

la rete di comunicazione fra i *leader* e i *follower*, è quindi:

$$K_{G_{lf}} = \begin{bmatrix} \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \end{bmatrix} \tag{9.1}$$

Compatibilmente con l'area a disposizione per eseguire la prova, si sono scelte le due regioni ed i relativi punti di misura riportati in Fig. 9.12. Analogamente alla prova precedente, nella generazione della traiettoria di ciascun *follower* si è posto un vincolo di *collision avoidance* in modo da evitare collisioni fra i *follower* stessi. Verranno ora presentati i risultati in varie fasi, evidenziando in particolare la configurazione degli

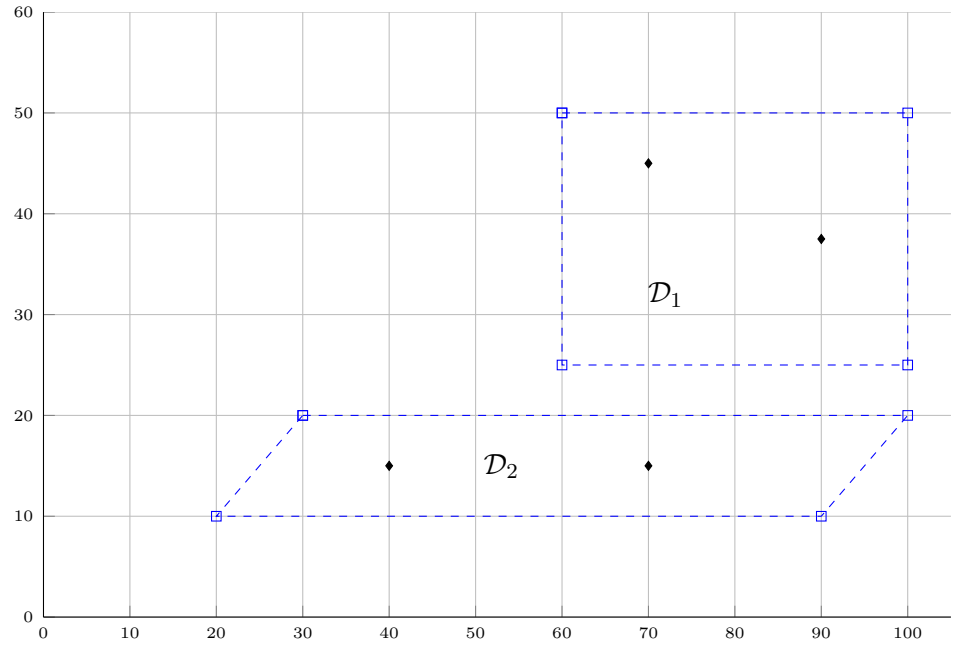


Figura 9.12: Prova Epuck - Regioni da visitare e relativi punti di misura

agenti quando sia ha una commutazione fra due stati negli automi delle due classi di agenti.

Posizione inziale e movimentazione verso la regione $\mathcal{D}_1$

Nella Fig. 9.13 sono illustrate le traiettorie degli agenti corrispondentemente allo scattare di alcune transizione. A partire dalla configurazione iniziale di Fig. 9.13a, i *leader* si muovono verso i rispettivi vertici garantendo che i *follower* restino sempre contenuti in base alla precedenti definizioni. Quando i *leader* giungono nei vertici, Fig. 9.13b, ciascun *follower* viene assegnato ad una posizione di misura ed inizia quindi a dirigersi verso tale punto, come illustrato nella Fig. 9.13c. Una volta raggiunti tali punti la regione $\mathcal{D}_1$ è conclusa ed i *leader* possono quindi passare alla successiva. Durante la movimentazione dalla configurazione iniziale alla prima regione si è registrata una perdita del contenimento ed il successivo recupero. Nel confronto fra le Figg. 9.14a e 9.14b si noti come nella prima il *follower* f_1 sia distante dal *convex hull* generato dai *leader* vicini e quindi non più contenuto e nella seconda come esso sia rientrato in tale regione poichè i *leader* si sono arrestati.

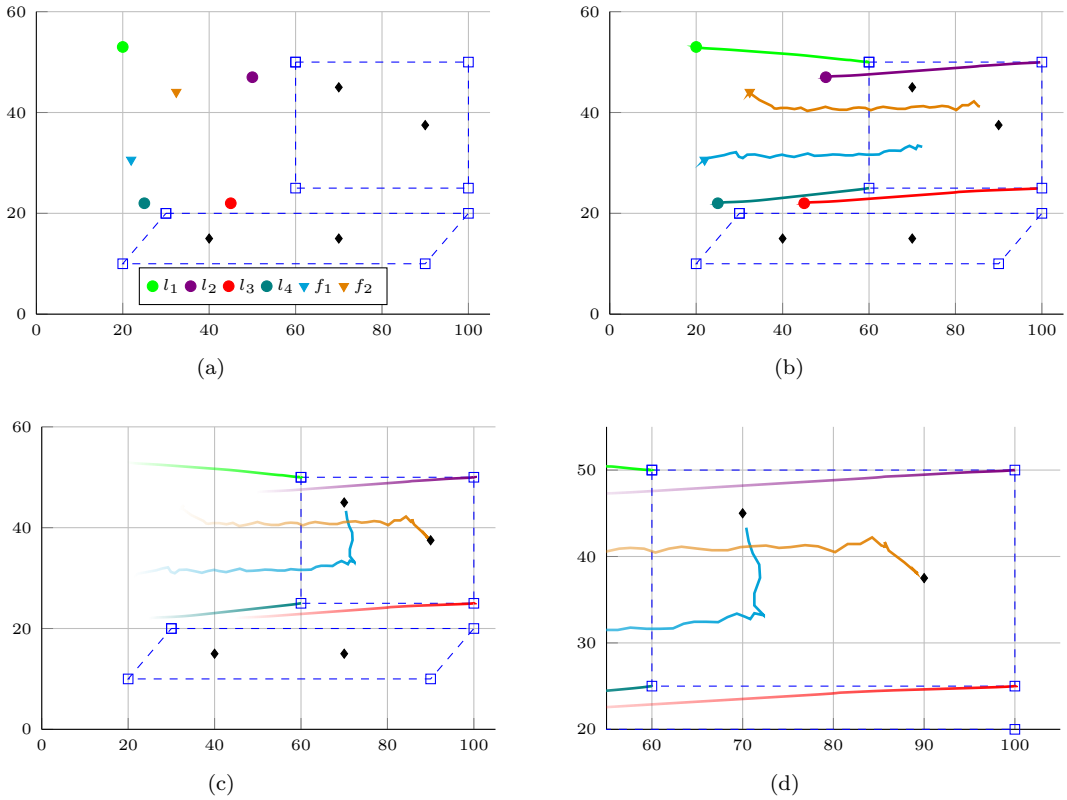


Figura 9.13: Prova Epuck - Configurazione inziale e traiettorie per la regione $\mathcal{D}_1$

Movimentazione verso la regione $\mathcal{D}_2$

Infine, nelle Figg. 9.15a e 9.15b sono riportate le configurazioni degli agenti corrispondentemente alle transizioni GO2TARGET e TARGET2GO nella movimentazione dalla regione $\mathcal{D}_1$ alla regione $\mathcal{D}_2$.

9.4 Problema della Navigazione in Ambiente Sconosciuto

Si presenta ora una prova della tecnica di controllo presentata nel Capitolo 8. Con l'obiettivo di limitare la complessità nell'implementazione degli algoritmi, si è scelto di non utilizzare i sensori di prossimità presenti nei robot *epuck* per la rilevazione dei punti visibili degli ostacoli adiacenti l'agente, optando invece per lo stesso algoritmo utilizzato in simulazione che permette, appunto, di calcolare tale punti a parti-

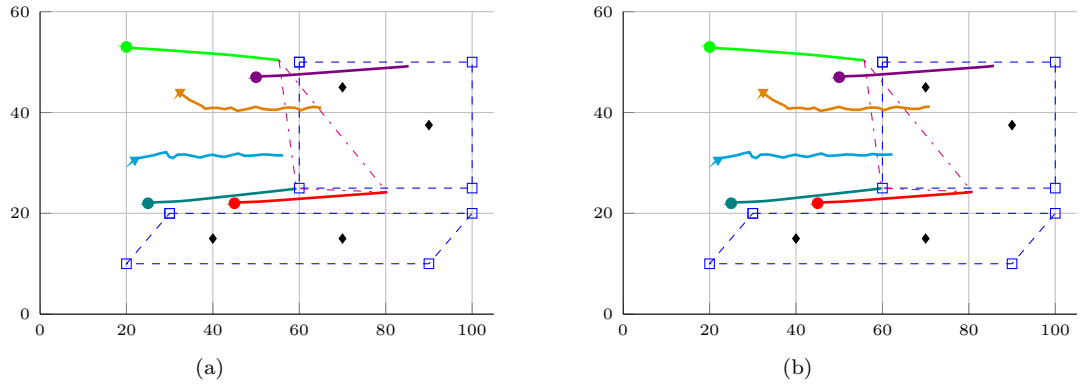


Figura 9.14: Prova Epuck - Perdita e recupero contenimento

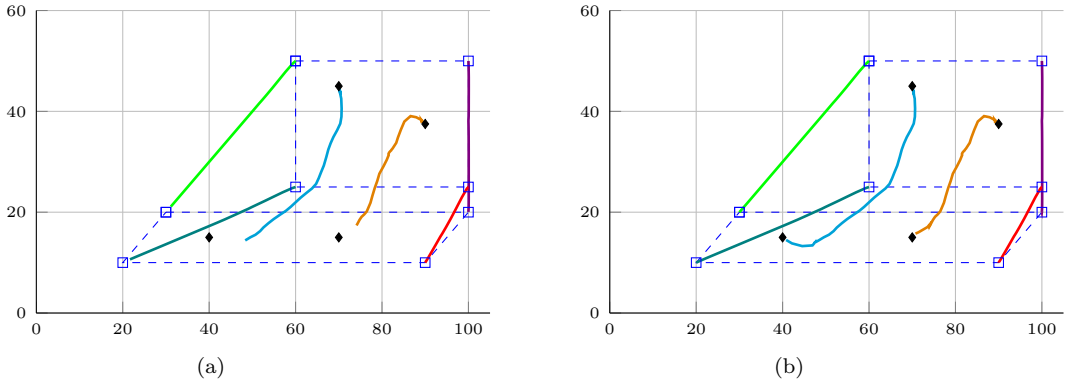


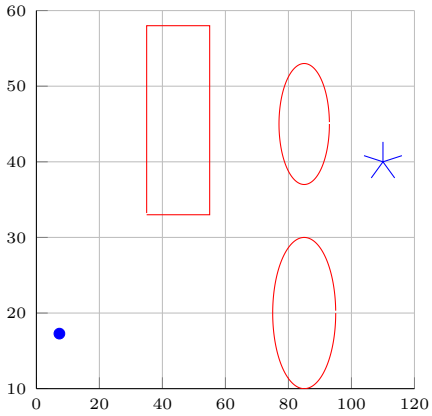
Figura 9.15: Epuck - Traittorie da $\mathcal{D}_1$ a $\mathcal{D}_2$

re dalla conoscenza della posizione e dell'orientamento dell'agente. I parametri utilizzati sono:

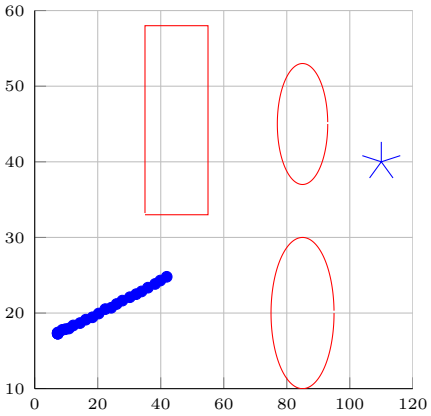
- Distanza a cui l'agente si deve mantenere dall'ostacolo: $d_{obst} = 8cm$
- Angolo di visuale: $\alpha_{vis} = \pi/2$
- Distanza massima di visuale: $d_{vis} = 15cm$
- Ampiezza arco di punti fittizi per il superamento di un punto cieco: $\alpha_{BL} = \pi/2$

Le varie fasi della navigazione ottenuta sono rappresentate in Fig. 9.16. In particolare si osservino i seguenti aspetti:

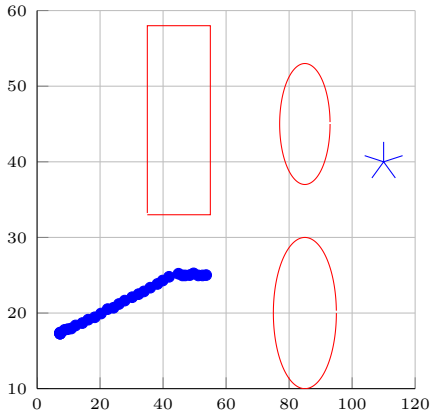
- L'agente non collide con nessun ostacolo, poichè esso viene riportato nello stato di *patrolling* ogni volta che viene violato il vincolo di distanza minima. Infatti, nella Fig. 9.16b, l'agente viola il detto vincolo portando allo scatto della transizione G2P.
- L'agente rileva ed evita correttamente la strettoia, trovando una strada alternativa per raggiungere l'obiettivo finale, come illustrato nelle Figg. 9.16e e 9.16f.
- Il *goal* viene correttamente raggiunto, aspetto, come già detto, non scontato.



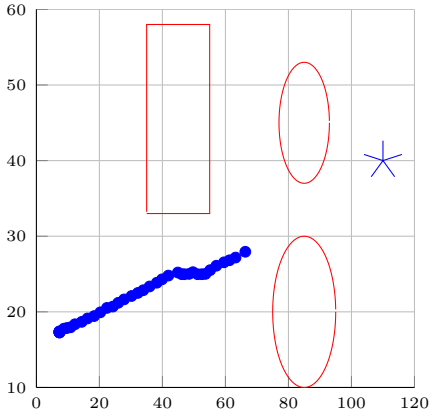
(a) Configurazione iniziale. Stato G.



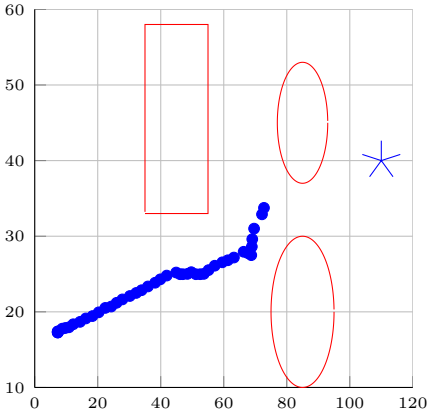
(b) Scatto di G2P



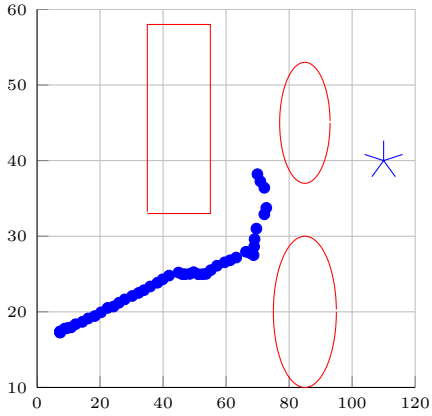
(c) Scatto di P2G



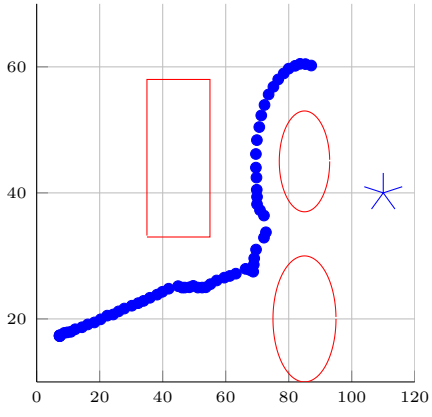
(d) Scatto di G2P



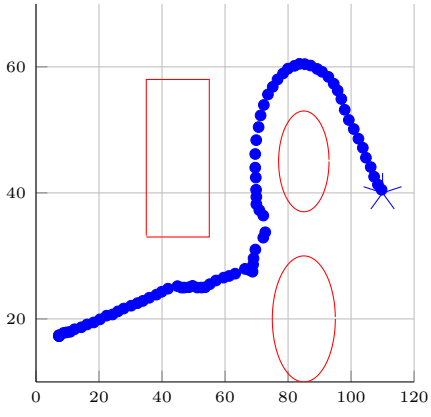
(e) Rilevazione strettoia e scatto di P2BN



(f) Scatto di BN2P



(g) Scatto di P2G



(h) Raggiungimento del *goal*

Figura 9.16: Prova sperimentale della navigazione in ambiente sconosciuto

Capitolo 10

Conclusioni e sviluppi futuri

La tecnica di controllo predittivo gerarchico analizzata in questa tesi presenta diversi vantaggi rispetto a molte tecniche proposte in letteratura. In particolare si vogliono rimarcare i seguenti punti:

- La struttura gerarchica permette di separare il problema di garantire robustezza nell'inseguimento della traiettoria di riferimento, rispetto a quello della generazione della stessa, rendendo quest'ultimo più semplice sia del punto di vista teorico che computazionale.
- La dinamica dell'agente è interamente gestita nei livelli inferiori di controllo mediante la tecnica *Tube-Based MPC* permettendo quindi di elaborare soluzioni per la generazione della traiettoria di carattere generale, indipendenti dall'effettivo agente utilizzato.
- La tecnica robusta *Tube-Based MPC* permette di conoscere 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, purchè rispettino le ipotesi introdotte.
- Dal punto di vista computazionale si evidenziano notevoli vantaggi rispetto a molte soluzioni presenti in letteratura. Infatti, è comune trovare problemi di ottimizzazione non lineari di elevate dimensioni, che possono facilmente compromettere l'effettiva applicabilità di tali soluzioni in un contesto reale a causa degli elevati tempi di elaborazione necessari. La tecnica qui proposta invece, a meno della generazione della traiettoria, richiede di risolvere un problema di programmazione lineare quadratica, per il quale esistono efficienti algoritmi risolutivi.

- La generazione della traiettoria può essere ottenuta mediante la soluzione di un problema di ottimizzazione a un passo che, grazie alla dimensioni ridotte, può essere facilmente risolto anche se di natura non lineare.

I vantaggi appena discussi aprono la strada a possibili sviluppi futuri volti a migliorare ulteriormente prestazioni ed efficienza dell'algoritmo proposto. In particolare si ritiene interessante lo sviluppo nelle seguenti direzioni:

- L'anello di controllo linearizzante più vicino all'agente non risulta definito a velocità eccessivamente ridotte, rendendo difficile la percorrenza di traiettorie lente. Interessante potrebbe quindì essere la ricerca di soluzioni alternative che permettano un controllo completo e uniforme al variare della velocità, come ad esempio quella discussa in [1].

- Con l'idea di rendere più precisa la misura della posizione dell'agente si può pensare di affiancare agli algoritmi di controllo un filtro di Kalman volto a stimare tale posizione nell'intervallo compreso fra due misure successive. In questo modo l'effetto di un eventuale collo di bottiglia legato alla determinazione della posizione dell'agente può essere attenuato.

- Dal punto di vista della generazione della traiettoria potrebbe essere interessante un ulteriore approfondimento e studio del problema della navigazione in ambiente sconosciuto dotando l'agente di svariati e sofisticati comportamenti che portino a traiettorie più naturali. A tal proposito si può pensare di utilizzare i sensori a bordo del robot *epuck*, come ad esempio la telecamera VGA e i sensori di prossimità disposti perimetralmente. In particolare, la prima permette di acquisire un'immagine dell'ambiente circostante che può poi essere analizzata per estrarre informazioni utili alla generazione della traiettoria.

- Attualmente tutti i livelli di controllo sono implementati in modo distribuito su un unico calcolatore, ma grazie al performante processore a bordo del robot *epuck* su può pensare di rielaborare tali algoritmi in modo che possano essere eseguiti localmente nell'agente stesso, rendendo il complessivo sistema di controllo più efficiente.

Al di là di questi sviluppi volti a migliorare le singole componenti del sistema di controllo completo, si ritiene interessante anche l'implementazione delle tecniche presentate per lo studio della robustezza a fronte di guasti e malfunzionamenti degli agenti coinvolti. In questo modo infatti sarà possibile incrementare ulteriormente la robustezza globale rendendo questa soluzione più adatta ad applicazioni pratiche al di fuori di un contesto di ricerca.

Appendice A

Moduli Implementativi

L'implementazione degli algoritmi di controllo presentati in questa tesi è stata realizzata in ambiente MATLAB utilizzando tecniche di programmazione ad oggetti. Questa scelta permette una più facile strutturazione modulare del codice e porta a sorgenti più leggibili e riutilizzabili. Inoltre, rende più facile l'utilizzo di tali moduli da parte di utenti estranei e soprattutto semplifica eventuali sviluppi futuri grazie alla possibilità offerta dal linguaggio di creare sottoclassi. Si sono create diverse classi, ciascuna dedicata a risolvere un compito ben specifico, che verranno ora presentate. L'idea delle successive implementazioni è quella di fornire ad un futuro utilizzatore il maggior livello di astrazione possibile. Tutte le classi qui descritte implementano un metodo chiamato `step` attraverso il quale si esegue un'iterazione del corrispondente algoritmo, analogamente alle classi del Vision Toolbox di Matlab.

A.1 Simulazione di un *epuck*: Epuck

La classe **Epuck** implementa tutto il necessario per simulare il comportamento di un robot *epuck* controllato mediante un anello interno di *feedback linearization*. Essa fornisce semplici metodi pubblici che permettono di eseguire tutte le operazione in modo chiaro. Nel codice che segue sono illustrati e descritti tutti gli attributi e metodo pubblici della classe in esame.

```
1 classdef Epuck < handle
2 %
3 %   -- Classe Epuck --
4 %   Questa classe simula il comportamento di un agente epuck controllato
```

```
5  %   mendiante l'anello linearizzante
6  %
7  %   # COSTRUTTORE: Epuck(x, y, theta, v_old, Ts)
8  %       * x, y:  posizione inziale
9  %       * theta: orientamento inziale
10 %       * v_old: velocità iniziale
11 %       * Ts: tempo di campionamento
12 %
13 %   # ATTRIBUTI PUBBLICI
14 %       * position: attuale posizione
15 %       * state: stato (non lineare)
16 %       * linState: stato linearizzato
17 %       * color: colore con cui viene stampata la posizione attraverso il
18 %                metodo plot. Rispetta la sintassi di plot ('r*', ...)
19 %
20 %
21 %   # METODI PUBBLICI
22 %       * step(ax, ay): applica all'agente un'accelerazione [ax, ay]
23 %       * plot(): stampa la posizione
24 %
25 %
26
27     properties
28
29         color = 'b*';
30
31     end
32
33     properties (SetAccess = protected)
34
35         position = [];
36         Ts = 0.1;
37         state = [];
38         linState = [];
39         v_old = 0;
40
41          % Dimensioni
42         E = 5.2;
43         R = 2.05;
44
45     end
46
47     methods
48
49         function pos = get.position(epuck)
50             pos = epuck.state(1:2, :);
51         end
52
53         function epuck = Epuck(x, y, theta, v_old, Ts)
```

```
54
55              % Stato iniziale
56              epuck.state = [x, y, theta]';
57
58              % Stato linearizzato
59              epuck.linState = [x, v_old*cos(theta), y, v_old*sin(theta)]';
60
61              % Velocità iniziale
62              epuck.v_old = v_old;
63
64              % Tempo di campionamentos
65              epuck.Ts = Ts;
66
67          end
68
69          function step(epuck, ax, ay)
70
71              % Anello linearizzante
72              [v, w] = epuck.feedbackLinearization(ax, ay);
73
74              % Aggiornamento dello stato
75              [¬, Y1] = ode23(@Epuck_ode2,[0, epuck.Ts], epuck.state, [], [w...
                    ,v]');
76              epuck.state = Y1(end, 1:3)';
77
78              % Aggiornamento stato non lineare
79              epuck.linState = [epuck.state(1); v*cos(epuck.state(3)); ...
                    epuck.state(2); v*sin(epuck.state(3))];
80
81          end
82
83          function h = plot(epuck)
84              h = plot(epuck.position(1), epuck.position(2), epuck.color);
85          end
86
87      end
88
89      methods (Access = protected)
90
91           % Implementazione anello linearizzante
92           function [v, w] = feedbackLinearization(epuck, ax, ay)
93
94                [¬, Y1] = ode23(@Epuck_ode, [0, epuck.Ts], [epuck.state; ...
                      epuck.v_old], [], [ax, ay]');
95                theta_new = Y1(end, 3);
96                omega10 = (-sin(epuck.state(3))*ax + cos(epuck.state(3))*ay) ...
                      / epuck.v_old;
97                w = 0.5 * (omega10 + (-sin(theta_new)*ax + cos(theta_new) * ...
                      ay) / Y1(end, 4));
```

```
98              v = (epuck.v_old + Y1(end, 4)) / 2;
99              epuck.v_old = Y1(end, 4);
100
101         end
102
103
104     end
105
106 end
```

A.2 Controllo di un *epuck* reale: EpuckReal

La classe **EpuckReal** estende la precedente **Epuck** aggiungendo le funzionalità necessarie alla comunicazione con l'*epuck*. Il grande vantaggio ottenuto da questa struttura è che per passare dalla simulazione alla prova sperimentale è sufficiente eseguire una chiamata al metodo dedicato alla connessione, tutto il resto è gestito internamente fornendo quindi all'utente un livello di astrazione elevato. Segue il codice relativo all'implementazione di tale classe, opportunamente commentato per chiarire le funzionalità e definizioni dei vari metodi e attributi pubblici della classe.

```
1  classdef EpuckReal < Epuck
2  %
3  %   -- Classe EpuckReal --
4  %   Questa classe eredita da Epuck e gestisce la comunicazione con
5  %   un epuck reale
6  %
7  %   # COSTRUTTORE: EpuckReal(x, y, theta, v_old, Ts)
8  %       * x, y:  posizione inziale
9  %       * theta: orientamento inziale
10 %       * v_old: velocità iniziale
11 %       * Ts: tempo di campionamento
12 %
13 %   # ATTRIBUTI PUBBLICI
14 %       * position: attuale posizione
15 %       * state: stato (non lineare)
16 %       * linState: stato linearizzato
17 %       * color: colore con cui viene stampata la posizione attraverso il
18 %                metodo plot. Rispetta la sintassi di plot ('r*', ...)
19 %       * saturation: saturazione in step/s della velocità di ciascuna
20 %                     ruota
21 %
22 %
```

```
23 %    # METODI PUBBLICI
24 %        * step(ax, ay): applica all'agente un'accelerazione [ax, ay]
25 %        * plot(): stampa la posizione
26 %        * connect(serialPortName): si connette all'agente utilizzando la
27 %                                   porta seriale chiamata 'serialPortName'
28 %        * fakeStep(ax, ay): questo metodo è uguale a step(ax, ay), ma non
29 %                            invia la velocità all'agente. Può essere
30 %                            utilizzato per evitare problemi a bassa
31 %                            velocità.
32 %        * close(): chiude la comunicazione con la porta seriale
33 %
34 %
35
36      properties
37
38          saturation = 300;
39
40      end
41
42      properties (SetAccess = protected)
43
44          port = 0;
45          cleanUp = [];
46
47      end
48
49
50      methods
51
52          function epuck = EpuckReal(x, y, theta, v_old, Ts)
53
54              epuck = epuck@Epuck(x, y, theta, v_old, Ts);
55
56          end
57
58          function connect(epuck, serialPortName)
59
60              % Connetti
61              epuck.port = serial(serialPortName, 'BaudRate', 115200,'...
                    inputBuffersize',4096,'OutputBufferSize',4096,'ByteOrder','...
                    littleendian');
62              fopen(epuck.port);
63              c = onCleanup(@()sendData(0,0, epuck.port));
64              c1 = onCleanup(@()fclose( epuck.port));
65              %fread(EpuckPort, EpuckPort.BytesAvailable); %Ripulisce la ...
                    porta seriale.
66              disp([serialPortName ' connected']);
67
68              %Reset della velocità del robot, per sicurezza.
```

```
69          sendData(0,0, epuck.port);
70
71          epuck.cleanUp = [epuck.cleanUp c c1];
72
73      end
74
75
76      function step(epuck, ax, ay)
77
78          % Linearizzazione
79          [v, w] = epuck.feedbackLinearization(ax, ay);
80
81          % Conversione input
82          [stepSpeedR, stepSpeedL] = epuck.convertInput(v, w);
83
84          % Invio velocità
85          epuck.sendData(stepSpeedR, stepSpeedL);
86
87          % Aggiornamento velocità lineare
88          epuck.v_old = v;
89
90          % Aggiornamento stato lineare
91          epuck.linState = [epuck.state(1), epuck.v_old*cos(epuck.state...
                (3)), epuck.state(2), epuck.v_old*sin(epuck.state(3))]';
92
93      end
94
95      function fakeStep(epuck, ax, ay)
96
97          % Linearizzazione
98          [v, ¬] = epuck.feedbackLinearization(ax, ay);
99
100         % Invio velocità
101         epuck.sendData(0, 0);
102
103         % Aggiornamento velocità lineare
104         epuck.v_old = v;
105
106         % Aggiornamento stato lineare
107         epuck.linState = [epuck.state(1), epuck.v_old*cos(epuck.state...
                (3)), epuck.state(2), epuck.v_old*sin(epuck.state(3))]';
108
109     end
110
111
112     function close(epuck)
113
114         epuck.sendData(0, 0);
115         fclose(epuck.port);
```

```
116
117         end
118
119     end
120
121     methods (Access = protected)
122
123         function [v, w] = feedbackLinearization(epuck, ax, ay)
124
125             if epuck.v_old < -1e-3 || epuck.v_old > 1e-3
126
127                 vp = ax * cos(epuck.state(3)) + ay * sin(epuck.state(3));
128                 v = epuck.v_old + vp * epuck.Ts;
129                 w = (-ax * sin(epuck.state(3)) + ay * cos(epuck.state(3)))...
                        / epuck.v_old;
130
131             else
132                 v = epuck.v_old + 1e-3 * epuck.Ts;
133                 w = 1e-3;
134             end
135
136         end
137
138         function sendData(epuck, stepSpeedR, stepSpeedL)
139
140             % Controllo della saturazione per evitare overflow
141             if (stepSpeedL > epuck.saturation)
142                 stepSpeedL = epuck.saturation;
143             end
144             if (stepSpeedL < -epuck.saturation)
145                 stepSpeedL = -epuck.saturation;
146             end
147             if (stepSpeedR > epuck.saturation)
148                 stepSpeedR = epuck.saturation;
149             end
150             if (stepSpeedR < -epuck.saturation)
151                 stepSpeedR = epuck.saturation;
152             end
153
154             % Impacchetta la velocità a 16bit
155             data = [stepSpeedL stepSpeedR];
156             int16(data);
157
158             % Il carattere 's' informa l'epuck che i dati successivi sono
159             % la velocià
160             fwrite(epuck.port,'s', 'char');
161             fwrite(epuck.port, 2*2, 'uint16');
162             fwrite(epuck.port, data, 'int16');
163         end
```

```
164
165         function [stepSpeedR, stepSpeedL] = convertInput(epuck, v, w)
166
167             % Step al giro
168             kk = 500/pi; % step/rad
169
170             % Velocità lineare ruote
171             vR = v + epuck.E * w / 2;
172             vL = v - epuck.E * w / 2;
173
174             % Velocità angolare
175             wL = vL / epuck.R;
176             wR = vR / epuck.R;
177
178             % Velocità in step/s
179             stepSpeedL = round(kk * wL);
180             stepSpeedR = round(kk * wR);
181
182
183         end
184
185
186
187     end
188
189 end
```

A.3 Gestione della telecamera e analisi delle immagini: CameraPosition

La classe **CameraPosition** gestisce la comunicazione con la telecamera utilizzata per la rilevazione di posizione e orientamento degli agenti e implementa l'algoritmo di analisi dell'immagine descritto nella Sezione 2.3. Essa permette inoltre di visualizzare un video in tempo reale del piano di lavoro in cui vengono sovrapposti i risultati dell'elaborazione, che, se desiderato, può essere salvato alla fine dell'esecuzione.

```
1 classdef CameraPosition < handle
2 %
3 %   -- Classe CameraPosition --
4 %   Questa classe gestisce la comunicazione con la telecamera e
5 %   l'elaborazione delle immagini acquisite per la rilevazione della
6 %   posizione degli agenti nel piano di lavoro
7 %
```

```
8  %   # COSTRUTTORE: CameraPosition(videoFlag, saveFlag, plotResultsFlag)
9  %       * videoFlag: boolean. Se vale 1 viene visualizzato il video con le
10 %                    immagini acquisite ed elaborate
11 %       * saveFlag:  boolean. Se vale 1 viene salvato il video con le
12 %                    immagini acquisite ed elaborate
13 %       * plotResultsFlag: boolean. Se vale 1 vengono stampati nella
14 %                          command window i vari passaggi
15 %                          nell'inizializzazione dell'istanza
16 %
17 %   # ATTRIBUTI PUBBLICI
18 %       * filename: nome del file in cui viene salvate il video
19 %                   (default 'VideoPosition.avi')
20 %       * environment: una lista di elementi da visualizzare in ogni ...
       frame.
21 %                      Ogni elemento nella lista (matlab cell) deve essere
22 %                      strutturato come:
23 %                          x0  ------------  xn
24 %                          y0  ------------  yn
25 %                      in cui (xi, yi) è un punto dell'elemento
26 %
27 %
28 %   # METODI PUBBLICI
29 %       * setDimension(xCm, yCm): metodo per impostare le dimensioni in cm
30 %                                 dello spazio di lavoro osservato
31 %
32 %       * [x, y, theta] = getPosition(robots): restituisce le posizioni e
33 %       orientamenti degli agenti. Il parametro 'robots' indica quanti
34 %       agenti sono presenti. I colori DEVONO rispettare la seguente
35 %       convenzione:
36 %                -> robots = 1: rosso
37 %                -> robots = 2: rosso (agente 1), blu (agente 2)
38 %                -> robots = 3: rosso (1), arancione (2), blu (3)
39 %
40 %       * close(): chiude la comunicazione con la telecamera
41 %
42 %
43 %
44     properties
45         % Nome del file in cui salvare il video
46         filename = 'VideoPosition.avi';
47
48         % Ambiente
49         environment = {};
50
51     end
52
53     properties (Access = private)
54
55         camera = [];
```

```
56          hcornerdet = [];
57          hdrawmarkersCorner = [];
58          hdrawmarkersCenter = [];
59          hdrawmarkersEnvironnment = [];
60          hdrawLine = [];
61          hText = [];
62          hvideoplayer = [];
63          videoFWriter = [];
64
65          videoFlag = 0;
66          saveFlag = 0;
67
68          xCm = 114.7;
69          yCm = 65.7;
70
71          cleanUp = [];
72
73      end
74
75      properties (Access = public)
76
77
78
79      end
80
81      methods
82
83          function obj = CameraPosition(videoFlag, saveFlag, plotResultsFlag...
                )
84
85              % Flags for printing, showing video and saving file
86              plotFlag = 1;
87              if nargin == 3
88                  plotFlag = plotResultsFlag;
89              end
90              obj.videoFlag = videoFlag;
91              obj.saveFlag = saveFlag;
92
93              % Find and open the right camera
94              devices = imaqhwinfo('macvideo');
95              for deviceIndex = 1:length(devices.DeviceIDs)
96
97                  if ¬isempty(strfind(devices.DeviceInfo(deviceIndex)...
                        .DeviceName, 'Microsoft'))
98                      break;
99                  end
100
101             end
102             obj.camera = videoinput('macvideo', deviceIndex);
```

```
103             obj.cleanUp = onCleanup(@() stop(obj.camera));
104             set(obj.camera,'TriggerRepeat', inf);
105             set(obj.camera,'FramesPerTrigger', 1);
106             triggerconfig(obj.camera, 'manual');
107             set(obj.camera,'ReturnedColorSpace','rgb');
108             obj.camera.ROIPosition = [0 65 640 365];
109           %  set(obj.camera,'FramesAcquiredFcnCount', 1);
110           %  set(obj.camera,'FramesAcquiredFcn', {'flushCamera'});
111             start(obj.camera);
112             if plotFlag
113                 disp([imaqhwinfo(obj.camera, 'DeviceName') ' connected']);
114             end
115
116             % Initialize vision toolbox objects
117             % Corner detector
118             if plotFlag
119                 fprintf('Allocating corner detector...');
120             end
121             obj.hcornerdet = vision.CornerDetector('Method', 'Local ...
                    intensity comparison (Rosen & Drummond)', '...
                    MaximumAngleThreshold', '157.5', 'MaximumCornerCount', 3, '...
                    NeighborhoodSize', [31 31]);
122             if plotFlag
123                 fprintf('done\n');
124             end
125             % Video
126             if (videoFlag || saveFlag)
127                 if plotFlag
128                     fprintf('Allocating object inserters...');
129                 end
130                 obj.hdrawmarkersEnvironnment = vision.MarkerInserter('...
                        Shape', 'Circle', 'BorderColorSource', 'Property', '...
                        BorderColor', 'Custom', 'CustomBorderColor', [255 0 0])...
                        ;
131                 obj.hdrawmarkersCorner =  vision.MarkerInserter('Shape', '...
                        Circle', 'BorderColorSource', 'Property', 'BorderColor'...
                        , 'Custom', 'CustomBorderColor', [255 0 255]);
132                 obj.hdrawmarkersCenter = vision.MarkerInserter('Shape', '...
                        Square', 'BorderColorSource', 'Property','BorderColor',...
                         'Custom', 'CustomBorderColor', [0 0 0]);
133                 obj.hdrawLine = vision.ShapeInserter('Shape', 'Lines');
134                 obj.hText = vision.TextInserter('Text', ['Robot 1 (r): (%3...
                        .1f, %3.1f, %3.1f)' char(10) 'Robot 2 (b): (%3.1f, %3...
                        .1f, %3.1f)' char(10) 'Robot 3 (a): (%3.1f, %3.1f, %3...
                        .1f)'], 'Location', [5 300], 'Color', [0 0 0]);
135                 if plotFlag
136                     fprintf('done\n');
137                 end
138             end
```

```
139
140             if videoFlag
141                 if plotFlag
142                     fprintf('Allocating video player...');
143                 end
144                 obj.hvideoplayer = vision.VideoPlayer('Position', [100 100...
                        700 400]);
145                 if plotFlag
146                     fprintf('done\n');
147                 end
148             end
149
150             % Salvataggio
151             if saveFlag
152                 if exist(obj.filename, 'file')
153                     delete(obj.filename)
154                     if plotFlag
155                         disp(['Old ' obj.filename ' deleted']);
156                     end
157                 end
158                 if plotFlag
159                     fprintf('Allocating video writer...');
160                 end
161                 obj.videoFWriter = vision.VideoFileWriter(obj.filename);
162                 if plotFlag
163                     fprintf('done\n');
164                 end
165             end
166         end
167
168         function setDimension(obj, xCm, yCm)
169
170             obj.xCm = xCm;
171             obj.yCm = yCm;
172
173         end
174
175         function [x, y, theta] = getPositions(obj, robots)
176
177             % Acquisisci e ruota l'immagine
178             trigger(obj.camera);
179             imageOriginal = getdata(obj.camera, 1, 'uint8');
180             imageRot = imrotate(imageOriginal, 180);
181
182             % Estrai la dimensione in pixel dell'immagine
183             xPixel = size(imageOriginal, 2);
184             yPixel = size(imageOriginal, 1);
185
186             % Alloca spazio per i layer degli agenti
```

```
187             % Itr(:,:,1) = Red
188             % Itr(:,:,2) = Blue
189             % Itr(:,:,3) = Orange
190             Itr = zeros(size(imageRot));
191
192             if obj.saveFlag || obj.videoFlag
193                 % Copia l'immagine originale in modo da poter aggiungere i
194                 % marker delle posizioni rilevate
195                 Ishow = imageRot;
196
197                 % Aggiungi l'ambiente (se impostato)
198                 if ¬isempty(obj.environment)
199                     EnvPts = [];
200                     for h = 1:length(obj.environment)
201                         % Estraggo i punti
202                         yO = cell2mat(obj.environment(h));
203                         EnvPts = [EnvPts yO];
204                     end
205                     % Converto i punti in pixel
206                     EnvPts(1, :) = EnvPts(1, :) * xPixel / obj.xCm;
207                     EnvPts(2, :) = yPixel - EnvPts(2, :) * yPixel / ...
                            obj.yCm;
208                     Ishow = step(obj.hdrawmarkersEnvironnment, Ishow, ...
                            uint32(EnvPts'));
209
210                 end
211
212             end
213
214             % Alloca spazio per le posizioni e orientamenti
215             x = zeros(3, 1);
216             y = zeros(3, 1);
217             theta = zeros(3, 1);
218
219             % Separa i livelli RGB
220             imR = imageRot(:,:,1) - rgb2gray(imageRot);
221             imG = im2single(imageRot(:,:,2));
222             imB = imageRot(:,:,3) - rgb2gray(imageRot);
223
224
225             if robots == 3  % ------> CASO 3 AGENTI
226                 % Separa il rosso dall'arancione utilizzando il layer G
227                 imR = imfilter(imR, fspecial('gaussian'));
228                 imR_bw = im2bw(imR, 0.2);
229                 imR_bw = bwareaopen(imR_bw, 50);
230                 imclearborder(imR_bw);
231                 [L, ¬] = bwlabel(imR_bw);
232                 imG = im2single(imG);
233                 imR_bw = im2single(imR_bw);
```

```
234             if mean(nonzeros(imG .* im2single((L == 1)))) < mean(...
                    nonzeros(imG .* im2single((L == 2))))
235                 imR = imR_bw .* im2single((L == 1));
236                 imO = imR_bw .* im2single((L == 2));
237             else
238                 imR = imR_bw .* im2single((L == 2));
239                 imO = imR_bw .* im2single((L == 1));
240             end
241             Itr(:,:,1) = imR;
242             Itr(:,:,3) = imO;
243
244             % Estrai il blu
245             imB = im2bw(imB, 0.2);
246             imB = bwareaopen(imB, 50);
247             Itr(:,:,2) = imclearborder(imB);
248         elseif robots == 2 % ------> CASO 2 AGENTI
249             % Non c'è l'arancione perciò il rosso può essere estratto
250             % dal layer R
251             imR = imfilter(imR, fspecial('gaussian'));
252             imR_bw = im2bw(imR, 0.2);
253             imR_bw = bwareaopen(imR_bw, 50);
254             Itr(:,:,1) = imR_bw;
255
256             % Estrai il blu
257             imB = im2bw(imB, 0.2);
258             imB = bwareaopen(imB, 50);
259             Itr(:,:,2) = imclearborder(imB);
260         else
261             % C'è solo il rosso. No problem.
262             imR = imfilter(imR, fspecial('gaussian'));
263             imR_bw = im2bw(imR, 0.2);
264             imR_bw = bwareaopen(imR_bw, 50);
265             Itr(:,:,1) = imR_bw;
266         end
267
268         for i = 1:robots %(rosso, blu, arancione)
269
270             % Corner Detection
271             cornerPoints = step(obj.hcornerdet, im2single(Itr(:,:,i)))...
                    ;
272
273             % Stampa i marker
274             if obj.videoFlag || obj.saveFlag
275                 Ishow = step(obj.hdrawmarkersCorner, Ishow, ...
                        cornerPoints);
276             end
277
278             % Controlla che i corner siano effettivamente 3
279             if size(cornerPoints, 1) < 3
```

```
280                     disp(['Error: less than 3 corners in layer ' num2str(i...
                            )]);
281                     % TODO: la gestione di questo caso può essere
282                     % migliorata (Kalman ?)
283                     x(i) = 0;
284                     y(i) = 0;
285                     theta(i) = 0;
286                     continue;
287                 end
288
289                 % Trova i due punti più vicini (C1 e C2) che rappresentano
290                 % la base. Il restante sarà la punta (V)
291                 cornerPoints = double(cornerPoints);
292                 p1 = cornerPoints(1, :);
293                 p2 = cornerPoints(2, :);
294                 p3 = cornerPoints(3, :);
295                 [~, idx] = min([norm(p1-p2), norm(p1-p3), norm(p2-p3)]);
296                 if idx == 1
297                     V = p3;
298                     C1 = p1;
299                     C2 = p2;
300                 elseif idx == 2
301                     V = p2;
302                     C1 = p1;
303                     C2 = p3;
304                 else
305                     V = p1;
306                     C1 = p2;
307                     C2 = p3;
308                 end
309
310                 % Punto medio della base
311                 Cm = (C1 + C2) / 2;
312
313                 % Il centro dell'agente si trova al 100*h per cento dal
314                 % centro Cm verso V
315                 h = 0.5;
316                 O = Cm + h * (V - Cm);
317                 x(i) = O(1);
318                 y(i) = O(2);
319
320                 % La direzione è definita dal vettore (V-Cm). La ...
                        coordinata
321                 % y deve essere invertita poichè le immagini hanno un
322                 % sistema di coordinate diverso.
323                 theta(i) = wrapTo2Pi(atan2(-(V(2) - Cm(2)), V(1) - Cm(1)))...
                        ;
324
325                 % Visualizza il centro e la direzione
```

```
326             if obj.videoFlag || obj.saveFlag
327                 Ishow = step(obj.hdrawmarkersCenter, Ishow, uint32([x ...
                        y]));
328                 Ishow = step(obj.hdrawLine, Ishow, uint32([x(i) y(i) V...
                        (1) V(2)]));
329             end
330
331         end
332
333         % Conversione px -> cm
334         x = x * obj.xCm / xPixel;
335         y = (yPixel - y) * obj.yCm / yPixel;
336
337         % Stampa i risultati nel video
338         if obj.videoFlag || obj.saveFlag
339            % Ishow = step(obj.hText, Ishow, [x(1) y(1) 180/pi*theta(1)...
                    x(2) y(2) 180/pi*theta(2) x(3) y(3) 180/pi*theta(3)]);
340         end
341
342         % Visualizza il video
343         if obj.videoFlag
344             step(obj.hvideoplayer, Ishow);
345         end
346
347         % Salva il file
348         if obj.saveFlag
349             step(obj.videoFWriter, Ishow);
350         end
351
352         % Restituisci le posizione e orientamenti richiesti
353         x = x(1:robots);
354         y = y(1:robots);
355         theta = theta(1:robots);
356
357     end
358
359     function close(obj)
360
361         stop(obj.camera);
362         if ~isempty(obj.hcornerdet)
363             release(obj.hcornerdet);
364         end
365         if ~isempty(obj.hdrawmarkersEnvironnment)
366             release(obj.hdrawmarkersEnvironnment);
367         end
368         if ~isempty(obj.hdrawmarkersCorner)
369             release(obj.hdrawmarkersCorner);
370         end
371         if ~isempty(obj.hdrawmarkersCenter)
```

```
372                 release(obj.hdrawmarkersCenter);
373             end
374             if ¬isempty(obj.hdrawLine)
375                 release(obj.hdrawLine);
376             end
377             if ¬isempty(obj.hText)
378                 release(obj.hText);
379             end
380             if ¬isempty(obj.hvideoplayer)
381                 release(obj.hvideoplayer);
382             end
383             if ¬isempty(obj.videoFWriter)
384                 release(obj.videoFWriter);
385             end
386
387         end
388
389     end
390
391 end
```

A.4 Algoritmo di controllo robusto: MPCTrajectory

La classe **MPCTrajectory** implementa la struttura gerarchica di controllo predittivo robusto introdotta nel Capitolo 4. La generazione della traiettoria viene realizzata mediante il problema di ottimizzazione (4.52) a cui è possibile aggiungere vincoli a piacere in modo da poter soddisfare diversi requisiti nella soluzione di questo problema. Per facilitare la comprensione e l'utilizzo di illustra il processo di inizializzazione di un'istanza della classe in esame:

1. Si crea l'istanza utilizzando il costruttore `MPCTrajectory(As, Bs, Cs)`
2. Si configura ciascun livello dell'architettura di controllo utilizzando i medoti:
   - `setupRobustMPCLayer(mpc, N, Qs, Rs, Xs, Us, DeltaZs, Ws)`: tube-based MPC
   - `setupStateInputReferenceLayer(mpc, eigenvalues)`: generazione delle traiettorie di riferimento dello stato e dell'ingresso

- `setupTrajectoryLayer(mpc, gamma, T, epsilon)`: generazione della traiettoria per le uscite secondo il problema (4.52)

3. Inizializzazione degli algoritmi utilizzando il metodo `initialize(y0)` in cui `y0` è la posizione iniziale dell'agente

Alternativamente, se è stata già creata un'istanza e se ne vogliono creare di identiche, anzichè rieseguire i passi 1. e 2. appena descritti è sufficiente chiamare il metodo `copy` che ritorna la copia voluta. Si è scelto di implementare questa funzionalità poichè tali passi sono computazionalmente dispendiosi e quindi effettuare una copia permette di risparmiare tempo.

```
1  classdef MPCTrajectory < handle
2  %
3  %   -- Classe MPCTrajectory --
4  %   Questa classe gestisce tutta la struttura gerarchica di controllo
5  %   robusto, implementando il problema (4.52) per la generazione della
6  %   traiettoria, permettendo però di aggiungere vincoli non lineari.
7  %   Processo di inizializzazione: Costruttore -> setup* -> initialize
8  %   oppure, a partire da un'altra istanza: copy -> inizialize
9  %
10 %
11 %
12 %   # COSTRUTTORE: MPCTrajectory(As, Bs, Cs)
13 %       * As: partizione della matrice dinamica A
14 %             riferita ad un una direzione
15 %       * Bs: partizione della matrice d'ingresso B
16 %             riferita ad un una direzione
17 %       * Cs: partizione della trasformazione di uscita
18 %             riferita ad un una direzione
19 %
20 %   # ATTRIBUTI PUBBLICI
21 %       * N: lunghezza orizzonte di predizione
22 %       * y_setpoint: posizione del goal
23 %       * epsilon: dimensione della bolla Beta_{epsilon}(0)
24 %       * positionReference: attuale posizione di riferimento da inseguire
25 %       * direction: direzione della traiettoria calcolata utilizzando gli
26 %           ultimi due punti generati.
27 %
28 %
29 %   # METODI PUBBLICI
30 %       * setupTrajectoryLayer(mpc, gamma, T, epsilon): configurazione del
31 %         livello di generazione della traiettoria utilizzando i ...
       parametri
32 %         passati. Per il significato di tali parametri si veda il ...
       problema
```

```
33 %          di ottimizzazione (4.52)
34 %
35 %       * setupStateInputReferenceLayer(mpc, eigenvalues): configurazione
36 %          del livello di generazione della traiettoria per lo stato e
37 %          l'ingresso (Sezione 4.4.1). Il parametro 'eigenvalues' contiene
38 %          gli autovalori in anello chiuso di tale livello.
39 %
40 %       * setupRobustMPCLayer(mpc, N, Qs, Rs, Xs, Us, ΔZs, Ws):
41 %          configurazione del livello Tube-based MPC (Sezione 4.4.2). Si
42 %          rimanda a tale sezione per il significato dei parametri
43 %          (il pedice 's' indica che sono insiemi riferiti alla partizione
44 %           del sistema in una sola direzione).
45 %
46 %       * initialize(y0): inizializza tutti i livelli di controllo
47 %          impostando 'y0' come posizione corrente dell'agente.
48 %
49 %       * resetTrajectory(Pa): resetta tutte le predizioni generate,
50 %           reinizializzando i livelli di controllo nella posizione ...
   corrente.
51 %
52 %       * [u, y_kplusN] = step(mpc, x, f_contraints_L1):
53 %           esegue un'iterazione di tutti i livelli. Parametri;
54 %              -> x: stato del sistema sotto controllo
55 %              -> f_contraints_L1: vincoli aggiuntivi da imporre al
56 %                   problema di generazione della traiettoria. Devono
57 %                   essere matlab function handler, coerentemente con il
58 %                   parametro NONLCON della funzione fmincon.
59 %           Restituisce l'ingresso da applicare al sistema.
60 %
61 %       * u = stepRobustLayer(mpc, x): esegue un'iterazione del solo
62 %            livello MPC Tube-Based. Da utilizzare sono per particolari
63 %            applicazioni (Si veda la classe MPCCustomTrajectory).
64 %
65 %       * mpc = copy(): restituisce una copia dell'istanza che lo chiama.
66 %            Questo metodo può essere usato quando si creano diverse
67 %            istanze del sistema di controllo per evitare di dover
68 %            ricomputare tutti i set ogni volta.
69 %
70 %
71 %
72     properties (SetAccess = protected)
73
74         % System
75         As
76         Bs
77         Cs
78         A
79         B
80         C
```

```
81
82          % Other
83          opt = optimset('Display','off');
84          oldTheta = 0;
85
86          % L1: trajectory layer
87          S_traj = []; % Cost function
88          gamma = 0;
89          T = zeros(2);
90          y_tilde = [];
91          H_eps = [];
92          L_eps_unitary = [];
93          eps_side = 100;
94
95
96          % L2: state, input references
97          Kx = [];
98          Ke = [];
99          x_tilde;
100         u_tilde;
101         e_tilde;
102
103         % L3: robust MPC layer
104         K
105         A_big;
106         Q_big;
107         S_robust;
108         EPS_H = [];
109         EPS_L = [];
110         ΔZ_H = [];
111         ΔZ_L = [];
112         H_robust; % 2, 3, 1, 45
113         L_23_robust; % 2, 3
114         L_45_a;%
115         L_45_b;% needed to compute L_45
116
117     end
118
119     properties (SetAccess = public)
120
121         % L1
122         N
123         y_setpoint = [];
124         epsilon = 1;
125
126     end
127
128     methods
129         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        %           CONSTRUCTOR
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        function mpc = MPCTrajectory(As, Bs, Cs)

            mpc.As = As;
            mpc.Bs = Bs;
            mpc.Cs = Cs;
            mpc.A = blkdiag(As, As);
            mpc.B = blkdiag(Bs, Bs);
            mpc.C = blkdiag(Cs, Cs);

        end



        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % L1:       TRAJECTORY
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        function setupTrajectoryLayer(mpc, gamma, T, epsilon)

            % Cost Function
            I = [eye(2); eye(2)];
            mpc.S_traj = I' * [gamma*eye(2) zeros(2); zeros(2) T] * I;
            mpc.T = T;
            mpc.gamma = gamma;

            % Beta epsilon unitary
            BeVert_int = zeros(mpc.eps_side, 2);
            for i = 1:mpc.eps_side
                BeVert_int(i, :) = [cos(pi/2+2*pi*(i-1)/mpc.eps_side) sin(...
                    pi/2+2*pi*(i-1)/mpc.eps_side)];
            end
            [mpc.H_eps, mpc.L_eps_unitary] = vert2lcon(BeVert_int);

            % epsilon
            mpc.epsilon = epsilon;

        end

        function y_t = positionReference(mpc)

            y_t = mpc.y_tilde((end-size(mpc.C, 1)+1):end);

        end

        function resetTrajectory(mpc, resetPosition)

            mpc.initialize(resetPosition);

```

```
178         end
179
180         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
181         % L2:   STATE, INPUT REFERENCE
182         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
183         function setupStateInputReferenceLayer(mpc, eigenvalues)
184             Kxe = place([mpc.A zeros(4,2) ; -mpc.C eye(2)] , [mpc.B ; ...
                    zeros(2,2)] , eigenvalues);
185             mpc.Kx = Kxe(:, 1:size(mpc.A, 1));
186             mpc.Ke = Kxe(:, (end-size(mpc.C)+1):end);
187         end
188
189
190         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
191         % L3:   ROBUST MPC LAYER
192         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
193         function setupRobustMPCLayer(mpc, N, Qs, Rs, Xs, Us, ΔZs, Ws)
194
195             mpc.N = N;
196
197             % Compute P and K using LQ
198             [Ks, Ps] = dlqr(mpc.As, mpc.Bs, Qs, Rs, []);
199
200             % Compute full system
201             mpc.K = blkdiag(Ks, Ks);
202             P = blkdiag(Ps, Ps);
203             Q = blkdiag(Qs, Qs);
204             R = blkdiag(Rs, Rs);
205
206
207             % Italic Matrices (prediction) mathcal{}
208             A_italic = zeros((N+1) * size(mpc.A, 1), size(mpc.A, 2));
209             B_italic = zeros(N * size(mpc.B, 1), N * size(mpc.B, 2));
210             Q_italic = zeros((N+1) * size(Q));
211             R_italic = zeros(N * size(R));
212             % A_italic
213             for i = 1:(N+1)
214                 A_italic(((i-1)*size(mpc.A, 1) + 1):(i*size(mpc.A, 1)), :)...
                        = mpc.A^(i-1);
215             end
216             % B_italic
217             for i = 2:(N+1)
218                 for j = 1:i-1
219                     B_italic(((i-1)*size(mpc.B, 1) + 1):(i*size(mpc.B, 1))...
                            , ((j-1)*size(mpc.B, 2) + 1):(j*size(mpc.B, 2))) = ...
                            (mpc.A^(i-j-1))*mpc.B;
220                 end
221             end
222             % Q_italic
```

```
223             for i = 1:N
224                 Q_italic(((i-1)*size(Q, 1) + 1):(i*size(Q, 1)), ((i-1)*...
                        size(Q, 2) + 1):(i*size(Q, 2))) = Q;
225             end
226             Q_italic((N*size(Q, 1) + 1):((N+1)*size(Q, 1)), (N*size(Q, 2) ...
                    + 1):((N+1)*size(Q, 2))) = P;
227             % R_italic
228             for i = 1:N
229                 R_italic(((i-1)*size(R, 1) + 1):(i*size(R, 1)), ((i-1)*...
                        size(R, 2) + 1):(i*size(R, 2))) = R;
230             end
231
232             % Big matrices mathbb{}
233             mpc.A_big = [A_italic B_italic; zeros(size(mpc.B, 2)*N, size(...
                    A_italic, 2)) eye(size(mpc.B, 2)*N)];
234             mpc.Q_big = blkdiag(Q_italic, R_italic);
235
236             % Cost Function
237             mpc.S_robust = mpc.A_big' * mpc.Q_big * mpc.A_big;
238
239             %%%%%% Vincoli
240             % Epsilon
241             EPS_verts = rakovic(Ws, (mpc.As - mpc.Bs*Ks), 0.001);
242             [EPS_Hs, EPS_Ls] = vert2lcon(EPS_verts);
243             mpc.EPS_H = blkdiag(EPS_Hs, EPS_Hs);
244             mpc.EPS_L = [EPS_Ls; EPS_Ls];
245             H1 = -mpc.EPS_H * [eye(size(mpc.A)) zeros(size(mpc.A, 1), size...
                    (mpc.B, 2)*N)];
246
247             % X_hat
248             if(~isempty(Xs))
249                 [X_Hs, X_Ls] = vert2lcon(Xs);
250                 [X_hat_Hs, X_hat_Ls] = pontrydiff([X_Hs, X_Ls], [EPS_Hs, ...
                        EPS_Ls]);
251                 % Two directions
252                 X_hat_H = blkdiag(X_hat_Hs, X_hat_Hs);
253                 X_hat_L = [X_hat_Ls; X_hat_Ls];
254
255                 X_hat_H_big = zeros(N*size(X_hat_H));
256                 for i = 1:N
257                     X_hat_H_big(((i-1)*size(X_hat_H, 1) + 1):(i*size(...
                            X_hat_H, 1)), ((i-1)*size(X_hat_H, 2) + 1):(i*size(...
                            X_hat_H, 2))) = X_hat_H;
258                 end
259                 A_italic_bar = A_italic(1:(end-size(mpc.A, 1)), :);
260                 B_italic_bar = B_italic(1:(end-size(mpc.B, 1)), :);
261                 H2 = X_hat_H_big * (A_italic_bar * [eye(size(mpc.A)) zeros...
                        (size(mpc.A, 1), size(mpc.B, 2)*N)] + B_italic_bar * [...
                        zeros(size(mpc.B, 2)*N, size(mpc.A, 1)) eye(size(mpc.B,...
```

```
                2)*N)]);
262             L2 = zeros(N*size(X_hat_L, 1), size(X_hat_L, 2));
263             for i = 1:N
264                 L2(((i-1)*size(X_hat_L, 1)+1):(i*size(X_hat_L, 1)), :)...
                        = X_hat_L;
265             end
266         else
267             H2 = [];
268             L2 = [];
269         end
270
271         % U_hat
272         if(~isempty(Us))
273             [U_Hs, U_Ls] = vert2lcon(Us);
274             [U_hat_Hs, U_hat_Ls] = pontrydiff([U_Hs, U_Ls], [EPS_Hs*Ks...
                    ' EPS_Ls]);
275             U_hat_H = blkdiag(U_hat_Hs, U_hat_Hs);
276             U_hat_L = [U_hat_Ls; U_hat_Ls];
277             U_hat_H_big = zeros(N*size(U_hat_H));
278             for i = 1:N
279                 U_hat_H_big(((i-1)*size(U_hat_H, 1) + 1):(i*size(...
                        U_hat_H, 1)), ((i-1)*size(U_hat_H, 2) + 1):(i*size(...
                        U_hat_H, 2))) = U_hat_H;
280             end
281             H3 = U_hat_H_big * [zeros(size(mpc.B, 2)*N, size(A_italic,...
                     2)) eye(size(mpc.B, 2)*N)];
282             L3 = zeros(N*size(U_hat_L, 1), size(U_hat_L, 2));
283             for i = 1:N
284                 L3(((i-1)*size(U_hat_L, 1)+1):(i*size(U_hat_L, 1)), :)...
                        = U_hat_L;
285             end
286         else
287             H3 = [];
288             L3 = [];
289         end
290
291         % Delta_Z e vincolo finale (k+N)
292         [ΔZ_Hs , ΔZ_Ls] = vert2lcon(ΔZs);
293         mpc.ΔZ_H = blkdiag(ΔZ_Hs, ΔZ_Hs);
294         mpc.ΔZ_L = [ΔZ_Ls; ΔZ_Ls];
295         Hzc = mpc.ΔZ_H * mpc.C;
296         Hze = zeros(N*size(Hzc));
297         for i = 1:N
298             Hze(((i-1)*size(Hzc, 1) + 1):(i*size(Hzc, 1)), ((i-1)*size...
                    (Hzc, 2) + 1):(i*size(Hzc, 2))) = Hzc;
299         end
300         Hze = blkdiag(Hze, mpc.EPS_H);
301         H45 = Hze*[A_italic B_italic];
302         Lze = repmat(mpc.ΔZ_L, N, 1);
```

```
303             Lze = [Lze; mpc.EPS_L];
304             mpc.L_45_a = Lze;
305             mpc.L_45_b = H45;
306 
307             % Join all contraints
308             mpc.H_robust = [H2; H3; H1; H45];
309             mpc.L_23_robust = [L2; L3];
310 
311         end
312 
313         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
314         %       INIZIALIZER
315         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
316         function initialize(mpc, y0)
317 
318             % Init vectors
319             mpc.y_tilde = repmat(y0, mpc.N, 1);
320             Chi = (eye(size(mpc.A, 1) + size(mpc.C, 1)) - [mpc.A-mpc.B*...
                    mpc.Kx, -mpc.B*mpc.Ke ; -mpc.C eye(size(mpc.C, 1))]) \ [...
                    zeros(size(mpc.A, 1), size(mpc.C, 1)); eye(size(mpc.C, 1))]...
                     * y0;
321             mpc.x_tilde = repmat(Chi(1:size(mpc.A, 1)), mpc.N, 1);
322             mpc.e_tilde = repmat(Chi((end-size(mpc.C, 1)+1):end), mpc.N, ...
                    1);
323             mpc.u_tilde = zeros(mpc.N*size(mpc.B, 2), 1);
324 
325         end
326 
327         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
328         %        STEP
329         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
330         function [u, y_kplusN] = step(mpc, x, f_contraints_L1)
331 
332 
333             % TRAJECTORY
334             % cost function
335             y_tilde_minus1 = mpc.y_tilde((end-size(mpc.C, 1)+1):end, 1);
336             g_traj = -mpc.gamma * y_tilde_minus1 - mpc.T * mpc.y_setpoint;
337             fcost = @(x) L1CostFunction(x, mpc.S_traj, g_traj);
338             % Linear Constraints
339             L_epsilon = mpc.epsilon * mpc.L_eps_unitary + mpc.H_eps * ...
                    y_tilde_minus1;
340             % Optimize
341             y_kplusN = fmincon(fcost, y_tilde_minus1, mpc.H_eps , ...
                    L_epsilon, [], [], [], [], f_contraints_L1, mpc.opt);
342             % Update Trajectory
343             mpc.y_tilde(1:size(mpc.C, 1)) = [];
344             mpc.y_tilde = [mpc.y_tilde; y_kplusN];
345 
```

```
346             % ROBUST LAYER
347             u = mpc.stepRobustLayer(x);
348         end
349 
350         function u = stepRobustLayer(mpc, x)
351 
352             % Get last trajectory point
353             y_kplusN = mpc.y_tilde(1:size(mpc.C, 1));
354 
355             % STATE, INPUT REFERENCE
356             x_tilde_minus1 = mpc.x_tilde((end-size(mpc.A, 1)+1):end, 1);
357             e_tilde_minus1 = mpc.e_tilde((end-size(mpc.C, 1)+1):end, 1);
358             x_kplusN = (mpc.A - mpc.B*mpc.Kx) * x_tilde_minus1 - mpc.B*...
                    mpc.Ke * e_tilde_minus1;
359             e_kplusN = -mpc.C * x_tilde_minus1 + e_tilde_minus1 + y_kplusN...
                    ;
360             u_kplusN = -mpc.Kx * x_tilde_minus1 - mpc.Ke * e_tilde_minus1;
361             % Update
362             mpc.x_tilde(1:size(mpc.A, 1)) = [];
363             mpc.x_tilde = [mpc.x_tilde; x_kplusN];
364             mpc.e_tilde(1:size(mpc.C, 1)) = [];
365             mpc.e_tilde = [mpc.e_tilde; e_kplusN];
366             mpc.u_tilde(1:size(mpc.B, 2)) = [];
367             mpc.u_tilde = [mpc.u_tilde; u_kplusN];
368 
369 
370             % ROBUST MPC
371             % csi opt vector
372             csi_tilde = [mpc.x_tilde(1:size(mpc.A, 1)); mpc.u_tilde];
373             % Linear Constraints
374             L1 = mpc.EPS_L - mpc.EPS_H * x;
375             L45 = mpc.L_45_a + mpc.L_45_b * csi_tilde;
376             L_robust = [mpc.L_23_robust; L1; L45]; % 2, 3, 1, 45
377 
378             % Find optimum
379             g_robust = - mpc.A_big' * mpc.Q_big' * mpc.A_big * csi_tilde;
380             csi_hat = quadprog(mpc.S_robust, g_robust, mpc.H_robust, ...
                    L_robust,[],[],[],[],[], mpc.opt);
381             x_hat = csi_hat(1:size(mpc.A, 1), 1);
382             u_hat = csi_hat((size(mpc.A, 1)+1):(size(mpc.A, 1)+size(mpc.B,...
                     2)), 1);
383             % Robust input
384             u = u_hat - mpc.K * (x - x_hat);
385 
386         end
387 
388         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
389         %         COPY INSTANCE
390         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
391         function newMPC = copy(mpc)
392
393             newMPC = MPCTrajectory(mpc.As, mpc.Bs, mpc.Cs);
394             p = properties(mpc);
395             for i = 1:length(p)
396                 newMPC.(p{i}) = mpc.(p{i});
397             end
398
399         end
400
401         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
402         %         SETTER
403         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
404         function set.y_setpoint(mpc, ysp)
405
406             if size(ysp, 2) > size(ysp, 1)
407                 mpc.y_setpoint = ysp';
408             else
409                 mpc.y_setpoint = ysp;
410             end
411
412         end
413
414         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
415         %         DIRECTION
416         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
417         function theta = direction(mpc)
418
419             % Se i punti sono troppo vicini, ritorna il vecchio valore di
420             % theta
421             yLastLast = mpc.y_tilde(1:2);
422             yLast = mpc.y_tilde(3:4);
423
424             if norm(yLast - yLastLast) < 0.1
425                 theta = mpc.oldTheta;
426             else
427                 theta = wrapTo2Pi(atan2(yLast(2) - yLastLast(2), yLast(1) ...
                        - yLastLast(1)));
428                 mpc.oldTheta = theta;
429             end
430
431         end
432
433     end
434
435 end
```

A.5 Algoritmo di controllo robusto personalizzabile: MPCCustomTrajectory

La classe **MPCCustomTrajectory** eredita da **MPCTrajectory** aggiungendo dei nuovi metodi che permettono un'implementazione personalizzata del livello di generazione della traiettoria. Il processo di inizializzazione è analogo a quello della super classe. La generazione della traiettoria invece deve essere eseguita esternamente alla classe e deve poi essere utilizzato il metodo `stepCustom`, anzichè il classico metodo `step`, come segue:

1. Si genera il nuovo punto della traiettoria, chiamato `y_next`, utilizzando un qualsiasi algoritmo
2. Si chiama il metodo `stepCustom(x, y_next)`, in cui `x` è lo stato dell'agente, che esegue un'iterazione dei livelli di controllo inferiori ritornando l'accelerazione da imporre all'agente

Inoltre è presente il metodo `trajDistanceLinConstraint()` che ritorna il vincolo (6.14) nella forma $Hx \leq L$ che, si ricorda, deve essere implementato indipendente dall'algoritmo utilizzato in modo da poter garantire le proprietà dimostrate.

```
1  classdef MPCCustomTrajectory < MPCTrajectory
2  %
3  %   -- Classe MPCCustomTrajectory --
4  %   Questa classe eredita da MPCTrajectory e permette al'utente di
5  %   implementare soluzioni diverse dal problema (4.52) per la generazione
6  %   della traiettoria.
7  %
8  %
9  %   # COSTRUTTORE: MPCCustomTrajectory(As, Bs, Cs)
10 %       * As: partizione della matrice dinamica A
11 %             riferita ad un una direzione
12 %       * Bs: partizione della matrice d'ingresso B
13 %             riferita ad un una direzione
14 %       * Cs: partizione della trasformazione di uscita
15 %             riferita ad un una direzione
16 %
17 %   # ATTRIBUTI PUBBLICI
18 %
19 %   # METODI PUBBLICI
20 %       * [H_eps, L_eps] = trajDistanceLinConstraint(): restituisce le
21 %           matrici che definiscono il vincolo (6.14) nella forma
```

```
22  %           H_eps*x ≤ L_eps, relativo alla distanza massima fra
23  %           due punti successivi della traiettoria generata che DEVE ...
    essere
24  %           implementato, indipendentemente dalla scelta del problema
25  %           utilizzato per la generazione.
26  %
27  %       * u = stepCustom(x, y_kplusN): esegue un'iterazione dei livelli
28  %           sotto quello della generazione della traiettoria utilizzando
29  %           'y_kplusN' come nuovo punto di quest'ultima (ora generato
30  %           esternamente dall'utente). Restituisce l'ingresso da applicare
31  %           al sistema.
32  %
33  %
34      methods
35
36          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
37          %           CONSTRUCTOR
38          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
39          function mpc = MPCCustomTrajectory(As, Bs, Cs)
40
41              mpc@MPCTrajectory(As, Bs, Cs)
42
43          end
44
45          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
46          % L1:       TRAJECTORY
47          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
48          function setupTrajectoryLayer(mpc, epsilon)
49
50              setupTrajectoryLayer@MPCTrajectory(mpc, 1, 10*eye(2), epsilon)...
                    ;
51
52          end
53
54
55          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
56          %        STEP
57          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
58          function u = stepCustom(mpc, x, y_kplusN)
59
60              % Aggionra orizzonte traiettoria
61              mpc.y_tilde(1:size(mpc.C, 1)) = [];
62              mpc.y_tilde = [mpc.y_tilde; y_kplusN];
63
64              % ROBUST LAYER
65              u = mpc.stepRobustLayer(x);
66
67          end
68
```

```
69          function [H_eps, L_eps] = trajDistanceLinConstraint(mpc)
70
71              % H
72              H_eps = mpc.H_eps;
73
74              % L
75              y_tilde_minus1 = mpc.y_tilde((end-size(mpc.C, 1)+1):end, 1);
76              L_eps = mpc.epsilon * mpc.L_eps_unitary + mpc.H_eps * ...
                    y_tilde_minus1;
77
78
79          end
80
81
82      end
83
84  end
```

Appendice B

Problema del Controllo di Formazione

Si presenta ora l'algoritmo utilizzato per l'implementazione del problema del controllo di formazione introdotto nel Capitolo 5. Per non appesantire il presente capitolo si è scelto di riportare solo il codice per la soluzione del problema descritto nella Sezione 6.2.3 in simulazione. Nei successivi capitoli si fornirà il codice di altri problemi trattati in cui sarà inclusa la gestione della telecamera e dei robot *epuck* in modo da presentare al lettore un esempio di utilizzo di tutte le classi presentate nell'Appendice A. Si noti come l'utilizzo di classi per l'implementazione degli algoritmi permetta di ottenere un codice chiaro, leggibile, facilmente espandibile e riutilizzabile.

```
%% Parametri dell'ambiente.
Δt = 3;              % Tempo totale loop feedbacklin (ogni loop 0.3s)
Tfrac = 10;              % Iterazioni feedback lin
tau = Δt + 0.7;      % Tempo loop MPC
N = 3;
MXITER = 500;

%% Definizione Epuck e stato iniziale
Lepuck = Epuck(44, 34, 0, 0.01, Δt / Tfrac);
Lepuck.color = 'r.';
F1epuck = Epuck(30, 35, 0, 0.01, Δt / Tfrac);
F1epuck.color = 'b.';
F2epuck = Epuck(14, 35, 0, 0.01, Δt / Tfrac);
F2epuck.color = 'g.';

%% Definizione setpoint
```

```
ybarsetpointLeader = [100 130]';
ybarsetpointF1 = [-20 -20]';
ybarsetpointF2 = [0 0]';

%% Definizione dei Set per TubeBased MPC
% Modello Agente parziale
As=[1 tau ; 0 1];
Bs=[0.5*tau^2 ; tau];
Cs=[1 0];
% Peso stato/ingresso
Qs = 5*eye(2);
Rs = 100*eye(1);
% Disturbo
w_arb = 1;
Ws = [1 1; 1 -1; -1,1; -1 -1] * w_arb;
% Vincoli stato
xbar = 200;
ybar = 200;
vbar = 10;
Xs = [xbar, vbar; -xbar, vbar; xbar, -vbar; -xbar, -vbar];
% Vincoli Ingresso
Us = [];
% Vincoli uscita
ΔZs = [-1; 1];

%% Definizione autovalori livello intermedio
autov = [0.3 0.31 0.32 0.33 0.34 0.35];

%% Definizione parametri livello di gen. della traiettoria
gamma = 1;
T = 10 * eye(2);
epsilon = 1.5;

%% Instanze per il controllo
% Leader
Lmpc = MPCTrajectory(As, Bs, Cs);
Lmpc.setupRobustMPCLayer(N, Qs, Rs, Xs, Us, ΔZs, Ws);
Lmpc.setupStateInputReferenceLayer(autov);
Lmpc.setupTrajectoryLayer(gamma, T, epsilon);
Lmpc.y_setpoint = ybarsetpointLeader;
Lmpc.initialize(Lepuck.position);
% Follower 1
F1mpc = Lmpc.copy();
F1mpc.y_setpoint = ybarsetpointF1;
F1mpc.initialize(F1epuck.position);
% Follower 2
F2mpc = Lmpc.copy();
F2mpc.y_setpoint = ybarsetpointF2;
F2mpc.initialize(F2epuck.position);
```

```
66
67
68  %% Definizione formazione
69  l_12 = 15;
70  psi_12 = deg2rad(135);
71  l_13 = 15;
72  psi_13 = deg2rad(-135);
73
74  %% Ostacoli
75  Obst = [];
76
77
78  %% Iterazioni
79  endTime=10000;
80  t=0;
81  time=0;
82  opt=optimset('Display','off');
83  eps = [];
84  yTilde = []; xTilde = []; xState = [];
85  figure; hold on; grid on;
86  plot(ybarsetpointLeader(1), ybarsetpointLeader(2), 'bp')
87  while (t<endTime)
88
89      %% Calcolo setpoint per follower
90      LeaderRefPos = Lmpc.positionReference;
91      thetaLeader = Lmpc.direction;
92      % F1
93      ybarsetpointF1(1) = LeaderRefPos(1) + l_12 * cos(thetaLeader + psi_12)...
            ;
94      ybarsetpointF1(2) = LeaderRefPos(2) + l_12 * sin(thetaLeader + psi_12)...
            ;
95      F1mpc.y_setpoint = ybarsetpointF1;
96      plot(ybarsetpointF1(1), ybarsetpointF1(2), 'ys')
97      % F2
98      ybarsetpointF2(1) = LeaderRefPos(1) + l_13 * cos(thetaLeader + psi_13)...
            ;
99      ybarsetpointF2(2) = LeaderRefPos(2) + l_13 * sin(thetaLeader + psi_13)...
            ;
100     plot(ybarsetpointF2(1), ybarsetpointF2(2), 'ms')
101     F2mpc.y_setpoint = ybarsetpointF2;
102
103     %% Velocità dell'agente in base alla posizione dei leader
104     epsm = 0.2;
105     epsM = 2.5;
106     dR2 = norm(ybarsetpointF1 - F1mpc.positionReference, 2);
107     dR3 = norm(ybarsetpointF2 - F2mpc.positionReference, 2);
108     epsNew = epsm + 2/pi * (epsM - epsm) * atan(20 / (dR3 + dR2));
109     Lmpc.epsilon = epsNew;
110     eps = [eps, epsNew];
```

```
111
112     %% Vincoli di distanza fra gli agenti
113     % F1 evita L
114     % F2 evita L e F1
115     dMin = 10;
116     % Estraggo le posizioni
117     if isempty(Obst)
118         fconstrF1 = @(x) trajLayerConstraints(x, Lepuck.position, dMin);
119         fconstrF2 = @(x) trajLayerConstraints(x, [Lepuck.position ...
                F1epuck.position], [dMin dMin]);
120     else
121         fconstrF1 = @(x) trajLayerConstraints(x, [Lepuck.position Obst...
                (1:2, :)], [dMin Obst(3, :)]);
122         fconstrF2 = @(x) trajLayerConstraints(x, [Lepuck.position ...
                F1epuck.position Obst(1:2, :)], [dMin dMin Obst(3, :)]);
123     end
124
125     %% Iterazioni degli algoritmi di controllo
126     [u_L, ytilde_L] = Lmpc.step(Lepuck.linState, []);
127     [u_F1, ytilde_F1] = F1mpc.step(F1epuck.linState, fconstrF1);
128     [u_F2, ytilde_F2] = F2mpc.step(F2epuck.linState, fconstrF2);
129
130     %% Anello Feedback Linearization
131     for j = 1:Tfrac
132         Lepuck.step(u_L(1), u_L(2));
133         F1epuck.step(u_F1(1), u_F1(2));
134         F2epuck.step(u_F2(1), u_F2(2));
135     end
136
137     %% Plot posizioni
138     plot(Lepuck);
139     plot(F1epuck);
140     plot(F2epuck);
141     drawnow;
142
143     %% Salvataggio dati
144     yTilde = [yTilde [Lepuck.position; F1epuck.position; F2epuck.position...
            ]];
145     xTilde = [xTilde [Lepuck.linState; F1epuck.linState; F2epuck.linState...
            ]];
146     xState = [xState [Lepuck.state; F1epuck.state; F2epuck.state]];
147
148 end
```

Appendice C

Problema del Contenimento

Si presenta ora il codice per la soluzione del problema del contenimento introdotto nel Capitolo 7.

C.1 Gestione dell'automa a stati finiti: ContainmentStateMachine

La gestione degli automi a stati finiti utilizzati per la soluzione del problema in esame è stata implementata attraverso l'utilizzo di una nuova classe chiamata **ContainmentStateMachine**. Essa si occupa di verificare le condizioni di scatto delle transizioni nel rispetto degli stati attuali. Si noti che i due automi (dei *leader* e dei *follower*) sono sempre sincronizzati e quindi, dal punto di vista implementativo, è sufficiente considerarne uno.

```
1  classdef ContainmentStateMachine < handle
2      %
3  %   -- Classe ContainmentStateMachine --
4  %   Questa classe rappresenta e gestisce gli automi a stati finiti per i
5  %   leader e per i follower nel problema del containment.
6  %
7  %
8  %   # COSTRUTTORE: ContainmentStateMachine(Leader_target, Follower_sensing...
       , Klf)
9  %       * Leader_target: regione inziale (ogni riga è un vertice)
10 %       * Follower_sensing: posizioni di sensing della prima regione (ogni
11 %                           riga è una posizione)
12 %       * Klf: matrice Klf del grafo di connessione
13 %
14 %   # ATTRIBUTI PUBBLICI
```

```
15 %       * epsilon: grandezza che definisce l'epsilon contenimento
16 %       * state: stato corrente dell'automa che assume i seguenti valori:
17 %                   -> ContainmentStates.STOP_CONTAINMENT
18 %                   -> ContainmentStates.GO_FOLLOW
19 %                   -> ContainmentStates.TARGET_SENSING
20 %       * transition: transizione scattata all'ultima iterazione che può
21 %                     assumere i seguenti valori:
22 %                         ->  ContainmentTransitions.NONE
23 %                         ->  ContainmentTransitions.STOP2GO
24 %                         ->  ContainmentTransitions.GO2TARGET
25 %                         ->  ContainmentTransitions.TARGET2GO
26 %                         ->  ContainmentTransitions.GO2STOP
27 %       * Leader_target: vertici regione corrente. Deve essere aggiornata
28 %                     dall'utente allo scatto delle opportune transizioni
29 %       * Follower_sensing: posizioni di sensing della regione corrente.
30 %                     Deve essere aggiornata dall'utente allo scatto
31 %                     delle opportune transizioni.
32 %
33 %
34 %    # METODI PUBBLICI
35 %       * step(machine, LeaderPos, FollowerPos): verifica di tutte le
36 %           transizioni e aggiornamento dello stato in base alle posizioni
37 %           dei leader (LeaderPos) e dei follower (FollowerPos)
38 %
39 %
40
41     properties
42         epsilon = 1;
43         state
44         transition = ContainmentTransitions.NONE;
45         Leader_target
46         Follower_sensing
47         Klf % graph matrix
48     end
49
50     methods
51
52         function machine = ContainmentStateMachine(Leader_target, ...
               Follower_sensing, Klf)
53
54             machine.state = ContainmentStates.STOP_CONTAINMENT;
55             machine.Leader_target = Leader_target;
56             machine.Follower_sensing = Follower_sensing;
57             Klf(Klf ≠ 0) = 1;
58             machine.Klf = Klf;
59
60         end
61
62         function step(machine, LeaderPos, FollowerPos)
```

```
63
64          % La posizione dei leader è una matrice in cui l'i-esima riga ...
                è
65          % la posizione dell'i-esimo leader. Uguale per i follower
66          nLeader = size(LeaderPos, 1);
67          nFollower = size(FollowerPos, 1);
68
69          switch (machine.state)
70              % ...
                    ------------------------------------------------------------...

71              case ContainmentStates.STOP_CONTAINMENT
72                  % Test della transizione STOP2GO: controllo che tutti ...
                        i
73                  % follower siano epsilon-contenuti nella rispettiva
74                  % convex-hull
75                  allFollowersContained = 1;
76                  for i = 1:nFollower
77                      % Computazione del convex-hull per l'i-esimo
78                      % follower a partire dai 'suoi' leader
79
80                      % Estrazione della posizione dei leader nel
81                      % rispetto della matrice Klf
82                      Lp = [];
83                      for j = 1:nLeader
84                          if machine.Klf(i, j) == 1
85                              Lp = [Lp; LeaderPos(j, :)];
86                          end
87                      end
88
89                      % Convex Hull
90                      CH = convhull(Lp(:, 1), Lp(:, 2));
91                      [Ach, bch] = vert2lcon(Lp(CH, :));
92
93                      % Distanza
94                      dist = distanceFromConvexSet(FollowerPos(i, :)', ...
                            Ach, bch);
95                      if dist > machine.epsilon
96                          allFollowersContained = 0;
97                          break;
98                      end
99
100                 end
101
102                 if allFollowersContained == 1
103                     machine.state = ContainmentStates.GO_FOLLOW;
104                     machine.transition = ...
                            ContainmentTransitions.STOP2GO;
105                 else
```

```
106                     machine.transition = ContainmentTransitions.NONE;
107                 end
108
109             % ...
                ------------------------------------------------------------...

110             case ContainmentStates.GO_FOLLOW
111                 % Test per la transizione GO2STOP: esiste almeno un
112                 % follower non epsilon-contenuto?
113                 allFollowersContained = 1;
114                 for i = 1:nFollower
115                     % Calcolo del convex-hull per il follower i in ...
                            base
116                     % ai 'suoi' leader
117                     Lp = [];
118                     for j = 1:nLeader
119                         if machine.Klf(i, j) == 1
120                             Lp = [Lp; LeaderPos(j, :)];
121                         end
122                     end
123                     % Convex Hull
124                     CH = convhull(Lp(:, 1), Lp(:, 2));
125                     [Ach, bch] = vert2lcon(Lp(CH, :));
126                     % Distanza
127                     dist = distanceFromConvexSet(FollowerPos(i, :)', ...
                            Ach, bch);
128                     if dist > machine.epsilon
129                         allFollowersContained = 0;
130                         break;
131                     end
132                 end
133
134                 if allFollowersContained == 0
135                     machine.state = ContainmentStates.STOP_CONTAINMENT...
                            ;
136                     machine.transition = ...
                            ContainmentTransitions.GO2STOP;
137                 else
138                     machine.transition = ContainmentTransitions.NONE;
139                 end
140                 % Test per la transizione GO2TARGET: tutti i leader
141                 % sono nei rispettivi target?
142                 leadersInPosition = 1;
143                 for i = 1:nLeader
144
145                     currentLeaderPosition = LeaderPos(i, :);
146                     currentLeaderTarget = machine.Leader_target(i, :);
147
```

```
148                         if norm(currentLeaderPosition - ...
                                currentLeaderTarget) > machine.epsilon
149                             leadersInPosition = 0;
150                             break;
151                         end
152                     end
153
154                     if leadersInPosition == 1
155                         machine.state = ContainmentStates.TARGET_SENSING;
156                         machine.transition = ...
                                ContainmentTransitions.GO2TARGET;
157                     else
158                         machine.transition = ContainmentTransitions.NONE;
159                     end
160
161                 % ...
                        ------------------------------------------------------------...

162                 case ContainmentStates.TARGET_SENSING
163                     % Test per TARGET2GO: tutti i follower sono nelle
164                     % rispettive posizioni di sensing?
165                     followersInPosition = 1;
166                     for i = 1:nFollower
167
168                         currentFollowerPosition = FollowerPos(i, :);
169                         currentFollowerTarget = machine.Follower_sensing(i...
                                , :);
170
171                         if norm(currentFollowerPosition - ...
                                currentFollowerTarget) > machine.epsilon
172                             followersInPosition = 0;
173                             break;
174                         end
175                     end
176
177                     if followersInPosition == 1
178                         machine.state = ContainmentStates.GO_FOLLOW;
179                         machine.transition = ...
                                ContainmentTransitions.TARGET2GO;
180                     else
181                         machine.transition = ContainmentTransitions.NONE;
182                     end
183             end
184         end
185     end
186 end
```

C.2 Codice per il controllo degli agenti

A partire dalla classe **ContainmentStateMachine**, unita a quelle precedentemente introdotte, si presenta ora il codice completo per la soluzione del problema in esame in cui si sono utilizzati gli *epuck* reali. In questo modo il lettore può verificare anche l'utilizzo delle classi **EpuckReal** e **CameraPosition**.

```
clear all;close all;clc; warning off all;


%% Parametri tempo e iterazione
Δt = 3;   % Tempo totale loop feedbacklin (ogni loop 0.3s)
Tfrac = 10;     % Iterazioni feedback lin
tau = Δt + 0.7;      % Tempo loop MPC
N = 3;

%% Camera
camera = CameraPosition(0, 1, 1);
% Acquisizione di qualche frame per stabilizzare l'immagine
for i = 1:5
    [xF, yF, thetaF] = camera.getPositions(2);
end

%% Stato iniziale dei robot e setpoint.
% L1
L1epuck = Epuck(20, 53, 0, 0.02, Δt / Tfrac);
L1epuck.color = 'g.';
% L2
L2epuck = Epuck(50, 47, 0, 0.02, Δt / Tfrac);
L2epuck.color = 'b.';
% L3
L3epuck = Epuck(45, 22, 0, 0.02, Δt / Tfrac);
L3epuck.color = 'k.';
% L4
L4epuck = Epuck(25, 22, 0, 0.02, Δt / Tfrac);
L4epuck.color = 'r.';
% F1
F1epuck = EpuckReal(xF(1), yF(1), thetaF(1), 0.02, Δt / Tfrac);
F1epuck.color = 'c.';
% F2
F2epuck = EpuckReal(xF(2), yF(2), thetaF(2), 0.02, Δt / Tfrac);
F2epuck.color = 'y.';

%% Connessione Follower
F1epuck.connect('/dev/tty.e-puck_2686-COM1');
F2epuck.connect('/dev/tty.e-puck_2540-COM1');
```

```
40
41
42 %% Definizione Vertici Regioni (ogni riga è un vertice)
43 D1 = [60 50; 100 50; 100 25; 60 25];
44 D2 = [30 20; 100 20;  90 10; 20 10];
45 regions = {D1, D2};
46
47 %% Definizioni Posizioni di sensing per i follower
48 S1 = [70 45; 90 37.5];
49 S2 = [40 15;  70 15];
50 sensings = {S1, S2};
51
52 %% Connectivity Matrix
53 Klf = 1/3 * [1 0 1 1; 1 1 1 0];
54
55 %% Plot posizioni
56 close all
57 figure(1)
58 hold on
59 grid on
60 for i = 1:length(regions)
61     % Follower
62     sens = cell2mat(sensings(i));
63     plot(sens(1, 1), sens(1, 2), 'cd', 'MarkerFaceColor', 'k');
64     plot(sens(2, 1), sens(2, 2), 'yd', 'MarkerFaceColor', 'k');
65     % Leader
66     reg = cell2mat(regions(i));
67     plot([reg(:, 1); reg(1, 1)], [reg(:, 2); reg(1, 2)], 'LineWidth', 2);
68     plot([reg(:, 1); reg(1, 1)], [reg(:, 2); reg(1, 2)], 'bs');
69 end
70 disp('Regions defined...')
71
72
73 %% Sistema linearizzato dell'agente (in una direzione)
74 As=[1 tau ; 0 1];
75 Bs=[0.5*tau^2 ; tau];
76 Cs=[1 0];
77
78 %% Definizione dei Set per TubeBased MPC
79 % Modello Agente parziale
80 As=[1 tau ; 0 1];
81 Bs=[0.5*tau^2 ; tau];
82 Cs=[1 0];
83 % Peso stato/ingresso
84 Qs = 5*eye(2);
85 Rs = 100*eye(1);
86 % Disturbo
87 w_arb = 1;
88 Ws = [1 1; 1 -1; -1,1; -1 -1] * w_arb;
```

```
% Vincoli stato
xbar = 200;
ybar = 200;
vbar = 10;
Xs = [xbar, vbar; -xbar, vbar; xbar, -vbar; -xbar, -vbar];
% Vincoli Ingresso
Us = [];
% Vincoli uscita
ΔZs = [-1; 1];

%% Definizione autovalori livello intermedio
autov = [0.3 0.31 0.32 0.33 0.34 0.35];

%% Definizione parametri livello di gen. della traiettoria
gamma = 1;
T = 10 * eye(2);
epsilon = 1.5;
epsilonLeader = 2;

%% Inizializzo oggetti per il controllo MPC
% L1
L1mpc = MPCTrajectory(As, Bs, Cs);
L1mpc.setupRobustMPCLayer(N, Qs, Rs, Xs, Us, ΔZs, Ws);
L1mpc.setupStateInputReferenceLayer(autov);
L1mpc.setupTrajectoryLayer(gamma, T, epsilon);
% L2
L2mpc = L1mpc.copy();
% L3
L3mpc = L1mpc.copy();
% L4
L4mpc = L1mpc.copy();
% F1
F1mpc = L1mpc.copy();
% F2
F2mpc = L1mpc.copy();
% Imposto la diversa velocità dei leader
L1mpc.epsilon = epsilonLeader;
L2mpc.epsilon = epsilonLeader;
L3mpc.epsilon = epsilonLeader;
L4mpc.epsilon = epsilonLeader;
% Inizializzazione
L1mpc.initialize(L1epuck.position);
L2mpc.initialize(L2epuck.position);
L3mpc.initialize(L3epuck.position);
L4mpc.initialize(L4epuck.position);
F1mpc.initialize(F1epuck.position);
F2mpc.initialize(F2epuck.position);


```

```
138 %% State Machine
139 stateMachine = ContainmentStateMachine(D1, S1, Klf);
140 LeadersPos = [L1epuck.position'; L2epuck.position'; L3epuck.position'; ...
        L4epuck.position'];
141 FollowersPos = [F1epuck.position'; F2epuck.position'];
142 stateMachine.epsilon = 1.5;
143 stateMachine.step(LeadersPos, FollowersPos);
144
145 %% Loop
146 % Indice regione corrente
147 currentRegionIndex = 1;
148 % Set Points
149 % I leader vanno verso la prima regione
150 region = cell2mat(regions(currentRegionIndex));
151 L1mpc.y_setpoint = region(1, :)';
152 L2mpc.y_setpoint = region(2, :)';
153 L3mpc.y_setpoint = region(3, :)';
154 L4mpc.y_setpoint = region(4, :)';
155 % Follower nella convex hull
156 F1mpc.y_setpoint = region' * Klf(1, :)'; % Klf * (L1, L2 ..., Ln)'
157 F2mpc.y_setpoint = region' * Klf(2, :)';
158 % Salvataggio lo stato dell'automa
159 stateMachineStateMemory = [];
160 % Contatore iterazioni
161 iterationCount = 0;
162 % Vettori tempo, posizione e riferimento
163 time = 0;
164 yTilde = [L1epuck.position; L2epuck.position; L3epuck.position; ...
        L4epuck.position; F1epuck.position; F2epuck.position];
165 xTilde = [L1epuck.linState; L2epuck.linState; L3epuck.linState; ...
        L4epuck.linState; F1epuck.linState; F2epuck.linState];
166 xState = [L1epuck.state; L2epuck.state; L3epuck.state; L4epuck.state; ...
        F1epuck.state; F2epuck.state];
167 while 1
168
169     % tic utilizzato per temporizzare l'anello esterno (MPC)
170     ticDPCLoop = tic;
171
172     %% Acquisizione della posizione degli agenti
173     [xF, yF, thetaF] = camera.getPositions(2);
174     F1epuck.state = [xF(1); yF(1); thetaF(1)];
175     F2epuck.state = [xF(2); yF(2); thetaF(2)];
176
177     %% Aggionramento della macchina a stati
178     LeadersPos = [L1epuck.position'; L2epuck.position'; L3epuck.position';...
            L4epuck.position'];
179     LeadersPosRef = [L1mpc.positionReference'; L2mpc.positionReference'; ...
            L3mpc.positionReference'; L4mpc.positionReference'];
180     FollowersPos = [F1epuck.position'; F2epuck.position'];
```

```
181     stateMachine.step(LeadersPos, FollowersPos);
182     stateMachineStateMemory = [stateMachineStateMemory; stateMachine.state...
            ];
183
184     %% Aggiornamento della regione corrente (se necessario)
185     if stateMachine.transition == ContainmentTransitions.TARGET2GO
186         fprintf('Regione %d completata...', currentRegionIndex)
187         if currentRegionIndex < length(regions)
188             % Aggiornamento dell'indice della regione
189             currentRegionIndex = currentRegionIndex + 1;
190             % Aggiornamento della regione corrente nell'automa
191             stateMachine.Leader_target = cell2mat(regions(...
                    currentRegionIndex));
192             stateMachine.Follower_sensing = cell2mat(sensings(...
                    currentRegionIndex));
193             fprintf('nuova regione: %d\n', currentRegionIndex)
194         else
195             % Tutte le regione sono state completate
196             fprintf('Tutte le regione completate!\n')
197             break;
198         end
199         % Reset contatore iterazioni
200         iterationCount = 0;
201     %% Mappatura posizioni di sensing in base alla posizione dei follower
202     elseif stateMachine.transition == ContainmentTransitions.GO2TARGET
203         % Assegna la posizone di sensing i al follower più vicino
204         sensingOriginal = cell2mat(sensings(currentRegionIndex));
205         sensing = zeros(size(sensingOriginal));
206         % Trova tutte le distanze fra le posizioni di sensing e i follower
207         distanceMatrix = inf*ones(size(sensingOriginal, 1));
208         % l'elemento i, j è la distanza dal sensing i al follower j
209         for j = 1:size(sensingOriginal, 1)
210             for i = 1:size(sensingOriginal, 1)
211                 distanceMatrix(i, j) = norm(FollowersPos(j, :) - ...
                        sensingOriginal(i, :));
212             end
213         end
214         % Computazione di tutte le distanze
215         permsFollowers = perms(1:size(sensingOriginal, 1));
216         distances = inf * ones(size(sensingOriginal, 1), 1);
217         for p = 1:size(permsFollowers, 1) % for ogni permutazione
218             distances(p) = distanceMatrix(1, permsFollowers(p, 1)) + ...
                    distanceMatrix(2, permsFollowers(p, 2));
219         end
220         [¬, minIdx] = min(distances);
221         bestPerm = permsFollowers(minIdx, :);
222         % Assegnamento della posizione
223         for j = 1:size(sensingOriginal, 1)
224             sensing(bestPerm(j), :) = sensingOriginal(j, :);
```

```
225         end
226
227         % Aggiornamento del manager dell'automa con le nuove posizioni di
228         % sensing appena assegnate
229         sensings(currentRegionIndex) = {sensing};
230         stateMachine.Follower_sensing = sensing;
231
232         % Reset contatore iterazioni
233         iterationCount = 0;
234     end
235
236
237     %% Assegnamento dei setpoint ai vari agenti a seconda dello stato
238     switch (stateMachine.state)
239
240         case ContainmentStates.GO_FOLLOW
241             % I leader vanno verso la regione corrente
242             region = cell2mat(regions(currentRegionIndex));
243             L1mpc.y_setpoint = region(1, :)';
244             L2mpc.y_setpoint = region(2, :)';
245             L3mpc.y_setpoint = region(3, :)';
246             L4mpc.y_setpoint = region(4, :)';
247             % I Follower stanno nella convex hull
248             F1mpc.y_setpoint = LeadersPosRef' * Klf(1, :)'; % Klf * (L1, ...
                    L2 ..., Ln)'
249             F2mpc.y_setpoint = LeadersPosRef' * Klf(2, :)';
250
251         case ContainmentStates.TARGET_SENSING
252             % I Leader stanno fermi dove sono
253             % I Follower si muovo nella rispettiva posizione di sensing
254             F1mpc.y_setpoint = sensing(1, :)';
255             F2mpc.y_setpoint = sensing(2, :)';
256
257
258         case ContainmentStates.STOP_CONTAINMENT
259             % I Leader stanno fermi dove sono
260             L1mpc.y_setpoint = L1mpc.positionReference;
261             L2mpc.y_setpoint = L2mpc.positionReference;
262             L3mpc.y_setpoint = L3mpc.positionReference;
263             L4mpc.y_setpoint = L4mpc.positionReference;
264             % I follower vanno nel convex hull
265             F1mpc.y_setpoint = LeadersPosRef' * Klf(1, :)'; % Klf * (L1, ...
                    L2 ..., Ln)'
266             F2mpc.y_setpoint = LeadersPosRef' * Klf(2, :)';
267
268     end
269
270
271     %% Collision Avoidance fra i follower
```

```
272     if exist('ytilde_L1', 'var')
273         OldRef = [ytilde_L1, ytilde_L2, ytilde_L3, ytilde_L4, ytilde_F1, ...
                ytilde_F2];
274         L1_collAvoid = []; %@(y) L1CollisionAvoidance(y, OldRef(:, 2:7), ...
                5*ones(6, 1));
275         L2_collAvoid = []; %@(y) L1CollisionAvoidance(y, OldRef(:, [1, ...
                3:7]), 5.1*ones(6, 1));
276         L3_collAvoid = []; %@(y) L1CollisionAvoidance(y, OldRef(:, [1:2, ...
                4:7]), 5.2*ones(6, 1));
277         L4_collAvoid = []; %@(y) L1CollisionAvoidance(y, OldRef(:, [1:3, ...
                5:7]), 5.3*ones(6, 1));
278         F1_collAvoid = @(y) L1CollisionAvoidance(y, OldRef(:, 6), 12);
279         F2_collAvoid = @(y) L1CollisionAvoidance(y, OldRef(:, 5), 12);
280     else
281         L1_collAvoid = [];
282         L2_collAvoid = [];
283         L3_collAvoid = [];
284         L4_collAvoid = [];
285         F1_collAvoid = [];
286         F2_collAvoid = [];
287     end
288
289     %% Iterazione algoritmo di controllo
290     [u_L1, ytilde_L1] = L1mpc.step(L1epuck.linState, L1_collAvoid);
291     [u_L2, ytilde_L2] = L2mpc.step(L2epuck.linState, L2_collAvoid);
292     [u_L3, ytilde_L3] = L3mpc.step(L3epuck.linState, L3_collAvoid);
293     [u_L4, ytilde_L4] = L4mpc.step(L4epuck.linState, L4_collAvoid);
294     [u_F1, ytilde_F1] = F1mpc.step(F1epuck.linState, F1_collAvoid);
295     [u_F2, ytilde_F2] = F2mpc.step(F2epuck.linState, F2_collAvoid);
296
297     %% Feedback linearization
298     for j = 1:Tfrac
299
300         % tic per temporizzare l'anello interno
301         ticFeedbackLin = tic;
302
303         % Acquisizione posizione agenti
304         [xF, yF, thetaF] = camera.getPositions(2);
305         F1epuck.state = [xF(1); yF(1); thetaF(1)];
306         F2epuck.state = [xF(2); yF(2); thetaF(2)];
307
308         % Applicazione legge di controllo
309         if iterationCount > N
310
311             L1epuck.step(u_L1(1), u_L1(2));
312             L2epuck.step(u_L2(1), u_L2(2));
313             L3epuck.step(u_L3(1), u_L3(2));
314             L4epuck.step(u_L4(1), u_L4(2));
315             if norm(F1epuck.position - ytilde_F1) > 1.2
```

```
316                 F1epuck.step(u_F1(1), u_F1(2));
317             else
318                 F1epuck.fakeStep(u_F1(1), u_F1(2));
319             end
320             if norm(F2epuck.position - ytilde_F2) > 1.2
321                 F2epuck.step(u_F2(1), u_F2(2));
322             else
323                 F2epuck.fakeStep(u_F2(1), u_F2(2));
324             end
325         end
326
327         % Temporizzazione
328         while(toc(ticFeedbackLin) < Δt/Tfrac)
329         end
330
331     end
332
333
334     %% Plot
335     plot(L1epuck);
336     plot(L2epuck);
337     plot(L3epuck);
338     plot(L4epuck);
339     plot(F1epuck);
340     plot(F2epuck);
341     plot(ytilde_F1(1), ytilde_F1(2), 'co');
342     plot(ytilde_F2(1), ytilde_F2(2), 'yo');
343
344     drawnow;
345
346     %% Temporizzazione
347     while(toc(ticDPCLoop) < Δt)
348     end
349
350     %% Aggiornamento del tempo e del contatore iterazioni
351     iterationCount = iterationCount + 1;
352     newTime = time(end) + Δt;
353     time = [time; newTime];
354
355     %% Aggiornamento vettori da salvare
356     yTilde = [yTilde [L1epuck.position; L2epuck.position; L3epuck.position...
            ; L4epuck.position; F1epuck.position; F2epuck.position]];
357     xTilde = [xTilde [L1epuck.linState; L2epuck.linState; L3epuck.linState...
            ; L4epuck.linState; F1epuck.linState; F2epuck.linState]];
358     xState = [xState [L1epuck.state; L2epuck.state; L3epuck.state; ...
            L4epuck.state; F1epuck.state; F2epuck.state]];
359
360
361 end
```

```
362
363 %% Savataggio risultati
364 save Epuck2_containment.mat yTilde xTilde xState time regions sensings Klf...
        stateMachineStateMemory
365 % I dati di ogni robot sono posti per riga in yTilde xTilde xState.
366
367 %% Chiusura connessioni
368 camera.close();
369 F1epuck.close();
370 F2epuck.close();
371
372 %% Plot stati dell'automa
373 figure
374 plotStates(stateMachineStateMemory);
```

Appendice D

Problema della Navigazione in Ambiente Sconosciuto

Si presenta il codice utilizzato per la soluzione del problema del contenimento introdotto nel Capitolo 8.

D.1 Gestione dell'automa a stati finiti: AutonomousNavigation

La gestione dell'automa a stati finiti introdotto nel Capitolo 8 è stata implementata in una classe chiamata **AutonomousNavigation** che analizza tutte le condizioni di scatto di tutte le transizioni. Inoltre essa fornisce anche un metodo, chiamato `getVisiblePoints`, che elabora i punti visibili dall'agente a partire dalla sua posizione e orientamento. A differenza di quando svolto per il problema del contenimento, questa classe gestisce anche la generazione della traiettoria, semplificandone ulteriormente l'utilizzo. Infatti ora il metodo `step` ritorna direttamente l'accelerazione da imporre all'agente.

```
1 classdef AutonomousNavigation < handle
2 %
3 %   -- Classe AutonomousNavigation --
4 %   Questa classe gestisce l'automa per la navigazione in ambiente
5 %   sconosciuto, oltre alla computazione dei punti visibili e la ...
      visibilità
6 %   del goal
7 %
```

```
8  %   # COSTRUTTORE: AutonomousNavigation(mpcCustom, Obstacles, ...
       angoloVisuale, distanzaVisuale, distanzaOstacolo)
9  %       * mpcCustom: istanza della classe MPCCustomTrajectory
10 %       * Obstacles: lista (matlab cell) con gli ostacoli
11 %       * angoloVisuale: angolo \alpha_{vis} dell'agente
12 %       * distanzaVisuale: distanza d_{vis} dell'agente
13 %       * distanzaOstacolo: distanza d_{pat} da mantenere dall'ostacolo
14 %
15 %   # ATTRIBUTI PUBBLICI
16 %       * onlyPatrollin: boolean. Se 1 allora viene risolto il solo
17 %                        problema del patrolling.
18 %       * goal: obiettivo da raggiungere
19 %       * Obstacles: vedi costruttore
20 %
21 %
22 %   # METODI PUBBLICI
23 %       * VisiblePts = getVisiblePoints(Pa, theta): ritorna una lista
24 %                      (matlab cell) dei punti visibili data la posizione
25 %                      Pa e orientamento theta dell'agente.
26 %
27 %       * [u, y_next, reset] = step(linState, theta): verifica tutte le
28 %                              condizioni di scatto delle transizioni ed
29 %                              elabora l'azione di controllo per l'agente...
       ,
30 %                              restituendo l'accelerazione u, il nuovo
31 %                              punto della traiettoria di riferimento e ...
       un
32 %                              flag 'reset' che vale 1 se la traiettoria ...
       è
33 %                              stata resettata.
34 %
35 %
36     properties (SetAccess = protected)
37
38         % Punti Visibili e precedenti
39         VisiblePoints = {};
40         OldVisiblePoints = {};
41
42         % Distanza minima dell'ostacolo più vicino
43         minimumDistanceToClosestObstacle = inf;
44         minimunDistanceToObstacles = {};
45
46         % Instanza della classe MPCCustomTrajectory
47         mpc = [];
48
49         % Parametri campo visivo agente
50         alphaVis = 0;
51         dMaxVis = 0;
52
```

```
53          % Distanza minima da ostacolo
54          dObstMin = 0;
55
56          % Stato automa
57          state = [];
58
59          % Valore di regime di epsilon nel livello di traiettoria
60          epsilonTrajectorySS = 0;
61
62          % Strettoia
63          bottleneckSetPoint = [];
64          bottleneckTargetObstacle = [];
65
66          % Punto cieco
67          blindDirection = 0;
68          gamma_blind = 0;
69          blindHandlePlot = [];
70
71      end
72
73      properties (SetAccess = public)
74          % Se vera, si risolve solo il problema del patrolling
75          onlyPatrolling = true;
76
77          % Goal (utilizzato se onlyPatrolling == false)
78          goal = [];
79
80          % Osctacoli (celle)
81          Obstacles = {};
82      end
83
84      methods
85
86          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
87          %   COSTRUTTORE
88          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
89          function nav = AutonomousNavigation(mpcCustom, Obstacles, ...
                angoloVisuale, distanzaVisuale, distanzaOstacolo)
90
91              nav.mpc = mpcCustom;
92              nav.Obstacles = Obstacles;
93              nav.alphaVis = angoloVisuale;
94              nav.dMaxVis = distanzaVisuale;
95              nav.dObstMin = distanzaOstacolo;
96
97              % Stato inziale in goal
98              nav.state = AutonomousNavigationState.GOAL;
99
100             % Valore epsilon di regime uguale a quello fornito
```

```
101             nav.epsilonTrajectorySS = mpcCustom.epsilon;
102         end
103 
104         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
105         %   SETTER
106         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
107         function set.goal(nav, goal)
108 
109             nav.goal = goal;
110             nav.mpc.y_setpoint = goal;
111 
112         end
113 
114         function set.onlyPatrolling(nav, pat)
115 
116             nav.onlyPatrolling = pat;
117             if pat
118                 nav.state = AutonomousNavigationState.PATROLLING;
119             else
120                 nav.state = AutonomousNavigationState.GOAL;
121             end
122 
123         end
124 
125 
126         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
127         %   PUNTI VISIBILI
128         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
129         function VisiblePts = getVisiblePoints(nav, Pa, theta)
130 
131             % Inizializzo le variabili
132             nav.VisiblePoints = {};
133             nav.minimumDistanceToClosestObstacle = inf;
134             nav.minimunDistanceToObstacles = {};
135             %% Trovo i punti visibili in ogni ostacolo
136             for h = 1:length(nav.Obstacles)
137                 y_O = [];
138                 % Ostacolo
139                 y_OBSTACLE = cell2mat(nav.Obstacles(h));
140 
141                 % Trovo i Punti visibili
142                 [indicePuntiVisibili, distMinObstacle] = visiblePoints(Pa,...
                        theta, nav.alphaVis, nav.dMaxVis, y_OBSTACLE, ...
                        nav.blindDirection);
143 
144                 % Estraggo i punti trovati, se almeno 3
145                 if length(indicePuntiVisibili) > 2
146                     y_O = y_OBSTACLE(:, indicePuntiVisibili);
```

```
147                     nav.minimunDistanceToObstacles(h) = num2cell(...
                            distMinObstacle);
148                 else
149                     nav.minimunDistanceToObstacles(h) = num2cell(inf);
150                 end
151
152                 % Salva i punti
153                 newVisiblePoints(h) = mat2cell(y_O);
154
155                 % Aggiorna la distanza minima
156                 if distMinObstacle < nav.minimumDistanceToClosestObstacle
157                     nav.minimumDistanceToClosestObstacle = distMinObstacle...
                            ;
158                 end
159             end
160             nav.VisiblePoints = newVisiblePoints;
161             VisiblePts = nav.VisiblePoints;
162         end
163
164         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
165         %   STEP
166         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
167         function [u, y_next, reset] = step(nav, linState, theta)
168
169             reset = 0;
170
171             if nav.onlyPatrolling
172
173                 % Estraggo posizione dallo stato
174                 Pa = linState([1 3]);
175
176                 if nav.state == AutonomousNavigationState.PATROLLING
177                     blindFlag = nav.checkBlindSpot();
178                     if blindFlag
179                         nav.state = AutonomousNavigationState.BLIND;
180                         disp('PATROLLING -> BLIND');
181                         % L'agente è in un punto cieco. Genera il cerchio
182                         % attorno all'ultimo punto
183                         nav.blindDirection = 0;
184                         reset = 1;
185
186                     %    nav.mpc.resetTrajectory(Pa);
187                     end
188                  elseif nav.state == AutonomousNavigationState.BLIND
189
190                     blindFlag = nav.checkBlindSpot();
191                     nav.blindVisiblePoints(Pa, theta);
192
193                     if ¬blindFlag
```

```
194                         nav.state = AutonomousNavigationState.PATROLLING;
195                         disp('BLIND -> PATROLLING');
196 
197                       %  nav.mpc.resetTrajectory(Pa);
198                     end
199 
200                 end
201 
202                 [u, y_next] = nav.stepPatrolling(linState);
203 
204             else
205 
206                 % Estraggo posizione dallo stato
207                 Pa = linState([1 3]);
208 
209                 %% Transizioni di stato
210                 % Valuto se il goal è visibile
211                 goalVisible = nav.visibleGoal(Pa);
212                 %fprintf('goalVisible: %d\n', goalVisible);
213 
214                 % Valuto se l'agente vede una strettoia
215                 [bottleneckFlag, bottleneckPoint] = nav.bottleneck();
216 
217                 % Se i due osctacoli non sono troppo vicini, i.e. l'agente
218                 % è bloccato, considero solo quello più vicino ad esso
219                 if length(nav.VisiblePoints) > 1 && ¬bottleneckFlag
220 %                     % Considero solo l'ostacolo più vicino
221 %                     Y = cell2mat(nav.minimunDistanceToObstacles);
222 %                     [¬, closestObstacleIndex] = min(Y);
223 %                     nav.VisiblePoints = nav.VisiblePoints(...
    closestObstacleIndex);
224 %                     fprintf('Solo ostacolo %d\n', closestObstacleIndex);
225                     % Considero l'ostacolo più vicino al goal
226                     % Per ogni ostacolo visibile calcolo la distanza minima
227                     % dal goal
228                     minDistances = inf*ones(1, length(nav.VisiblePoints));
229                     for h = 1:length(nav.VisiblePoints)
230                         % Estraggo i punti visibili
231                         yO = cell2mat(nav.VisiblePoints(h));
232                         for k = 1:size(yO, 2)
233                             % Calcolo la distanza
234                             d = norm(yO(:, k) - nav.goal);
235                             if d < minDistances(h)
236                                 minDistances(h) = d;
237                             end
238                         end
239                     end
240                     [¬, closestObstacleIndex] = min(minDistances);
```

```
241                     nav.VisiblePoints = nav.VisiblePoints(...
                            closestObstacleIndex);
242                     fprintf('Solo ostacolo %d\n', closestObstacleIndex);
243                 end
244
245                 % Se sono nello stato di PATROLLING ed il goal è visible,
246                 % allora vado nello stato di GOAL
247                 if nav.state == AutonomousNavigationState.PATROLLING
248
249                     % Valuto se non ci sono punti visibili
250                     blindFlag = nav.checkBlindSpot();
251                     %fprintf('blindFlag: %d\n', blindFlag);
252
253                     % Transizione nello stato BOTTLENECK
254                     if bottleneckFlag
255                         nav.state = AutonomousNavigationState.BOTTLENECK;
256                         disp('PATROLLING -> BOTTLENECK');
257
258
259                         % Trovo il punto più distante e dirigo l'agente
260                         % verso tale direzione
261                         plot(bottleneckPoint(1), bottleneckPoint(2), 'kp')...
                                ;
262                         [fp, obstacleIndex] = nav.farthestVisiblePoint(Pa,...
                                bottleneckPoint);
263                         plot(fp(1), fp(2), 'mp', 'MarkerSize', 15);
264                         nav.bottleneckSetPoint = fp;
265                         nav.bottleneckTargetObstacle = obstacleIndex;
266
267                         % Resetto la generazione della traiettoria
268                         nav.mpc.resetTrajectory(Pa);
269                         nav.mpc.y_setpoint = nav.bottleneckSetPoint;
270
271                         for kk = 1:5
272                             nav.mpc.step(linState, []);
273                         end
274
275                     elseif blindFlag
276                         nav.state = AutonomousNavigationState.BLIND;
277                         disp('PATROLLING -> BLIND');
278
279                         % L'agente è in un punto cieco. Genera il cerchio
280                         % attorno all'ultimo punto
281                         nav.blindDirection = 0;
282                     %     nav.mpc.resetTrajectory(Pa);
283
284                     % Transizione nello stato GOAL
285                     elseif goalVisible && ...
                            nav.minimumDistanceToClosestObstacle > (0.8*...
```

```
                    nav.dObstMin)
286                     nav.state = AutonomousNavigationState.GOAL;
287                     disp('PATROLLING -> GOAL');
288                     %    nav.mpc.resetTrajectory(Pa);
289                     nav.mpc.epsilon = nav.epsilonTrajectorySS;
290                 end
291
292
293
294                 % Se invece sono nello stato GOAL e vedo dei punti più
295                 % vicini di dObstMin, passo nello stato di PATROLLING
296             elseif nav.state == AutonomousNavigationState.GOAL
297                 if (¬goalVisible && ...
                        nav.minimumDistanceToClosestObstacle < (2*...
                        nav.dObstMin))
298                     nav.state = AutonomousNavigationState.PATROLLING;
299                     disp('GOAL -> PATROLLING');
300
301                     % Quando avviene questa transizione è necessario
302                     % bloccare l'agente nella posizione corrente ...
                            imponendo
303                     % la traiettoria futura pari alla posizione ...
                            attuale
304                     nav.mpc.epsilon = nav.epsilonTrajectorySS;
305                     nav.mpc.resetTrajectory(Pa);
306
307                     for kk = 1:5
308                         nav.stepPatrolling(linState);
309                     end
310
311                 end
312
313             elseif nav.state == AutonomousNavigationState.BOTTLENECK
314
315                 % bottleneckTargetObstacle
316                 % minimunDistanceToObstacles = {};
317
318                 transition = 0;
319                 for h = 1:length(nav.minimunDistanceToObstacles)
320
321                     dist = cell2mat(nav.minimunDistanceToObstacles(h))...
                            ;
322                     % La distanza dall'ostacolo verso il quale l'...
                            agente
323                     % si sta dirigendo è maggiore degli altri.
324                     if h == nav.bottleneckTargetObstacle
325                         if dist < 2*nav.dObstMin
326                             transition = 1;
327                             break;
```

```
328                         end
329                     elseif dist < 0.8*nav.dObstMin
330                             transition = 1;
331                             break;
332                     end
333
334                 end
335
336                 if transition
337                     nav.state = AutonomousNavigationState.PATROLLING;
338                     disp('BOTTLENECK -> PATROLLING');
339                     nav.mpc.resetTrajectory(Pa);
340
341
342                     for kk = 1:5
343                         nav.stepPatrolling(linState);
344                     end
345                 end
346
347             elseif nav.state == AutonomousNavigationState.BLIND
348
349                 blindFlag = nav.checkBlindSpot();
350                 nav.blindVisiblePoints(Pa, theta);
351
352                 if ¬blindFlag
353                     nav.state = AutonomousNavigationState.PATROLLING;
354                     disp('BLIND -> PATROLLING');
355                 end
356
357             end
358
359             %% Step
360             % GOAL
361             if nav.state == AutonomousNavigationState.GOAL
362                 % Aggiorno il valore di epsilon
363                 if nav.mpc.epsilon < nav.epsilonTrajectorySS
364                     nav.mpc.epsilon = nav.mpc.epsilon + ...
                            nav.epsilonTrajectorySS/10;
365                 else
366                     nav.mpc.epsilon = nav.epsilonTrajectorySS;
367                 end
368
369                 % Utilizzo il layer traiettoria classico per il ...
                        raggiungimento di
370                 % un obiettivo implementato nella classe MPCTrajectory...
                         (metodo step)
371                 nav.mpc.y_setpoint = nav.goal;
372                 [u, y_next] = nav.mpc.step(linState, []);
373
```

```
374                 % PATROLLING
375             elseif nav.state == AutonomousNavigationState.PATROLLING
376 
377                 % Risolvo il problema del patrolling
378                 [u, y_next] = nav.stepPatrolling(linState);
379 
380                 % BOTTLENECK
381             elseif nav.state == AutonomousNavigationState.BOTTLENECK
382 
383                 nav.mpc.y_setpoint = nav.bottleneckSetPoint;
384                 [u, y_next] = nav.mpc.step(linState, []);
385 
386 
387             elseif nav.state == AutonomousNavigationState.BLIND
388 
389                 % Risolvo il problema del patrolling
390                 [u, y_next] = nav.stepPatrolling(linState);
391 
392 
393             end
394 
395         end
396 
397     end
398 
399 end
400 
401 methods (Access = private)
402 
403     function [u, y_next] = stepPatrolling(nav, linState)
404 
405         %% Funzione di costo
406         % Determino il numero totale di punti visibili
407         N_traj = 0;
408         for h = 1:length(nav.VisiblePoints)
409             % Estraggo i punti visibili
410             y_O = cell2mat(nav.VisiblePoints(h));
411             % Sommo il numero al totale
412             N_traj = N_traj + size(y_O, 2);
413         end
414         % Impongo una funzione di costo per ogni ostacolo
415         constrFunct = @(x) patrollingConstrFunctionMultiObstacle(x, ...
                nav.dObstMin, nav.VisiblePoints);
416 
417         % vincoli lineari:
418         %       y_{k+i} - y_{k+i-1} \in \beta_{\epsilon}(0), i = 1, ...
                ..., N
419         %       Δ_i ≥ 0
420         %
```

```
421             [H_eps, L_eps] = nav.mpc.trajDistanceLinConstraint();
422             A = blkdiag(H_eps, -1*eye(N_traj));
423             b = [L_eps; zeros(N_traj, 1)];
424 
425             % Funzione di costo
426             costFunct = @(x) patrollingCostFunctionMultiObstacle(x, ...
                    nav.VisiblePoints);
427 
428             % Punto iniziale (posizione corrente e tutti i Δ a 0
429             startPoint = [nav.mpc.positionReference(); zeros(N_traj, 1)];
430 
431             % Risolvi problema di ottimizzazione
432             optVector = fmincon(costFunct, startPoint, A , b, [], [], [], ...
                    [], constrFunct, optimset('Display', 'off'));
433             y_next = optVector(1:2, 1);
434 
435             % Step
436             u = nav.mpc.stepCustom(linState, y_next);
437 
438         end
439 
440 
441         function visible = visibleGoal(nav, Pa)
442 
443             visible = 1;
444             for h = 1:length(nav.VisiblePoints)
445 
446                 % Estraggo il primo e l'ultimo punto visibili. L'idea è ...
                        che
447                 % l'ostacolo è visibile se è al di fuori del cono con
448                 % centro nella posizione dell'agente, definito da questi
449                 % due punti estremi.
450                 y_O = cell2mat(nav.VisiblePoints(h));
451                 if size(y_O, 2) == 0 continue; end
452                 qf = y_O(:, end);
453                 qi = y_O(:, 1);
454 
455                 % Trovo l'angolo che i punti visibili estremi formano con
456                 % la posizione dell'agente
457                 gamma_qf = atan2(Pa(2) - qf(2), Pa(1) - qf(1));
458                 gamma_qi = atan2(Pa(2) - qi(2), Pa(1) - qi(1));
459 
460                 % Trovo l'angolo che il goal forma con l'agente
461                 gamma_g = atan2(nav.goal(2) - Pa(2), nav.goal(1) - Pa(1));
462 
463                 % Valuto se il goal è contenuto nel cono o meno
464                 gamma_diff_i = wrapToPi(gamma_g - gamma_qi);
465                 gamma_diff_f = wrapToPi(gamma_g - gamma_qf);
466                 gamma_diff_fi = wrapToPi(gamma_qf - gamma_qi);
```

```
467
468             %  plot([Pa(1) qf(1)], [Pa(2) qf(2)], 'b');
469             %  plot([Pa(1) nav.goal(1)], [Pa(2) nav.goal(2)], 'g');
470
471             %  fprintf('(%d) - i: %f, f: %f, if: %f ', h, gamma_diff_i...
                    , gamma_diff_f, gamma_diff_fi);
472
473             if gamma_diff_fi ≥ 0
474                 if ¬(gamma_diff_f ≤ 0)
475                     visible = 0;
476                     %         fprintf(' ->  non visible\n');
477                 else
478                     %         fprintf(' -> visible\n');
479                 end
480             elseif gamma_diff_fi ≤ 0
481                 if ¬(gamma_diff_f ≥ 0)
482                     visible = 0;
483                     %    fprintf(' ->  non visible\n');
484                 else
485                     %    fprintf(' -> visible\n');
486                 end
487             end
488
489             % Se risulta visibile, verifico che la traiettoria
490             % rettilinea risultante sia sufficientemente distante
491             % dall'ostacolo. In parte parole calcolo la distanza della
492             % retta generata dal punto qf e verifico che sia almeno
493             % dObstMin.
494             if visible
495                 % Se l'angolo è pi/2, tan(pi/2) non è infinita poichè
496                 % c'è un valore massimo che i numeri possono assumere.
497                 % Perciò la singolarità propria della retta in forma
498                 % esplicita non si presenta.
499                 m = tan(gamma_g);
500                 q = Pa(2) - m*Pa(1);
501                 dist = abs((qf(2) - m*qf(1) - q) / sqrt(1 + m^2));
502                 if dist < 0.7*nav.dObstMin
503                     visible = 0;
504                 end
505             end
506
507
508         end
509     end
510
511
512     function [bottleneck, bottleneckPoint] = bottleneck(nav)
513
514         % Il punto bottleneckPoint è la media geometrica dei due punti
```

```
515             % più vicini e serve per localizzare dove si trova la ...
                    strettoia.
516             % Ovviamente sarà valido se bottleneck == 1.
517             bottleneckPoint = [];
518
519             % Assumo che non sia una strettoia.
520             % Se c'è un'ostacolo solo non ci sono problemi.
521             bottleneck = 0;
522             if length(nav.VisiblePoints) > 1
523                 % Cerco due punti la cui distanza è inferiore a 2*...
                        dObstMin.
524                 % Se esistono, allora siamo in una strettoia poichè non si
525                 % passa.
526                 obstaclePairs = nchoosek(1:length(nav.VisiblePoints), 2);
527                 for p = 1:size(obstaclePairs, 1)
528
529                     % Estraggo gli indici della coppia di ostacoli
530                     h = obstaclePairs(p, 1);
531                     k = obstaclePairs(p, 2);
532
533                     % Estraggo i punti degli ostacoli e verifico che ce ne
534                     % sia almeno uno per ostacoli
535                     yO_h = cell2mat(nav.VisiblePoints(h));
536                     yO_k = cell2mat(nav.VisiblePoints(k));
537                     if size(yO_h, 2) == 0 || size(yO_k, 2) == 0
538                         continue;
539                     end
540
541                     % Misuro la distaza di ogni punto di h rispetto ad ...
                            ogni
542                     % punto di k. Se trovo una coppia di punti più vicini
543                     % di 2*dObstMin, sono in una strettoia.
544                     for h_i = 1:size(yO_h, 2)
545                         for k_i = 1:size(yO_k, 2)
546
547                             % Estraggo la coppa di punti
548                             Ph = yO_h(:, h_i);
549                             Pk = yO_k(:, k_i);
550
551                             % Valuto la distanza e, nel caso, aggiorno le
552                             % uscite
553                             dist_hk = norm(Ph - Pk);
554                             if dist_hk < 2*nav.dObstMin
555                                 bottleneck = 1;
556                                 bottleneckPoint = mean([Ph Pk], 2);
557                                 break;
558                             end
559                         end
560
```

```
561                     % Se è gia stato trovata la strettoia, interrompi
562                     if bottleneck == 1
563                         break;
564                     end
565
566                 end
567
568                 % Se è gia stato trovata la strettoia, interrompi
569                 if bottleneck == 1
570                     break;
571                 end
572             end
573         end
574
575     end
576
577     function [fp, obstacleIndex] = farthestVisiblePoint(nav, Pa, Pb)
578
579         % Trovo il punto visibile più distante dalle posizoni Pa e Pb,
580         % rispettivamente dell'agente e del punto cieco
581         fp = [];
582         maxDistanceFound = -inf;
583         for h = 1:length(nav.VisiblePoints)
584
585             % Estraggo i punti dell'ostacolo corrente e verifico che ...
                    ce
586             % ne siano
587             yO = cell2mat(nav.VisiblePoints(h));
588             if (size(yO, 2) == 0) continue; end;
589
590             % Calcolo la distanza per ogni punto
591             for i = 1:size(yO, 2)
592
593                 % Estraggo il punto
594                 Pi = yO(:, i);
595
596                 % Se la distanza è maggiore di quella trovata finora,
597                 % aggiorno il valore massimo e salvo il punto corrente
598                 % come punto più distante
599                 dist_a = norm(Pi - Pa);
600                 dist_b = norm(Pi - Pb);
601                 dist = dist_a + dist_b;
602                 if dist > maxDistanceFound
603                     fp = Pi;
604                     maxDistanceFound = dist;
605                     obstacleIndex = h;
606                 end
607             end
608         end
```

```
609         end
610
611
612          function blindFlag = checkBlindSpot(nav)
613
614             %% Valuto se non esistono punti visibili e, nel caso, creo un ...
                    cerchio
615             blindFlag = 1;
616             for h = 1:length(nav.VisiblePoints)
617                 % Estraggo i punti visibili
618                 y_O = cell2mat(nav.VisiblePoints(h));
619                 if nav.state ≠ AutonomousNavigationState.BLIND && size(y_O...
                        , 2) > 5
620                     blindFlag = 0;
621                     break;
622                 elseif nav.state == AutonomousNavigationState.BLIND && ...
                        size(y_O, 2) > 5
623                     blindFlag = 0;
624                 end
625             end
626
627             if blindFlag == 0 && nav.state ≠ ...
                    AutonomousNavigationState.BLIND
628                 % Nel caso ce ne siano aggiorno il vettore dei punti ...
                        precedenti
629                 nav.OldVisiblePoints = nav.VisiblePoints;
630             end
631
632         end
633
634         function blindVisiblePoints(nav, Pa, theta)
635
636             % Nel caso non ce ne siano, prendo l'ostacolo che ne aveva ...
                    prima
637             % e ci costruisco una circonferenza centrata nell'ultimo punto
638             % Trovo quindi l'ostacolo che ne aveva almeno uno prima
639             yLast = [];
640             % Trova l'ostacolo più vicino fra i precedenti
641             distances = zeros(1, length(nav.OldVisiblePoints));
642             for h = 1:length(nav.OldVisiblePoints)
643
644                 % Estraggo i punti visibili
645                 y_O_old = cell2mat(nav.OldVisiblePoints(h));
646                 if size(y_O_old, 2) > 0
647
648                     % Distanza dal primo
649                     qi_dist = norm(y_O_old(:, 1) - Pa);
650
651                     % Distanza dall'ultimo
```

```
652                 qf_dist = norm(y_O_old(:, end) - Pa);
653
654                 % Prendo la minore
655                 if qi_dist > qf_dist
656                     distances(h) = qf_dist;
657                 else
658                     distances(h) = qi_dist;
659                 end
660             else
661                 distances(h) = inf;
662             end
663         end
664         % Trovo l'indice dell'ostacolo più vicino
665         [~, closestIdx] = min(distances);
666         % Prendo il punto più vicino, fra primo e ultimo nell'ostacolo
667         y_O_old = cell2mat(nav.OldVisiblePoints(closestIdx));
668         yLast = y_O_old(:, end);
669
670         if ~isempty(yLast)
671
672             % Creo la circonferenza attorno all'ultimo punto y_Last
673             % Angolo iniziale pari all'angolo fra la posizione ...
                    corrente
674             % dell'agente e l'ultimo punto
675             % Tale circonferenza deve essere generata nella
676             % direzione dell'agente in modo che ne prosegua il
677             % movimento. In altre parole, se l'ultimo punto si
678             % trova e sinistra dell'agente è necessario generarla
679             % in senso antiorario, altrimenti in senso orario.
680             gamma = atan2(Pa(2) - yLast(2), Pa(1) - yLast(1));
681
682             if nav.blindDirection == 0
683                 % Sono appena entrato -> trovo da che parte è
684                 gamma_diff = wrapToPi(gamma - theta);
685                 nav.gamma_blind = gamma;
686                 if gamma_diff < 0
687                     sx = 1;
688                 else
689                     sx = 0;
690                 end
691
692             elseif nav.blindDirection == 1
693                 sx = 1;
694             elseif nav.blindDirection == -1
695                 sx = 0;
696             end
697             y_O = [];
698             if sx
699                 disp('SX');
```

```
700                     % L'ostacolo è a sx -> antiorario
701                     nav.gamma_blind = gamma;
702                     for th = (nav.gamma_blind):(pi/10):(nav.gamma_blind+pi...
                            /2)
703                         newP = yLast + 0.05*nav.dObstMin * [cos(th); sin(...
                                th)];
704                         y_O = [y_O newP];
705                     end
706
707                     % Cerca i punti solo in senso antiorario
708                     nav.blindDirection = 1;
709                 else
710 %                     plot(yLast(1), yLast(2), 'm*');
711                     % L'ostacolo è a dx -> orario
712                     nav.gamma_blind = gamma;
713                     for th = (nav.gamma_blind-pi/2):(pi/10):(...
                            nav.gamma_blind)
714                         newP = yLast + 0.05*nav.dObstMin * [cos(th); sin(...
                                th)];
715                         y_O = [y_O newP];
716                     end
717                     y_O = flipdim(y_O, 2);
718
719                     % Cerca i punti solo in senso orario
720                     nav.blindDirection = -1;
721
722                 end
723
724                 if ~isempty(nav.blindHandlePlot)
725                     delete(nav.blindHandlePlot);
726                 end
727                 nav.blindHandlePlot = plot(y_O(1, :), y_O(2, :), '.', '...
                        Color', [0 162 216]/255);
728                 nav.VisiblePoints = {y_O};
729             end
730
731         end
732
733     end
734
735 end
```

D.2 Codice per il controllo degli agenti

Infine, a partire dalla classe precedente, si presenta il codice completo per la soluzione del problema in esame in simulazione.

```
clear all;close all;clc; warning off all;

disp('——————— Patrolling ———————');
disp(' ');
disp(' ');

%% Parametri tempo e iterazione
Δt = 3;   % Tempo totale loop feedbacklin (ogni loop 0.3s)
Tfrac = 10;     % Iterazioni feedback lin
tau = Δt + 0.7;      % Tempo loop MPC
N = 3;
MXITER = 500;

%% Visibilità ambiente
dMax = 15;
angoloVisuale = pi/2;

%% Definizione e plot ostacoli
Npoints = 400;
% C1
radius = 10;
center = [25 25]';
C1 = circleObstacle(center, radius, Npoints);
% C2
radius = 12;
center = [50 47]'; % 40
C2 = circleObstacle(center, radius, Npoints);
% R1
w = 25;
h = 25;
center = [75+w/2 65+h/2]';
R1 = rectangleObstacle(center, w, h, Npoints);
% R2
w = 40;
h = 35;
center = [w/2 100-h/2]';
R2 = rectangleObstacle(center, w, h, Npoints);
% Ostacoli
Obstacles = {C1, C2, R1, R2};
theta_init = 0;
% Plot
close all
figure(1)
daspect([1 1 1]);
hold on
grid on
for h = 1:length(Obstacles)
    y_OBSTACLE = cell2mat(Obstacles(h));
```

```
49      plot([y_OBSTACLE(1, :) y_OBSTACLE(1, 1)], [y_OBSTACLE(2, :) y_OBSTACLE...
            (2, 1)], 'r', 'LineWidth', 2);
50  end
51  disp('Obstacle defined...')
52  axis([5 60 0 45])
53
54
55  %% Definizione Goal e posizione iniziale
56  goal = [90; 100];
57  y_init = [10; 0];
58
59  %% Distanza da mantenere
60  d = 3;
61
62  %% Definizione robot virtuale
63  % R1
64  R1epuck = Epuck(y_init(1), y_init(2), theta_init, 0.01, Δt / Tfrac);
65  R1epuck.color = 'b.';
66
67  %% Definizione dei Set per TubeBased MPC
68  % Modello Agente parziale
69  As=[1 tau ; 0 1];
70  Bs=[0.5*tau^2 ; tau];
71  Cs=[1 0];
72  % Peso stato/ingresso
73  Qs = 5*eye(2);
74  Rs = 100*eye(1);
75  % Disturbo
76  w_arb = 1;
77  Ws = [1 1; 1 -1; -1,1; -1 -1] * w_arb;
78  % Vincoli stato
79  xbar = 200;
80  ybar = 200;
81  vbar = 10;
82  Xs = [xbar, vbar; -xbar, vbar; xbar, -vbar; -xbar, -vbar];
83  % Vincoli Ingresso
84  Us = [];
85  % Vincoli uscita
86  ΔZs = [-1; 1];
87
88  %% Definizione autovalori livello intermedio
89  autov = [0.3 0.31 0.32 0.33 0.34 0.35];
90
91  %% Definizione parametri livello di gen. della traiettoria
92  gamma = 1;
93  T = 10 * eye(2);
94  epsilon = 1.5;
95
96  %% Inizializzo oggetti per il controllo MPC
```

```
97  R1mpc = MPCCustomTrajectory(As, Bs, Cs);
98  R1mpc.setupRobustMPCLayer(N, Qs, Rs, Xs, Us, ΔZs, Ws);
99  R1mpc.setupStateInputReferenceLayer(autov);
100 R1mpc.setupTrajectoryLayer(epsilon);
101 R1mpc.initialize(R1epuck.position);
102
103 %% Istanza AutonomousNavigation e configurazione
104 navigator = AutonomousNavigation(R1mpc, Obstacles, angoloVisuale, dMax, d)...
        ;
105 navigator.goal = goal;
106 navigator.onlyPatrolling = false;
107 if ¬navigator.onlyPatrolling
108     plot(goal(1), goal(2), 'bp', 'MarkerSize', 15);
109 end
110
111 %% Loop
112 % Handle plot punti visibile
113 handleVisPts = zeros(1, length(Obstacles));
114 NoldPoints = 20;
115 handlePos = zeros(NoldPoints, 1);
116 handleTraj = zeros(NoldPoints, 1);
117 oldPointsCounter = 1;
118 % Vettori tempo, posizione e riferimento
119 time = 0;
120 % Iterazioni
121 Niter = 0;
122 % Vettori per salvare la prova
123 yTilde = R1epuck.position;
124 xTilde = R1epuck.linState;
125 xState = R1epuck.state;
126 while 1
127
128     %% Iterazione
129     Niter = Niter + 1;
130
131     %% Orientamento agente
132     theta = wrapToPi(R1epuck.state(3));
133
134     %% Trovo i punti visibili e li plotto
135     VisiblePoints = navigator.getVisiblePoints(R1epuck.position, theta);
136     % Plot
137     for h = 1:length(VisiblePoints)
138         % Estraggo i punti visibili
139         y_O = cell2mat(VisiblePoints(h));
140
141          % Cancello punti precedenti
142         if handleVisPts(h) > 0
143             delete(handleVisPts(h));
144             handleVisPts(h) = 0;
```

```
145         end
146
147         % Plot punti visibli
148         if ¬isempty(y_O)
149             handleVisPts(h) = plot(y_O(1, :), y_O(2, :), 'ks');
150         end
151     end
152
153     %% Step trajectory layer
154     [u_R1, y_next] = navigator.step(R1epuck.linState, theta);
155
156     %% Feedback linearization
157     if Niter > N+2
158         for j = 1:Tfrac
159             R1epuck.step(u_R1(1), u_R1(2));
160         end
161     end
162
163     %% Plot
164     if oldPointsCounter == NoldPoints
165         if handlePos(1) > 0
166             delete(handlePos(1));
167         end
168         if handleTraj(1) > 0
169             delete(handleTraj(1));
170         end
171         handlePos(1) = [];
172         handleTraj(1) = [];
173     end
174     % Posizione agente
175     handlePos(oldPointsCounter) = plot(R1epuck);
176     % Riferimento
177     handleTraj(oldPointsCounter) = plot(y_next(1), y_next(2), 'gd');
178     if oldPointsCounter < NoldPoints
179         oldPointsCounter = oldPointsCounter + 1;
180     end
181     drawnow;
182
183
184     % Aggiorna vettori da salvare
185     yTilde = [yTilde R1epuck.position];
186     xTilde = [xTilde R1epuck.linState];
187     xState = [xState R1epuck.state];
188
189     % Update time
190     time = time + tau;
191
192     % Calcolo la distanza dal goal e fermo la simulazione se
193     % sufficientemente vicino
```

```
194     if norm(goal - R1epuck.position) < 0.2
195         break;
196     end
197
198 end
199
200 %% Save results
201 save Sim_patrolling.mat yTilde xTilde xState time regions sensings Klf ...
        stateMachineStateMemory
202 % I dati di ogni robot sono posti per riga in yTilde xTilde xState.
```

Bibliografia

[1] Khoukhi Amar e Shahab Mohamed. "Stabilized Feedback Control of Unicycle Mobile Robots". In: *International Journal of Advanced Robotic Systems* 10 (2013), pp. 187–194.

[2] Tucker Balch e Ronald C. Arkin. "Behavior-based formation control for multirobot teams". In: *Robotics and Automation, IEEE Transactions on* 14.6 (1998), pp. 926–939.

[3] Yongcan Cao e Wei Ren. "Containment control with multiple stationary or dynamic leaders under a directed interaction graph". In: *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*. 2009, pp. 3014–3019.

[4] Yongcan Cao, Wei Ren e Magnus Egerstedt. "Distributed containment control with multiple stationary or dynamic leaders in fixed and switching directed networks". In: *Automatica* 48 (2012), pp. 1586–1597.

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

[6] Zhou Chao et al. "Collision-free UAV formation flight control based on nonlinear MPC". In: *Electronics, Communications and Control (ICECC), 2011 International Conference on*. 2011, pp. 1951–1956.

[7] Marcello Farina e Riccardo Scattolini. "Distributed predictive control: A non-cooperative algorithm with neighbor-to-neighbor communication for linear systems". In: *Automatica* 48.6 (2012), pp. 1088 – 1096.

[8] R. Fierro et al. "Hybrid Control of Formation of Robots". In: *International Conference on Robotics and Automation* (2001), pp. 157–162.

[9] Matthew Flint, Marios Polycarpou e Emmanuel Fernandez-Gaucherand. "Cooperative path-planning for autonomous vehicles using dynamic programming". In: *World Congress*. Vol. 15. 2004, pp. 1303–1303.

[10] Luca Galbusera, Giancarlo Ferrari-Trecate e Riccardo Scattolini. "A Hybrid model predictive control scheme for containment and distributed sensing in multi-agent systems". In: *System & Control Letters* 62 (2013), pp. 413–419.

[11] Luca Giulioni. "Studio e Applicazioni di Tecniche di Controllo Predittivo per il Posizionamento e il Coordinamento di Agenti Mobili". Tesi di laurea mag. Politecnico di Milano, 2012.

[12] Hiroki Kawakami e Toru Namerikawa. "Virtual Structure Based Target-enclosing Strategies for Nonholonomic Agents". In: *International Conference on Control Applications*. 2008, pp. 1043–1048.

[13] Y. Kawauchi, M. Inaba e T. Fukuda. "A principle of distributed decision making of Cellular Robotic System (CEBOT)". In: *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. 1993, 833–838 vol.3.

[14] I. Kolmanovsky e N.H. McClamroch. "Developments in nonholonomic control problems". In: *Control Systems, IEEE* 15.6 (1995), pp. 20–36.

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

[16] M. Anthony Lewis e Kar-Han Tan. "High Precision Formation Control of Mobile Robots Using Virtual Structures". In: *Autonomous Robots* 4 (1997), pp. 387–403.

[17] Jianzhen Li, Wei Ren e Shengyuan Xu. "Distributed Containment Control with Multiple Dynamic Leaders for Double-Integrator Dynamics Using Only Position Measurements". In: *Automatic Control, IEEE Transactions on* 57.6 (2012), pp. 1553–1559.

[18] L. Marìn et al. "Implementation of a bug algorithm in the e-puck from a hybrid control viewpoint". In: *Methods and Models in Automation and Robotics (MMAR), 2010 15th International Conference on*. 2010, pp. 174–179.

[19] Maja J. Matarič. "From Local Interactions to Collective Intelligence". In: *The Biology and Technology of Intelligent Autonomous Agents*. A cura di Luc Steels. Vol. 144. NATO ASI Series. Springer Berlin Heidelberg, 1995, pp. 275–295. ISBN: 978-3-642-79631-9.

[20] Alexey S. Matveev, Hamid Teimoori e Andrey V. Savkin. “A method for guidance and control of an autonomous vehicle in problems of border patrolling and obstacle avoidance”. In: *Automatica* 47.3 (2011), pp. 515 –524.

[21] Alexey S. Matveev, Chao Wang e Andrey 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 (2012), pp. 769 –788.

[22] David Mayne et al. “Constrained model predictive control: Stability and optimality”. In: *Automatica* 36 (2000), pp. 789–814.

[23] David Mayne et al. “Robust output feedback model predictive control of contrained linear systems”. In: *Automatica* 42 (2006), pp. 1217–1222.

[24] Reza Olfati-Saber, J. Alex Fax e Richard M. Murray. “Consensus and Cooperation in Networked Multi-Agent Systems”. In: Proceedings of the IEEE 95 (2007), pp. 215–233.

[25] Wei Ran, W. Beard e Ella M. Atkins. “Information consensus in multivehicle cooperative control”. In: *Control Systems Magazine* 27 (2007), pp. 71–82.

[26] Riccardo Scattolini. “Architectures for distributed and hierarchical Model Predictive Control - A review”. In: *Journal of Process Control* 19.5 (2009), pp. 723 –731.

[27] Riccardo Scattolini e Lalo Magni. *Complementi di Controlli Automatici*. Pitagora Editrice Bologna, 2006.

[28] D.H. Shim, H.J. Kim e S. Sastry. “Decentralized nonlinear model predictive control of multiple flying robots”. In: *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*. Vol. 4. 2003, pp. 3621–3626.

[29] Jean-Jacques Slotine e Weiping Li. *Applied Nonlinear Control*. Prentice Hall, ott. 1991. ISBN: 0130408905.

[30] Glenn Wagner, Minsu Kang e H. Choset. “Probabilistic path planning for multiple robots with subdimensional expansion”. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. 2012, pp. 2886–2892.

[31] Bai Wenfeng, Han Lina e Shi Yunfeng. “An improved A* algorithm in path planning”. In: *Computer, Mechatronics, Control and Electronic Engineering (CMCE), 2010 International Conference on*. Vol. 3. 2010, pp. 235–237.

[32] Feng Xie e Rafael Fierro. “On Motion Coordination of Multiple Vehicles with Nonholonomic Constraints”. In: *American Control Conference*. 2007, pp. 1888–1893.

[33] Mohamed Younis e Kemal Akkaya. “Strategies and techniques for node placement in wireless sensor networks: A survey”. In: *Ad Hoc Networks* 6 (2008), pp. 621–655.

[34] Yi Zhu et al. “A new Bug-Type navigation algorithm considering practical implementation issues for mobile robots”. In: *Robotics and Biomimetics (ROBIO)*. 2010, pp. 531–536.