Approccio comportamentale al controllo coordinato di WMR

G. Barbera, P. Calore, G. Cosi
Università degli studi di Padova
Dipartimento di Ingegneria dell'Informazione
Padova, Italy, 35100

28 Marzo 2008

Abstract

Con questo lavoro si intende affrontare in modo completo il problema della navigazione autonoma per squadre di veicoli: in particolare si fa riferimento al Null Space Behavioral Control, una tipologia di controllo comportamentale che organizza in struttura gerarchica i task da eseguire, semplificando notevolmente la complessità computazionale e consentendo di affrontare anche missioni molto complesse. In particolare viene affrontato il delicato problema del controllo delle collisioni, analizzando le problematiche teoriche e pratiche che possono insorgere e proponendo un algoritmo di controllo valido in un vasto campo di applicazioni. Vengono studiati quindi i principali task di controllo coordinato di veicoli, e.g. raggiungimento di un target, tracking del baricentro della squadra, convergenza ad un punto con varianza desiderata, movimento in formazione. Tutti i task sono stati simulati per veicoli anolonomi di tipo uniciclo, e i risultati sono supportati da una serie di verifiche sperimentali su un gruppo di WMR (ePuck) controllati con una telecamera.

Introduzione

Negli ultimi decenni la robotica industriale ha rivestito un ruolo fondamentale nei processi produttivi ed è stata tra i protagonisti della ricerca nel campo dell'automazione; sempre maggiori attenzioni sono state riposte inoltre nella robotica cosiddetta di servizio, in grado di ridurre enormemente la presenza dell'utenza umana nelle più svariate applicazioni (soprattutto in quelle più pericolose) quali la sorveglianza di vaste aree, il trasporto e la raccolta di materiale, azioni di esplorazione o di soccorso.

Questo articolo si inserisce alla base di tali problematiche, e affronta il problema del path planning e del coordinamento di una squadra di robot, analizzando le prestazioni di uno degli algoritmi di controllo più avanzati tra quelli presenti in letteratura, il Null Space Based Control. Viene presentata l'interpretazione geometrica di questo recente approccio al problema, con una particolare attenzione alla fase implementativa, per la quale vengono introdotti personali accorgimenti che ne semplificano l'utilizzo per il compimento di task multipli. E' riportata inoltre una completa panoramica circa il controllo di veicoli, per una valutazione delle principali differenze rispetto ad altri approcci.

Stato dell'arte

Indipendentemente dal tipo di applicazione o dal numero di agenti mobili in gioco, il problema della pianificazione del moto (*motion planning*) è l'aspetto che fino ad ora è stato al centro delle ricerche e degli studi, con particolare attenzione al caso anolonomo: ogni tipo di veicolo su ruote, imbarcazione o velivolo può essere infatti rappresentato in prima approssimazione da un modello anolonomo caratterizzato da equazioni

della cinematica fortemente non lineari, e la categoria di veicoli su ruote WMR (Wheeled Mobile Robot) risulta attualmente la più importante a livello accademico.

Per una panoramica completa sui diversi tipi di WMR si faccia riferimento a [3], nel quale vengono illustrate le differenze strutturali dei vari veicoli WMR, con una particolare attenzione alla modellizzazione cinematica e dinamica. Per una trattazione più rigorosa e approfondita si veda [9]. Nel nostro caso, per le dimensioni e i pesi contenuti dei robot utilizzati, e per le caratteristiche dei motori (gli E-puck montano due motori a passo indipendenti) verrà adottata la modellizzazione cinematica degli unicicli, scelta che ci permetterà di semplificare notevolmente il problema e di concentrare maggiori attenzioni sull'implementazione del controllo e sugli algoritmi di coordinazione.

La pianificazione del moto di un uniciclo (e in generale degli WMR) viene fatta in considerazione del tipo di *task* che si vuole far eseguire al robot, e dell'ambiente sul quale esso si muove; gli algoritmi di controllo del moto possono essere distinti in tre principali categorie: *path following*, *trajectory tracking* e *posture stabilization*:

- *path following*: con l'applicazione di tale tecnica si vuole che il robot insegua un particolare cammino geometrico nello spazio cartesiano (generalmente rette e circonferenze) partendo da una generica posizione iniziale, facente parte o meno del cammino geometrico, senza dover soddisfare determinate specifiche temporali.
  Viene dunque data una descrizione geometrica del cammino cartesiano assegnato: tale informazione è solitamente espressa in forma parametrica e quindi esprime il moto desiderato in funzione del parametro. Per questo particolare tipo di task la dipendenza dal tempo non è rilevante in quanto l unico parametro da controllare è legato all errore geometrico di piazzamento tra robot e cammino da seguire. Per questo motivo si assegna ad uno dei due ingressi la velocità di avanzamento del veicolo (che assumerà un valore arbitario costante o variabile con il tempo secondo una legge nota), mentre il secondo ingresso sarà utilizzato per il controllo. Il controllo avrà quindi il compito di stabilizzare a zero l'errore tra il cammino e la posizione effettiva del veicolo (distanza d in figura). Tale errore è in generale funzione scalare di tutte le variabili di stato del sistema meno una legata alla rinuncia del controllo sulla velocità lineare (come ad esempio la distanza relativa tra percorso e veicolo).

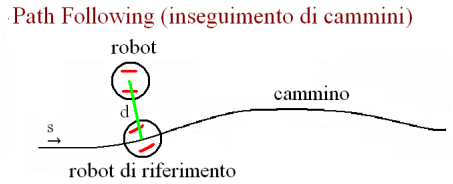

Figura 1: *tecnica di inseguimento path-following*

- *posture stabilization* il robot deve raggiungere una determinata configurazione finale desiderata partendo da una generica configurazione iniziale.

- *trajectory tracking*: con tale tecnica si impone che il robot debba raggiungere e seguire una determinata traiettoria geometrica nello spazio cartesiano, cioè un cammino assegnato ad una determinata legge temporale partendo da una generica posizione iniziale, che può eventualmente far parte della traiettoria.
  Anche se si può separare la traiettoria da inseguire in un cammino geometrico parametrizzato ed una una legge temporale per il parametro, tale separazione non è strettamente necessaria. Spesso è più conveniente considerare lo spazio della traiettoria come la desiderata evoluzione nel tempo dalla posizione del robot. Sarà dunque necessario stabilizzare a zero l'errore bidimensionale cartesiano (e_p in figura), legato alla differenza tra configurazione attuale e configurazione desiderata, utilizzando entrambi gli ingressi di controllo. Questo porta ad effettuare un controllo in tutto lo spazio di stato senza dover rinunciare ad alcun grado di libertà del sistema.

In questo progetto si è preferito utilizzare la tecnica del *trajectory tracking* rispetto a quella del *path following* e del *posture stabilization*, in quanto più robusta, precisa, presenta un minore errore a regime ed inoltre consente di inseguire una traiettoria con

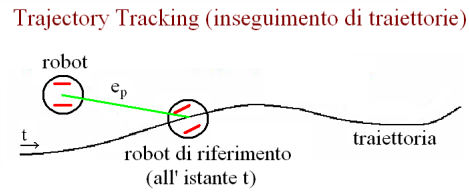

Figura 2: *tecnica di inseguimento trajectory tracking*

una legge oraria, indispensabile per raggiungere gli obiettivi proposti.

Per risolvere le tre problematiche sopra presentate si possono utilizzare svariati metodi. I due più importanti ed utilizzati sono: il *controllo non lineare secondo Lyapunov* e il *controllo mediante feedback-linearization*. Per maggiori approfondimenti sul controllo non lineare secondo Lyapunov, si veda [1], [2] e [3]. In questo progetto si è scelto di utilizzare il controllo mediante *feedback-linearization*, di cui in seguito viene data una trattazione più completa e la cui letteratura è molto vasta (si veda [3]). Se si vuole invece trovare un confronto implementativo tra le due tecniche si veda [5].

Nel caso di condizioni ambientali più complesse per la presenza di ostacoli statici, dinamici o di altri agenti, gli algoritmi di pianificazione del moto subiscono un notevole aumento di complessità in quanto si presentano aspetti diversi dal semplice controllo in velocità : la scelta del percorso ottimo in un ambiente delimitato, algoritmi per evitare le collisioni, assegnazione di priorità diversificate a più task, analisi ed implementazione di algoritmi comportamentali e decisionali etc. Il problema di ricerca di percorsi in assenza di collisioni è di importanza fondamentale nelle applicazioni di robot mobili, in quanto rappresenta il primo vero passo verso la completa automatizzazione degli stessi.

Normalmente il problema del controllo di robot in ambienti complessi si divide quindi in due fasi: una prima fase di creazione del percorso, e una fase di attuazione del controllo vero e proprio, utilizzando generalmente gli algoritmi citati precedentemente.

Indipendentemente dal modo in cui gli agenti ricevono informazioni relative agli ostacoli da evitare (sistema GPS, telecamere, sensori a bordo, ultrasuoni, conoscenza a priori etc.) gli algoritmi di motion planning si dividono in gruppi, che si differenziano principalmente per due aspetti: il metodo attraverso il quale viene calcolato il percorso desiderato e il modo in cui viene rappresentato lo spazio libero (*free space*: l'area nella quale è concesso il movimento). Rispettivamente si hanno quindi algoritmi di path planning tenenti conto o meno dei vincoli anolonomi che caratterizzano i WMR e algoritmi che rappresentano lo spazio e gli ostacoli in modo esatto o approssimato. In [10] si trova un'analisi completa di questi criteri.

Negli algoritmi di *roadmap* si crea una mappa delle possibili traiettorie permesse (Fig. 3) , e dei punti raggiungibili, adeguatamente connessi tra di loro e a seconda dei casi si sceglie il percorso migliore. In [11]

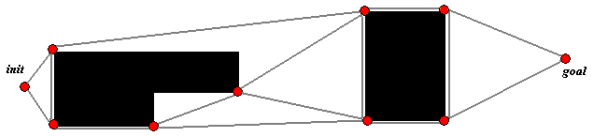

Figura 3: *Set di punti e traiettorie sicure che compongono una semplice roadmap. Il grafo risultante è detto anche grafo della visibilità*

gli autori presentano una generalizzazione della rete di Kohonen utilizzata nell'approssimazione del diagramma di Voronoi, criterio spesso richiamato nella creazione delle roadmaps per la pianificazione del moto di robot [10].

Nei metodi cosiddetti di *decomposizione* lo spazio viene suddiviso in celle che non si sovrappongono, e i percorsi da generare sono ricavati da un grafo delle connessioni che memorizza le relazioni di adiacenza tra le celle. Tali percorsi saranno quindi una successione predefinita di celle entro le quali il robot dovrà muoversi, per esempio passando per il baricentro delle stesse.

Gli algoritmi di *decomposizione approssimata* si differenziano da quella *esatta* per il modo in cui si divide lo spazio libero: un famoso algoritmo (divide and label) appartenente alla prima famiglia esegue una decomposizione *quadtree* utilizzando celle dalla forma semplice e predefinita, normalmente rettangolare, in modo da ridurre la complessità numerica di un algoritmo altrimenti decisamante complesso (Fig. 4). In

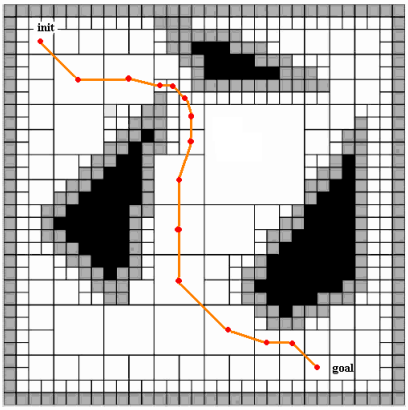

Figura 4: *Suddivisione quadtree dello spazio. Le celle bianche rappresentano lo spazio libero, le nere lo spazio occupato da ostacoli, e quelle grige lo spazio misto.*

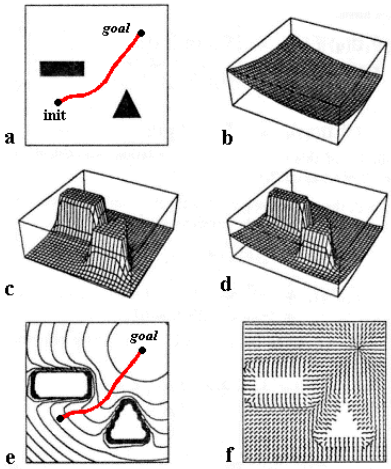

Figura 5: *La figura mostra un campo di potenziale attrattivo (b), un campo di potenziale repulsivo (c), la somma dei due (d) e il percorso calcolato per raggiungere il goal, in rosso.*

[12] il metodo che prende il nome di *Markov Decision Processes* viene implementato per la ricerca di un percorso in un ambiente decomposto secondo quadtree. Algoritmi più avanzati utilizzano una decomposizione a celle triangolari, con l'eventuale collasso dei lati in esubero [13].

Un altro algoritmo presentato in [10] è il *metodo dei potenziali* il quale si basa sulla creazione di un campo di potenziale artificiale nello spazio libero da ostacoli. Tale potenziale viene calcolato come la somma di due componenti, la prima ha lo scopo di attrarre il robot verso lo stato finale assegnando a ciascun punto della mappa un potenziale che è funzione crescente della distanza dallo stato finale. Per quanto riguarda la seconda componente, di tipo repulsivo, essa pone ad un potenziale elevato i punti appartenenti agli ostacoli, mentre quelli appartenenti allo spazio libero vengono messi ad un potenziale inversamente proporzionale alla distanza dagli ostacoli stessi. A questo punto il percorso viene tracciato seguendo le linee di campo (Fig. 5). Questo metodo soffre però del problema dei minimi locali che si formano nel potenziale calcolato.

Gli algoritmi citati hanno generalmente il compito di generare una traiettoria sicura ma olonoma , che non consideri quindi i vincoli della cinematica (massimo raggio di curvatura di un robot, l'impossibilità di traslare lateralmente, etc.), traiettoria che verrà poi approssimata da una concatenazione di manovre elementari che altro non sono l'attuazione del controllo vero e proprio. In letteratura si trovano svariati esempi di generazione di traiettorie esatte (non-olonome): in particolare in [14], gli autori propongono un miglioramento nella generazione della prima traiettoria geometrica , modificando il metodo dei potenziali tramite una funzione costo che tenga conto della anolonomia che caratterizza i WMR.

La creazione di traiettorie esatte, non-olonome e quindi immediatamente attuabili, rappresenta un capitolo importante nel controllo dei robot. Si veda [15] nel quale viene proposto uno studio sulla pianificazione del moto anolonomo utilizzando sinusodi come ingressi per sistemi sterzanti. Si veda ancora [9] per una tarttazione generale del controllo non-olonomo.

Un ulteriore approccio consiste nella cosiddetta *quantizzazione del controllo*: anzichè quantizzare il tempo, lo stato del sistema, o i valori di ingresso ed uscite si sceglie un numero finito di traiettorie di controllo, dette primitive di movimento, la combinazione delle quali genera i moti ammissibili (Fig. 6). Si può trovare una prima presentazione di tale criterio

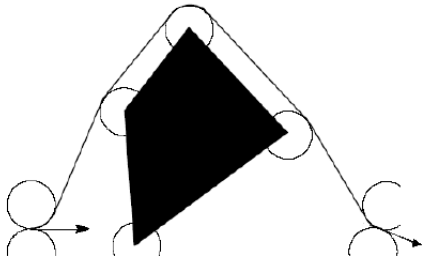

Figura 6: *Percorso in un ambiente con ostacolo, generato dalla combinazione di traiettorie primitive.*

in [10], e la sua implementazione in [16], nel quale si definiscono, per esempio, manovre a velocità angolare costanti. Questo tipo di controllo risulta molto appropriato in sistemi caratterizzati da tempi di campionamento troppo lunghi, come è stato osservato in [17].

Uno degli aspetti più interessanti nello studio dei WMR è senza dubbio il controllo coordinato di una squadra di più robot, sia per la completezza dell'argomento, sia per gli sviluppi teorici e pratici degli ultimi anni.

Numerosi sono i vantaggi che derivano dall'utilizzo di più agenti; primo tra tutti la possibilità di ridurre il costo del sistema, costruendo un numero maggiore di robot specifici, piuttosto che un singolo robot con tutte la capacità computazionali, meccanichee e dinamiche in grado di risolvere task molto complessi. Generalmente una squadra di robot è infatti composta da varietà di robot eterogenei, ognuno dei quali è specificamente costruito in modo da poter risolvere semplici sub-task. Inoltre si ha un notevole incremento della robustezza dell'intero sistema, che risulta quindi operativo anche in condizioni di malfunzionamento di alcuni degli agenti in gioco. Anche task semplici possono essere eseguiti in modo decisamente più efficiente e rapido da una squadra di robot, ma indipendentemente dal compito assegnato, il *controllo in formazione* risulta essere uno dei temi centrali del controllo coordinato.

In letteratura esistono numerosi approcci al problema, tanto che elencare tutte le tipologie di controllo di squadra sarebbe un compito assai difficile; ci limitiamo a darne una classificazione di base, richiamando i concetti principali e analizzando le differenze:

- *Leader following*: un veicolo viene designato *leader*, e ha il compito di tracciare la traiettoria che dovrà essere seguita dagli altri robot con un certo offset, anche tempo-variante, che determina il tipo di formazione; varianti più avanzate mostrano la possibilità di designare leader multipli, e sub-leader, creando strutture gerarchice ad albero. Tale tipologia di controllo è particolarmente indicata per applicazioni come movimento di una catena di veicoli (si pensi ad esempio al movimento di veicoli su strada), ma non sfrutta le informazioni sullo stato *globale* del sistema [7]. Un ulteriore punto debole di questo tipo di soluzione è la mancanza di robustezza: un malfunzionamento del leader comporta infatti il malfunzionamento dell'intero sistema.

  In [18] gli autori presentano un'implementazione del controllo di una squadra di robot mobili anolonomi coordinati secondo il criterio del leader following; in questo caso, considerando i vincoli anolonomi che caratterizzano i robot da controllare, le posizioni relative degli inseguitori non sono fisse, ma possono variare entro un cono predefinito.

- *Virtual structure*: viene creata una struttura virtuale che contiene le posizioni desiderate dei veicoli ad ogni istante; tale struttura è creata in modo da permettere traslazioni, rotazioni e dilatazioni o restringimenti della struttura, definendone la dinamica e ricavando da questa i controlli da attuare per i singoli agenti ; in questo modo si ha un maggior grado di controllo sul comportamento globale del sistema, ma non si ha un feedback di controllo sulla formazione (ogni veicolo si muove indipendentemente da ciò che fanno gli altri) [6]. L'approccio tramite virtual structure può essere affrontato come un particolare problema di leader following: ad un punto arbitrario della struttura, generalmente il baricentro della formazione desiderata, viene assegnato il ruolo di leader, mentre agli agenti quello di follower; si parla in questo caso di *virtual leader*.

  In [19] viene messo a confronto il comportamento di una squadra di agenti controllati secondo il procedimento virtual leader e un controllo di tipo

behavioral, circa la frequente problematica del consensus dinamico. Un'altra applicazione del metodo virtual leader viene proposto in [20], nel quale vengono analizzate semplici manovre (traslazione, rotazione ed espansione) in formazione di una squadra di veicoli, utilizzando l'algoritmo dei potenziali artificiali.

- *Behavioral*: il cuore di questo tipo di algoritmi è l'idea di poter lavorare sull'implementazione di diversi singoli comportamenti per ogni agente, e derivarne l'azione di controllo globale come una combinazione di questi comportamenti, pesata sulle informazioni disponibili del sistema. Tali comportamenti sono associati a semplici *subtask* quali il raggiungimento di configurazioni arbitrarie (goal), evitare ostacoli o collisioni con altri agenti, muoversi in formazioni predefinite, rimanere entro un certo raggio di comunicazione, etc. Grande pregio di questo tipo di controllo risulta essere la naturalezza nel derivare strategie di controllo per il compimento di obiettivi altrimenti molto complessi, e la naturale propensione per un'implementazione di tipo decentralizzato, come viene proposto in [4]. E' comunque da notare che un'analisi matematica approfondita di questo tipo di procedimento risulta essere piuttosto difficile, e non sempre è garantita la stabilità del sistema.

L'approccio Behavioral

Il problema del controllo coordinato di una squadra di robot anolonomi se affrontato mediante un approccio di tipo *behavioral* risulta essere un argomento molto vario e diversificato. In letteratura tale criterio vine applicato nelle più svariate situazioni, presentandosi in diverse varianti a seconda dello scopo; una trattazione completa anche in questo caso risulterebbe alquanto dispersiva, daremo quindi una panoramica sulle principli tre possibilità di implementazione. Nei vari casi, in base alla conoscienza dello stato del robot e alle informazioni relative all'ambiente (ostacoli, altri agenti nelle vicinanze, stato dell'eventuale configurazione desiderata, etc), vengono calcolati in parallelo i controlli per i *subtask* e successivamente assemblati in modo arbitrario. I controlli, come verrà approfondito più avanti, sono controlli di velocità, implementati generalmente mediante l'algoritmo di *feedback linearization*.

Come è stato anticipato nella sezione precedente, il controllo *behavioral* permette di selezionare un insieme di comportamenti predefiniti, in modo che cooperino in tempo reale al compimento del task richiesto. I singoli *subtask* indipendenti con le relative leggi di controllo sono regolati da una struttura gerarchica dinamica che ne regola le priorità, e garantisce un comportamento globale che soddisfi ogni singolo obiettivo. La cooperazione tra i comportamanti è gestita da un sistema di pesi che varia a seconda della situazione impostando priorità maggiore agli obiettivi primari. Per esempio, evitare le collisioni è chiaramente un task fondamentale, al quale viene assegnata in ogni caso la priorità primaria per preservare l'integrità dell'hardware utilizzato. Il modo col quale l'algoritmo discrimina i diversi task in base alle priorità, rappresenta la principale differenza tra i tre tipi di controllo behavioral: *competitive* , *cooperative* e *null-space*.

Competitive

La coordinazione in questo caso viene vista come una competizione tra i vari comportamenti; soltanto un comportamento vince e soltanto il suo rispettivo controllo viene inviato all'agente per essere attuato (Fig. 7). Colui che ha il compito di impostare la gerarchia dei *subtask* in base alle informazioni disponibili dai sensori è chiamato *supervisore*, o *arbitro*; grazie alla sua azione i controlli secondari vengono inibiti e vengono soddisfatti i relativi task soltanto dopo il

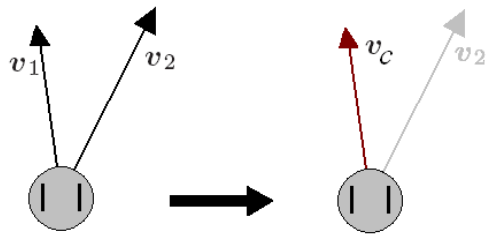

Figura 7: *Caso di due comportamenti in conflitto. Le velocità v_1 e v_2 sono le uscite di due subtask di diversa priorità, e v_c è l'uscita del comportamento vincente.*

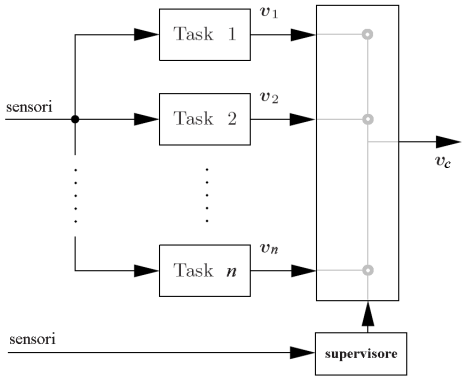

Figura 8: *Schema semplificato del controllo a livelli per un sistema di n task, caratteristico dell'approccio competitivo.*

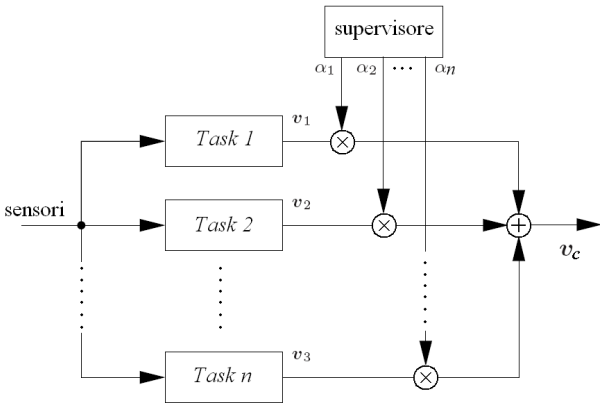

Figura 9: *Schema semplificato del controllo cooperativo per un sistema di n task*

successo del task primario. Per questo motivo si parla talvolta di struttura a livelli, *layered* in letteratura (Fig. 8).

Cooperative

La fusione dei comportamenti in gioco permette l'uso delle uscite di più di un solo comportamento contemporaneamente. L'uscita risultante è in questo caso una somma di tutti i comandi di movimento opportunamente moltiplicata per un vettore di guadagni calcolato dal supervisore in base ai dati provenienti dai sensori (Fig. 9). In questo modo oltre all'ordine gerarchio delle priorità può essere modificato dinamicamente anche il peso di ogni singolo controllo.

La sostanziale differenza col metodo competitivo è che l'approccio cooperativo realizza una combinazione lineare delle uscite elaborate per ogni task, e la dinamica risultante sarà quindi una soluzione più naturale e completa. In Fig. 10 la cooperazione di più task porta all'elaborazione di un'uscita che è il compromesso corretto per il compimento di task multipli, ed è da notare che il criterio competitivo può sempre essere ottenuto anche in ambito cooperativo con una adeguata scelta del vettore dei pesi. In [4] è proposto una particolare implementazione del controllo cooperativo, nel quale vengono calcolati i controlli per elementari manovre in formazione (EFM), volti all'eliminazione di un'errore a regime composto da componenti relative a due sub-task principali: il raggiungimento di un goal arbitrario e il mantenimento della formazione. Un altro significativo esempio di controllo *behavioral* cooperativo si trova in [8]. Gli autori analizzano le prestazioni di un controllo in formazione di una squadra di robot in laboratorio e di una squadra di veicoli DARPA, utilizzando il noto metodo del *motor-schema*: i risultati ottenuti, a livello di prestazioni e di errori relativi alle posizioni desiderate, risultano ottimi e migliori rispetto ad esperimenti simili eseguiti secondo il criterio del *leader following*. Un particolare pregio del controllo behavioural in formazione emerge nel frequente caso di presenza di ostacoli: i robot sembrano lavorare insieme per minimizzare l'errore globale sulla formazione, aspettando per esempio quei robot che rimangono impediti dietro agli ostacoli.

Gli esempi appena citati sono caratterizzati da un'implementazione di tipo decentralizzato; tale scelta solleva importanti problematiche, legate principalmente al tipo di comunicazione esistente tra i diversi agenti. Un sistema decentralizzato porta comunque importanti vantaggi rispetto al caso centralizzato, in quanto risulta decisamente più robusto in presenza di guasti o malfunzionamenti e attuabile anche in assenza di un feedback di posizione globale (GPS o videocamere). In [21] troviamo quindi un sistema completamente decentralizzato multi robot formato da agenti

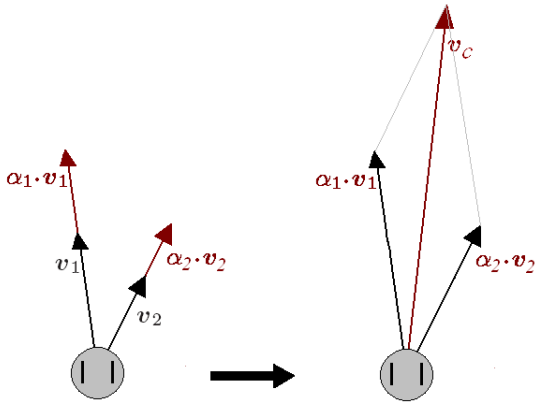

Figura 10: *Caso di due comportamenti cooperanti. Le velocità v_1 e v_2 sono le uscite di due subtask mentre i diversi coefficienti α_1 e α_2 calcolati dal supervisore corrispondono alle priorità degli obiettivi. v_c è l'uscita del comportamento risultante, somma vettoriale delle uscite.*

eterogenei cooperanti al compimento di una missione complessa; l'intero progetto sottolinea l'importanza della valutazione di situazioni anomale e presenta soddisfacenti risultati in quanto a robustezza, flessibilità e affidabilità in caso di condizioni inaspettate (come guasti, cambiamenti improvvisi dell'ambiente, aggiunta di nuovi agenti).

NSB: Null Space Behavioral

Negli ultimi anni è stato affrontato un nuovo approccio al problema del controllo di tipo *behavioral*, denominato *Null-Space-Based Behavioral control (NSB)*. Il lavoro [22], nel quale un robot manipolatore si muove in un ambienete con un ostacolo, può essere considerato la prima vera applicazione dell'algoritmo NSB. Negli algoritmi visti precedentemente, è necessaria una fase di tuning euristico dei parametri che gestiscono l'assegnazione delle priorità e i relativi coefficienti per il raggiungimento dell'obiettivo del task con la più alta priorità.

Anche il controllo NSB prevede l'implementazione di un supervisore per l'assegnazione delle priorità ai subtask disponibili, determinando quindi un ordine gerarchico dipendente dalle informazioni disponibili . Ma il punto fondamentale di questo tipo di controllo è l'interpretazione geometrica dei conflitti che vengono a crearsi al momento della cooperazione di questi subtask. Come nei casi precedenti ogni obiettivo determina una particolare uscita che viene elaborata come se fosse attivo un singolo task; a questo punto, prima di aggiungere il proprio contributo alla determinazione della velocità totale di controllo, le uscite dei task di priorità inferiore vengono proiettate sullo *spazio nullo* dell'uscita del task di priorità immediatamente superiore, in modo da rimuoverne soltanto le componenti che sono in conflitto (Fig. 11).

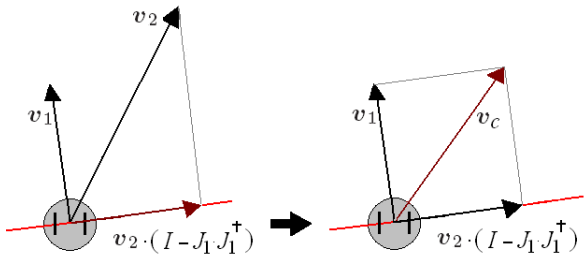

Figura 11: *Esempio di due comportamenti cooperanti nel caso NSB. Le velocità v_1 e v_2 sono le uscite di due subtask indipendenti mentre v_c è l'uscita risultante. L'uscita v_2 del task a priorità inferiore viene proiettata sullo spazio ortogonale al task primario, prima di calcolarne il contributo sul controllo totale.*

In questo modo si garantisce il successo del *task primario*, mentre i task di priorità inferiore vengono soddisfatti solo nel sottospazio nel quale non risultano in conflitto con quello che ha la priorità più alta.

Diamo ora una trattazione matematica dell'algoritmo NSB. Definiamo la posizione cartesiana dell'i-esimo veicolo come un vettore 2-dimesionale

$$\mathbf{p_i} = \begin{bmatrix} x_i & y_i \end{bmatrix}^{\mathbf{T}} \tag{1}$$

e la sua velocità lineare

$$\mathbf{v_i} = \begin{bmatrix} \dot{x}_i & \dot{y}_i \end{bmatrix}^{\mathbf{T}}. \tag{2}$$

Considerato un plotone di N veicoli è utile mantenere una notazione compatta definendo il vettore $\mathbf{p} \in \mathbb{R}^{2N}$ come

$$\mathbf{p} = \begin{bmatrix} \mathbf{p_1^T} & \dots & \mathbf{p_N^T} \end{bmatrix}^{\mathbf{T}} \tag{3}$$

con le relative velocità $\mathbf{v} \in \mathbb{R}^{2N}$

$$\mathbf{v} = \left[\begin{array}{ccc} \mathbf{v_1^T} & \dots & \mathbf{v_N^T} \end{array} \right]^{\mathbf{T}}. \tag{4}$$

Introduciamo ora $\sigma \in \mathbb{R}^m$ definita come *variabile di task*, funzione di $\mathbf{p}$. Come vedremo in dettaglio più avanti, ogni task viene implementato indipendentemente: il problema del raggiungimento di un particolare obiettivo non è altro che un problema di controllo della variabile m-dimensionale σ

$$\sigma = \mathbf{f}(\mathbf{p}). \tag{5}$$

dalla quale si ricava la corrispondente relazione differenziale

$$\dot{\sigma} = \frac{\partial \mathbf{f}(\mathbf{p})}{\partial \mathbf{p}} \mathbf{v} = \mathbf{J}(\mathbf{p})\mathbf{v}, \tag{6}$$

con $\mathbf{J} \in \mathbb{R}^{\mathbf{m} \times \mathbf{nN}}$ la matrice Jacobiana del task. Notiamo che n dipende dal particolare tipo di hardware utilizzato, e nel nostro caso di robot mobile differenziale vincolato al piano, abbiamo $n = 2$.

Conoscendo la dinamica desiderata della variabile di *task* $\sigma_\mathbf{d}(\mathbf{t})$, un metodo pratico per calcolare le traiettorie di riferimento $\mathbf{p_d}(\mathbf{t})$ che devono seguire i veicoli è quello di agire a livello differenziale invertendo la relazione (6). Essendo generalmente $2N \gg m$ (la matrice J è spesso non quadrata) il problema presenta infinite soluzioni, e risulta quindi ragionevole scegliere quella a norma minima:

$$\mathbf{v}_d = \mathbf{J}^\dagger \dot{\sigma}_d = \mathbf{J}^T \left(\mathbf{J}\mathbf{J}^T \right)^{-1} \dot{\sigma}_d \tag{7}$$

A questo punto per sfruttare in modo completo le informazioni disponibili, i controllori del moto dei veicoli necessitano dei riferimenti di posizione $\mathbf{p_d}(\mathbf{t})$ oltre che a quelli di velocità $\mathbf{v_d}(\mathbf{t})$; questi possono essere ottenuti banalmente integrando rispetto al tempo la $\mathbf{v_d}(\mathbf{t})$.

L'integrazione tempo-discreta delle velocità di riferimento dei veicoli potrebbe portare a derive numeriche che comprometterebbe le prestazioni del sistema. Tale deriva viene quindi compensata dall'utilizzo in catena chiusa dell'algoritmo conosciuto, nell' ambito dei manipolatori industriali, come Closed Loop Inverse Kinematics (CLIK). Più precisamente si definisce l'errore $\tilde{\sigma} = \sigma_d - \sigma$, e si corregge la (7) nel modo seguente

$$\mathbf{v}_d(t_k) = \mathbf{J}^\dagger \left(\dot{\sigma}_d(t_k) + \mathbf{\Lambda} \tilde{\sigma}(t_k) \right) \tag{8}$$

$$\mathbf{p}_d(t_k) = \mathbf{p}_d(t_{k-1}) + \mathbf{v}_d(t_k)\Delta t \tag{9}$$

dove t_k è il k-esimo istante di campionamento, Δt il periodo di campionamento, e Λ una matrice di guadagno, semi-definita positiva, sulla quale si potrà agire in fase di tuning del controllore.

Ogni singolo task produce ad ogni istante un'uscita del tipo (8); queste vengono classificate ed ordinate dal supervisore che ne imposta le relative priorità in base alle informazioni disponibili. Denotando con il pedice i, l'i-esimo task e ipotizzando che tale pedice indichi anche il grado di priorità ($i = 1 \Rightarrow$ *priorità massima*),

$$\mathbf{v}_i = \mathbf{J}_i^\dagger \left(\dot{\sigma}_{i,d} + \mathbf{\Lambda}_i \tilde{\sigma}_i \right) \tag{10}$$

l'algoritmo NSB compone la velocità globale di controllo proiettando le uscite dei task secondari sullo *spazio nullo* dei task di priorità superiore, e in un secondo momento calcola la somma, come si osserva nello schema in Fig. 12.

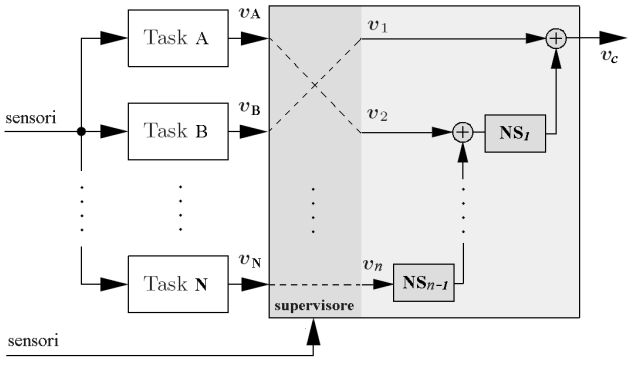

Figura 12: *Schema semplificato del controllore NSB per un sistema a n task. Il supervisore dopo aver impostato l'ordine di priorità dei controlli, procede al calcolo delle componenti delle velocità secondarie solidali allo spazio nullo delle velocità primarie, e in seguito elabora la velocità totale di controllo* v_c.

In un sistema composto da n *task*, denotando con $v^{(i)}$ la velocità di controllo comprendente i task da n a i, si dà la versione ricorsiva dell'algoritmo

$$\mathbf{v}^{(i)} = \mathbf{v}_i + \left(\mathbf{I} - \mathbf{J}_i^\dagger \mathbf{J}_i \right) \mathbf{v}^{(i+1)}, \qquad i = n, \dots, 1 \tag{11}$$

con $v^{(n+1)} = 0$.

Il fattore $\left(\mathbf{I} - \mathbf{J}_i^{\dagger}\mathbf{J}_i\right)$ è proprio lo spazio nullo del task i-esimo sul quale andranno proiettate le velocità inferiori (*NS* in Fig. 12).

Un esempio di velocità di controllo per un sistema di 3 task è il seguente:

$$\mathbf{v}_d = \mathbf{v}_1 + \left(\mathbf{I} - \mathbf{J}_1^{\dagger}\mathbf{J}_1\right)\left[\mathbf{v}_2 + \left(\mathbf{I} - \mathbf{J}_2^{\dagger}\mathbf{J}_2\right)\mathbf{v}_3\right] \quad (12)$$

Modellizzazione cinematica dell'uniciclo

La scelta del modello riveste un'importanza fondamentale, e influisce in modo determinante sulla progettazione del sistema di controllo; naturalmente tale scelta dipende sia dallo specifico sistema in esame, sia dal tipo di controllo che si vuole ottenere. Nel nostro caso, in considerazione della tipologia di robot che utilizzeremo, abbiamo scelto il modello cinematico dell'uniciclo, che governa il comportamento delle variabili di maggior interesse per i nostri obiettivi, ovvero posizione e velocità angolare e lineare dei robot.

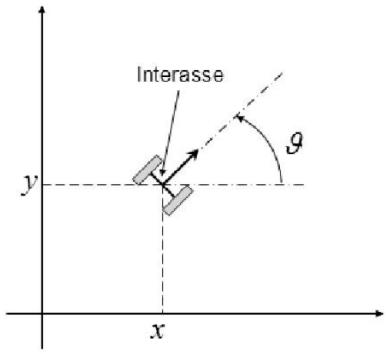

Figura 13: *Schematizzazione dell'uniclo: le variabili di principale interesse sono la posizione del baricentro del robot* (x, y) *e la posizione angolare* θ.

Tale tipologia di WMR è soggetta al vincolo anolonomo che impedisce il movimento in direzione parallela all'interasse, ed è descritta dall'equazione:

$$\dot{x}\sin(\theta) - \dot{y}\sin(\theta) = 0 \quad (13)$$

che, espressa in forma Pfaffiana, diventa:

$$A(q)\dot{q} = \begin{bmatrix} \sin(\theta) & \cos(\theta) & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0 \quad (14)$$

Possiamo quindi riscrivere il modello considerando gli ingressi di controllo v e ω, rispettivamente velocità lineare e angolare del robot:

$$\dot{q} = \begin{bmatrix} \sin(\theta) & 0 \\ \cos(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (15)$$

WMR: controllo individuale via feedback linearization

In letteratura si trovano svariate strategie di controllo per WMR, e la loro validità dipende fortemente dal tipo di *task* che si vuole eseguire; per operazioni di un certo grado di complessità, come il movimento coordinato di una squadra di robot, sono necessari controlli robusti ed accurati; per questo utilizziamo un algoritmo di *tracking* basato sulla linearizzazione in retroazione, che garantisce buone prestazioni al costo di una puntuale conoscenza dello stato del sistema. Bisogna infatti che siano note, ad ogni istante, posizione, velocità e accelerazione sia del riferimento sia del robot da controllare. Tale soluzione sarebbe quindi da scartare a priori se le misure di posizioni e velocità dei veicoli fossero affette da considerevole rumore o non fossero totalmente attendibili.

Siamo dunque interessati all'inseguimento di una traiettoria di riferimento con una certa legge oraria; il che equivale ad annullare l'errore:

$$e(t) = \begin{bmatrix} e_x(t) \\ e_y(t) \end{bmatrix} = \begin{bmatrix} x(t) - x_r(t) \\ y(t) - y_r(t) \end{bmatrix} \quad (16)$$

A tal proposito scegliamo dunque come uscita del sistema MIMO

$$y(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} q(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} \quad (17)$$

Derivando una sola volta le due componenti compare l'ingresso di controllo v in forma esplicita:

$$\begin{cases} z_1(t) = x(t) \\ \dot{z_1}(t) = v(t)\cos(\theta(t)) \end{cases} \qquad \begin{cases} z_2(t) = x(t) \\ \dot{z_2}(t) = v(t)\sin(\theta(t)) \end{cases}$$

Il grado relativo tuttavia non è definito, in quanto, non comparendo ω, la matrice $E(q)$ risulta singolare, e dunque non invertibile:

$$\nu = E(q)u = \begin{bmatrix} \dot{z_1} \\ \dot{z_2} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (18)$$

e quindi non è possibile ottenere una *linearizzazione esatta* (input-stati), ma si deve ricorrere alla *linearizzazione dinamica* del sistema: si considera il nuovo stato $[x\ y\ \theta\ \mu]^T$, dove $\mu = \dot{v}$ è il nuovo ingresso di controllo insieme ad ω, ottenendo

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\mu} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \mu + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \omega \quad (19)$$

A questo punto è possibile derivare ulteriormente le funzioni d'uscita, ricavando

$$\begin{cases} \xi_1 = x \\ \dot{\xi_1} = v\cos\theta \\ \ddot{\xi_1} = \mu\cos\theta - v\omega\sin\theta \end{cases} \quad \begin{cases} \xi_3 = y \\ \dot{\xi_3} = v\sin\theta \\ \ddot{\xi_3} = \mu\sin\theta + v\omega\cos\theta \end{cases}$$

da cui

$$\nu = \begin{bmatrix} \ddot{\xi_1} \\ \ddot{\xi_3} \end{bmatrix} = \begin{bmatrix} \cos\theta & -v\sin\theta \\ \sin\theta & v\cos\theta \end{bmatrix} \begin{bmatrix} \mu \\ \omega \end{bmatrix} \quad (20)$$

Ora la matrice $E(q)$ non è più singolare, e si ha un grado relativo $r = 4 = n$, quindi si può procedere con la linearizzazione input-stati del nuovo sistema (19), con ingressi $u = [\mu\ \omega]^T$: supponendo $v \neq 0$ (il che comporterà certe attenzioni nel controllo effettivo del robot), si può operare il seguente cambio di coordinate

$$\xi = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_4 \end{bmatrix} = \begin{bmatrix} x \\ v\cos\theta \\ y \\ v\sin\theta \end{bmatrix} \quad (21)$$

Essendo $E(q)$ invertibile, la retroazione linearizzante risulta

$$u = E(q)^{-1}\nu = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta/v & \cos\theta/v \end{bmatrix} \nu \quad (22)$$

Per controllare l'errore (16) a zero utilizziamo un PD con parametri K_{p1}, K_{p2}, K_{d1}, K_{d2}

$$\begin{cases} u_1 = \ddot{x_r} + K_{p1}(x_r - x) + K_{d1}(\dot{x_r} - \dot{x}) \\ u_2 = \ddot{y_r} + K_{p2}(y_r - y) + K_{d1}(\dot{y_r} - \dot{y}) \end{cases} \quad (23)$$

Introducendo il nuovo stato $\xi = v$ nel controllore si ottiene la legge di controllo

$$\begin{cases} \dot{\xi} = u_1\cos\theta + u_2\sin\theta \\ v = \xi \\ \omega = (u_2\cos\theta - u_1\sin\theta)/\xi \end{cases} \quad (24)$$

che verrà utilizzata per stabilizzare ogni veicolo sulla propria traiettoria desiderata.

Risultati in simulazione

Per verificare le prestazioni del controllo via *feedback linearization* abbiamo condotto delle prove di *tracking* su un riferimento con andamento sinusoidale, ottenendo risultati soddisfacenti; è comunque da tenere presente che, in sede di simulazione, il calcolo di posizione, velocità ed accelerazione del riferimento può essere fatto anche fuori linea, con la massima precisione, essendo la traiettoria nota a priori e priva di incertezze.

Il modello discreto del sistema è stato ottenuto integrando la (15): utilizzando l'approssimazione Runge-Kutta del 2° ordine si perviene alle equazioni di aggiornamento:

$$\begin{aligned} x(k+1) &= x(k) + v(k)T_c\cos\left(\theta(k) + \omega(k)T_c/2\right) \\ y(k+1) &= y(k) + v(k)T_c\sin\left(\theta(k) + \omega(k)T_c/2\right) \\ \theta(k+1) &= \theta(k) + \omega(k)T_c \end{aligned}$$

che sono state utilizzate per tutte le simulazioni; in Fig. 14 è mostrata la simulazione dell'inseguimento di un riferimento con componenti di velocità x e y sinusoidali da parte di un uniciclo con condizioni iniziali $[x\ y\ \theta]^T = [0.5\ 0.5\ \pi]^T$; aumentando il valore dei parametri di controllo l'errore di inseguimento si assesta più rapidamente a zero, al costo però di valori più alti per le velocità di controllo. Questo aspetto è particolarmente critico, soprattutto in vista dell'applicazione pratica, dove sarà necessario valutare gli effetti della saturazione degli attuatori.

Controllo coordinato di una squadra di robot: l'approccio NSB

Gli strumenti fino ad ora presentati serviranno come mezzo per applicare algoritmi più complessi di movimento coordinato, ottenuti tramite controllo Null

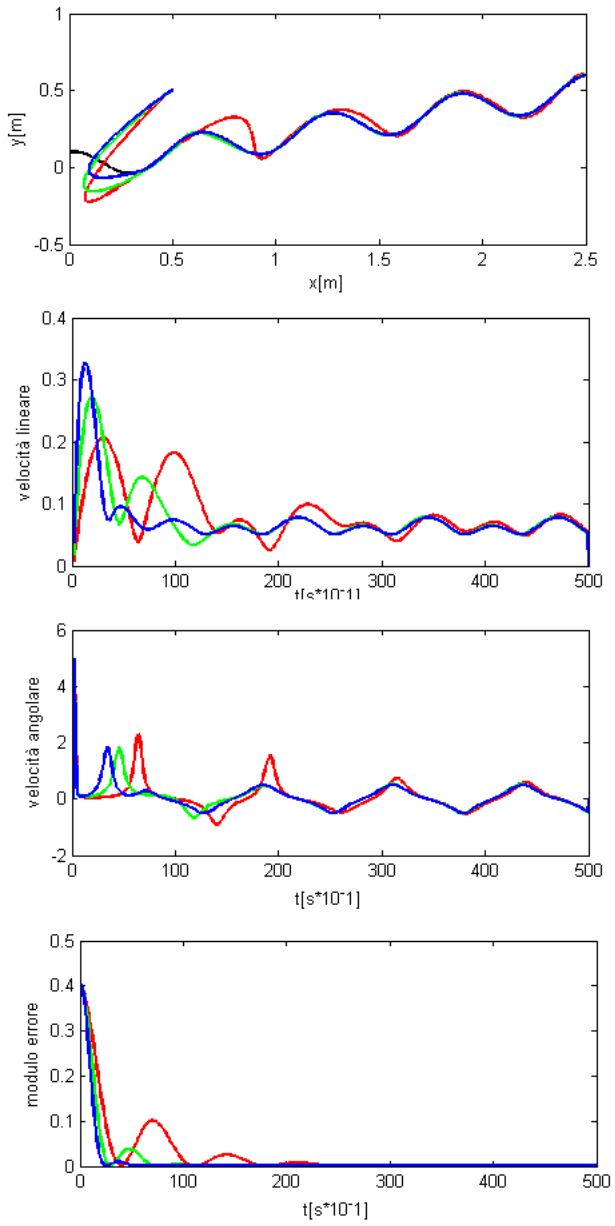

Figura 14: *Dall'alto in basso: grafici delle traiettorie (in nero quella di riferimento), delle velocità lineari (in [m/s]), delle velocità angolari (in [rad/s]) e degli errori di inseguimento (in [m]) per diversi valori delle costanti di controllo ($K_{p1} = K_{p2} = K_{d1} = K_{d2} = 0.2$ in rosso, $K_{p1} = K_{p2} = K_{d1} = K_{d2} = 0.5$ in verde, $K_{p1} = K_{p2} = K_{d1} = K_{d2} = 1$ in blu).*

Space Behavior. L'idea che sta alla base del controllo NSB è la creazione di singoli *task* indipendenti con relative leggi di controllo regolati da una struttura gerarchica che ne regola le priorità, pur garantendo un comportamento globale che soddisfi ogni singolo obiettivo; definiamo di seguito le leggi di controllo relative alle principali manovre che interessano il movimento coordinato di una squadra di veicoli.

Obstacle avoidance

Indubbiamente evitare le collisioni è l'obiettivo primario per qualunque missione di squadra; se così non fosse si metterebbe a rischio come minimo l'integrità dei veicoli. Data l'importanza del *task*, vogliamo analizzarne a fondo i principali aspetti, perlomeno relativamente al controllo NSB. Consideriamo dapprima il semplice caso di un singolo veicolo e di un ostacolo puntiforme da cui ci si vuole mantenere a distanza d. Definiamo allo scopo la variabile di *task*

$$\sigma_{obs} = \mathbf{f}(\mathbf{p}) = ||\mathbf{p} - \mathbf{p}_{obs}|| \tag{25}$$

dove $\mathbf{p_{obs}}$ è il vettore di posizione dell'ostacolo; la la variabile di *task* desiderata è dunque $\sigma_{obs,d} = d$. Dalla (6) si ricava la relazione differenziale:

$$\dot{\sigma} = \mathbf{J}_{obs}(\mathbf{p})\mathbf{v_{obs}} = \left[\frac{\mathbf{p} - \mathbf{p_{obs}}}{||\mathbf{p} - \mathbf{p_{obs}}||}\right]^{\mathbf{T}} \mathbf{v_{obs}} \tag{26}$$

In base alla (10) si ottiene infine la velocità di controllo:

$$\mathbf{v_{obs}} = \mathbf{J_{obs}}^{\dagger}\mathbf{\Lambda_{obs}}(\mathbf{d} - ||\mathbf{p} - \mathbf{p_{obs}}||) \tag{27}$$

in cui $\mathbf{\Lambda}_{obs} \in \mathbb{R}^2$ è il fattore di peso del *task*. Si noti come tale legge di controllo in realtà non regoli una distanza minima da mantenere rispetto all'ostacolo, ma imponga al veicolo di mantenere *esattamente* distanza d dall'ostacolo; naturalmente un simile comportamento non consentirebbe di portare a termine alcun tipo di missione.

Per ovviare a questo inconveniente la soluzione più immediata è quella di utilizzare una variabile decisionale che attivi e disattivi il controllo per evitare gli ostacoli: come primo approccio adottiamo il semplice algoritmo:

- *Abilitare il controllo*, quando la distanza tra veicolo e ostacolo è minore di un certo valore di soglia d.
- *Disabilitare il controllo*, quando la direzione del veicolo (calcolata in base alle velocità fornite dai rimanenti *task*) è uscente dal cerchio di raggio d centrato nell'ostacolo.

Proponiamo di seguito un metodo decisionale per l'abilitazione del controllo per le collisioni: si consideri la velocità di controllo $\mathbf{v}_c$ calcolata in base alla (11) escludendo il *task* primario di *obstacle avoidance*, e lo spazio nullo $\mathbf{N}_o = \left(\mathbf{I} - \mathbf{J}_o^{\dagger}\mathbf{J}_o\right)$; si ha che la funzione scalare s.d.p.

$$g(t) = \frac{\mathbf{v}_c^T(t)\mathbf{N}_o\mathbf{v}_c(t)}{\mathbf{v}_c^T(t)\mathbf{v}_c(t)} \tag{28}$$

ha un massimo unitario nell'istante in cui $\mathbf{v}_c$ e la velocità di controllo del *task* primario $\mathbf{v}_o$ hanno la stessa direzione (punto A di Fig. 15). Si può quindi abilitare il controllo quando sono verificate simultaneamente le seguenti condizioni:

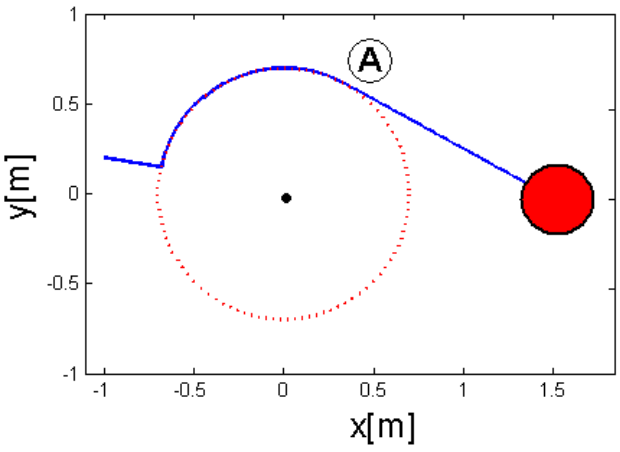

Figura 15: *Traiettoria simulata per l'*obstacle avoidance*: partendo dal punto* $(-1, 0.2)$ *il veicolo raggiunge il punto* $(1.5, 0)$ - task *secondario - evitando l'ostacolo posto in* $(0, 0)$ - task *primario - con raggio di* $0.7m$. *Nel punto* A *la funzione* $g(t)$ *raggiunge il suo massimo in* 1.

- La distanza tra veicolo e ostacolo è minore di un certo valore di soglia d.
- $\dot{g}(t) > 0$.

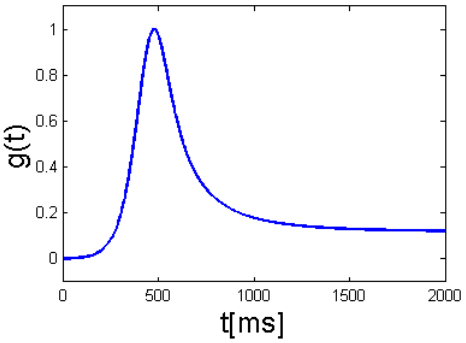

Figura 16: *Andamento nel tempo della funzione* $g(t)$.

L'andamento temporale di $g(t)$ è mostrato in Fig. 16. Una doverosa osservazione in vista di applicazioni pratiche è che, per tempi di campionamento lunghi (maggiori del decimo di secondo), le velocità subiscono variazioni di direzione consistenti: si verifica infatti, a causa della prima condizione, un rapido *switching* ON/OFF del controllo *obstacle avoidance*. Se ciò avviene in tempi sufficientemente rapidi, si ha un cambiamento quasi continuo di direzione, senza importanti conseguenze sull'inseguimento; al contrario possono insorgere indesiderate oscillazioni sulle velocità di controllo, come mostrato in Fig. 17 e Fig. 18. A ciò si aggiunga che tali oscillazioni riguardano il riferimento ideale; eseguire un tracking su traiettorie così discontinue, soprattutto con saturazioni sulle velocità angolari, può diventare veramente problematico. Per risolvere il problema, laddove non sia possibile diminuire il tempo di controllo, proponiamo la seguente soluzione: si consideri una distanza di soglia $d' > d$ entro la quale attivare il controllo *obstacle avoidance*; a questo punto si introduce un supervisore che attivi il controllo per le collisioni se almeno una delle seguenti condizioni è soddisfatta:

- La distanza dall'ostacolo è minore di d
- La distanza dall'ostacolo è minore di d' e $\dot{g}(t) > 0$.

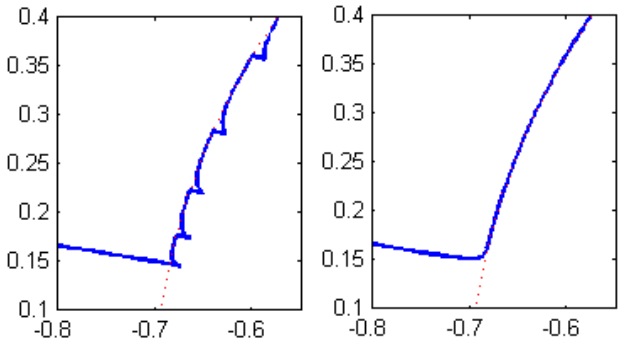

Figura 17: *Particolare delle traiettorie di riferimento per i due algoritmi presentati (tempo di controllo $T_c = 0.1s$): si noti l'andamento discontinuo nel primo caso.*

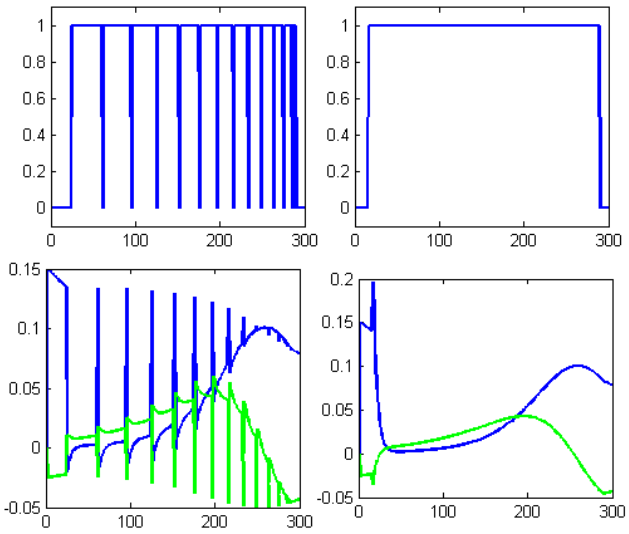

Figura 18: In alto*: grafico dei primi campioni temporali della variabile decisionale per l'attivazione del controllo* obstacle avoidance *nei due algoritmi; nel secondo caso il controllo viene attivato un'unica volta.* In basso*: andamento delle velocità di controllo $\dot{x}$ (in* blu*) e $\dot{y}$ (in* verde*); le evidenti oscillazioni sono eliminate con il nuovo algoritmo proposto, rimane solamente il transitorio iniziale di attivazione dell'*obstacle avoidance.

Si confronti la differenza in Fig.18: utilizzando l'algoritmo proposto il controllo viene attivato un'unica volta per l'intero *task*; diversamente si generano pericolose oscillazioni sulle velocità di controllo.

Finora abbiamo considerato il semplice caso di un veicolo e un ostacolo fisso; ma come si può affrontare il caso in cui più veicoli debbano evitare collisioni, oltre che con ostacoli fissi, anche tra di loro? Supponiamo dapprima che gli unici ostacoli siano i veicoli stessi, e ci chiediamo come gestire il controllo simultaneo della squadra per evitare collisioni. Come primo approccio presentiamo la soluzione più semplice, che si rivela un buon compromesso tra prestazioni e rapidità di calcolo: ad ogni ciclo di controllo viene calcolato per ogni veicolo l'ostacolo più vicino, e viene applicato l'algoritmo di controllo delle collisioni per quel solo ostacolo. In questo modo tutte le distanze minime sono garantite, ma non sempre viene scelta la traiettoria migliore in funzione dei *task* secondari.

Un altro metodo di affrontare il problema (utilizzando una soglia d' sufficientemente grande rispetto a d) è quello di considerare tutti gli ostacoli, con un peso diverso a seconda, ad esempio, della distanza.

L'estensione al caso di ostacoli fissi è pressoché immediata: basterà introdurre un vettore con le posizioni $\mathbf{p}_k$ di ognuno di essi, e valutarlo insieme al vettore delle posizioni dei veicoli nel calcolo dell'ostacolo più vicino.

Nel caso di ostacoli non puntiformi (schematizzabili, in generale, come poligoni convessi), gli algoritmi presentati restano validi: è sufficiente applicare il controllo al punto più vicino del poligono rispetto al veicolo. Questo adattamento può tuttavia far insorgere problematiche computazionali, in quanto il calcolo di tale minimo, per un numero elevato di veicoli, può diventare oneroso in un controllo centralizzato. In questa sede rimandiamo ulteriori, seppur interessanti, approfondimenti sull'argomento.

Convergenza ad un goal

Il più semplice *task* di movimento coordinato è il raggiungimento di un punto prestabilito da parte di ogni robot della squadra; a tal proposito consideriamo un team di N veicoli, la cui posizione è data dal vettore

$\mathbf{p} \in \mathbb{R}^{2N}$:

$$\mathbf{p(t)} = \begin{bmatrix} x_1(t) \\ y_1(t) \\ x_2(t) \\ y_2(t) \\ \vdots \\ x_N(t) \\ y_N(t) \end{bmatrix} \tag{29}$$

e prendiamo come variabile di *task* la posizione stessa ($\mathbf{f(p)} = \mathbf{p}$):

$$\sigma_{goal} = \mathbf{p} \tag{30}$$

La matrice jacobiana associata a σ è

$$J = I \in \mathbb{R}^{2N \times 2N} \tag{31}$$

Il valore desiderato è in questo caso $\sigma_{goal,d} = \mathbf{p}_{goal}$, dove $\mathbf{p}_{goal}$ è il vettore delle coordinate del punto da raggiungere, dunque la velocità di controllo risulta, secondo la(10):

$$\mathbf{v}_{goal} = \mathbf{\Lambda}_{goal}(\mathbf{p}_{goal} - \mathbf{p}) \tag{32}$$

in cui $\mathbf{\Lambda}_{goal} \in \mathbb{R}^{2N \times 2N}$ è la matrice diagonale di peso del *task*.

Proponiamo di seguito una significativa simulazione in cui viene svolto come *task* primario l'*obstacle avoidance* e come *task* secondario il raggiungimento di un punto prestabilito. Abbiamo considerato una squadra di $N = 6$ robot, scegliendo arbitrariamente come condizioni iniziali

$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ -0.6 \\ 0.8 \\ 0.5 \\ -0.3 \\ 0.7 \end{bmatrix} \mathbf{y}_0 = \begin{bmatrix} -0.6 \\ -0.2 \\ -0.4 \\ 0.6 \\ -0.7 \\ 0.2 \end{bmatrix} \theta_0 = \begin{bmatrix} \pi/2 \\ \pi \\ 0 \\ 3\pi/2 \\ -\pi/2 \\ -\pi/6 \end{bmatrix}$$

e come posizione goal

$$\mathbf{p} = [0 \ -0.75 \ 0 \ -0.45 \ 0 \ -0.15 \ 0 \ 0.15 \ 0 \ 0.45 \ 0 \ 0.75]$$

Vogliamo dunque disporre i veicoli verticalmente sul piano (x, y) a $30cm$ l'uno dall'altro, mantenendoli durante lo spostamento ad una distanza non inferiore a $d = 20cm$. Per evitare le collisioni abbiamo implementato l'algoritmo presentato nella sezione precedente, ottenendo per ogni singolo robot due tipologie

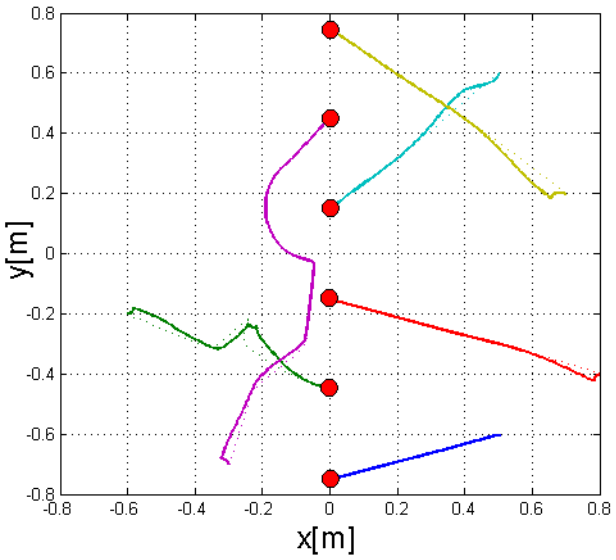

Figura 19: *Grafico* (x, y) *delle traiettorie dei riferimenti (linee tratteggiate) e dei veicoli simulati (linee continue) per il controllo NSB obstacle avoidance e raggiungimento del punto di goal.*

di velocità di controllo: nel caso non ci siano ostacoli (i.e. altri veicoli) lungo la traiettoria da seguire, la velocità di controllo è $\mathbf{v}_1 = \mathbf{v}_{goal}$; nel caso invece in cui $\mathbf{v}_1$ porterebbe due veicoli a distanza minore di d, si deve proiettare la componente di velocità del *task* secondario sullo spazio nullo del *task* primario, ottenendo, secondo la (11):

$$\mathbf{v}_2 = \mathbf{v}_{obs} + \left(\mathbf{I} - \mathbf{J}_{obs}^{\dagger}\mathbf{J}_{obs}\right)\mathbf{v}_{goal} \tag{33}$$

dove $\mathbf{v}_{obs}$ e $\mathbf{v}_{goal}$ sono date rispettivamente da (27) e (32).

Per $\mathbf{\Lambda}_{obs} = 0.8I^{2\times2}$ e $\mathbf{\Lambda}_{goal} = 0.5I^{12\times12}$, e utilizzando costanti unitarie per il controllo via *feedback linearization*, abbiamo ottenuto le traiettorie di Fig. 19. Le velocità di controllo sono state saturate, imponendo il limite di $0.1m/s$ per le velocità lineari e $2rad/s$ per le velocità angolari.

Convergenza del baricentro con varianza unidimensionale

Vogliamo ora che siano che siano portate a termine, con la priorità indicata, le seguenti operazioni:

- Evitare collisioni.
- Portare il baricentro della formazione in una posizione desiderata $\sigma_{bar,d} \in \mathbb{R}^2$.
- Distribuire i veicoli attorno al baricentro con varianza desiderata $\sigma_{var,d} \in \mathbb{R}$.

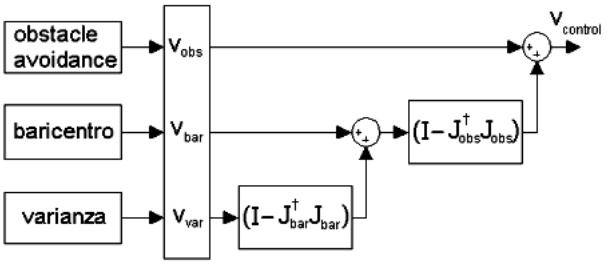

Figura 20: *Schema a blocchi dell'algoritmo di controllo NSB per spostamento del baricentro con varianza unidimensionale*

In questo caso sono dunque necessari 3 *task*, e si noti in Fig. 20 come ognuno di essi sia proiettato sullo spazio nullo del *task* con priorità superiore. Presentiamo di seguito gli algoritmi di calcolo delle velocità per la convergenza al baricentro e per la distribuzione dei robot con varianza assegnata.

Con baricentro della formazione si intende la media delle posizioni dei veicoli

$$\sigma_{bar} = \frac{1}{N}\sum_{k=1}^{N} \mathbf{p}_k, \qquad N > 1 \tag{34}$$

dove $\mathbf{p}_k$ esprime la posizione (x, y) del veicolo k-esimo. La jacobiana relativa a σ_{bar} è

$$\mathbf{J}_{bar} = \frac{1}{N}\begin{bmatrix} 1 & 0 & \dots & 1 & 0 \\ 0 & 1 & \dots & 0 & 1 \end{bmatrix} \in \mathbb{R}^{2\times 2N} \tag{35}$$

con pseudoinversa

$$\mathbf{J}^{\dagger}_{bar} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{2N\times 2} \tag{36}$$

e questo porta a

$$\mathbf{v}_{bar} = \mathbf{J}^{\dagger}_{bar}\left(\mathbf{v}_{bar,d} + \mathbf{\Lambda}_{bar}\left(\sigma_{bar,d} - \sigma_{bar}\right)\right) \tag{37}$$

dove $\mathbf{v}_{bar,d}$ è la velocità del baricentro desiderato (non necessariamente nota a priori), e $\mathbf{\Lambda}_{bar}$ è la matrice 2×2 di peso del *task*.

La varianza unidimensionale delle posizioni dei veicoli rispetto al baricentro invece è definita come:

$$\sigma_{var} = \frac{1}{N}\sum_{k=1}^{N} (\mathbf{p}_k - \mathbf{p}_{bar})^T(\mathbf{p}_k - \mathbf{p}_{bar}) \tag{38}$$

Supponendo $N > 1$ è possibile calcolare la matrice jacobiana $\mathbf{J}_{var} \in \mathbb{R}^{1\times 2N}$

$$\mathbf{J}_{var} = \frac{2(N-1)}{N^2}\begin{bmatrix} [\mathbf{p}_1 - \mathbf{p}_{bar}]^T & \dots & [\mathbf{p}_N - \mathbf{p}_{bar}]^T \end{bmatrix} \tag{39}$$

la cui pseudoinversa risulta

$$\mathbf{J}^{\dagger}_{var} = \frac{N^2}{2(N-1)}\begin{bmatrix} \frac{\mathbf{p}_1 - \mathbf{p}_{bar}}{[\mathbf{p}_1 - \mathbf{p}_{bar}]^T[\mathbf{p}_1 - \mathbf{p}_{bar}]} \\ \vdots \\ \frac{\mathbf{p}_N - \mathbf{p}_{bar}}{[\mathbf{p}_N - \mathbf{p}_{bar}]^T[\mathbf{p}_N - \mathbf{p}_{bar}]} \end{bmatrix} \tag{40}$$

Invertendo la (6) otteniamo infine la velocità di controllo

$$\mathbf{v}_{var} = \mathbf{J}^{\dagger}_{var}\mathbf{\Lambda}_{var}\left(\sigma_{var,d} - \sigma_{var}\right) \tag{41}$$

dove $\sigma_{var,d} \in \mathbb{R}$ è la varianza desiderata. Unendo i tre *task* si ottiene la velocità di controllo

$$\mathbf{v}_c = \mathbf{v}_{obs} + \left(\mathbf{I} - \mathbf{J}^{\dagger}_{obs}\mathbf{J}_{obs}\right)\left[\mathbf{v}_{bar} + \left(\mathbf{I} - \mathbf{J}^{\dagger}_{bar}\mathbf{J}_{bar}\right)\mathbf{v}_{var}\right]$$

Naturalmente nel caso in cui non vi sia pericolo di collisione e il controllo per l'*obstacle avoidance* sia disattivato si ha

$$\mathbf{v}_c = \mathbf{v}_{bar} + \left(\mathbf{I} - \mathbf{J}^{\dagger}_{bar}\mathbf{J}_{bar}\right)\mathbf{v}_{var} \tag{42}$$

La simulazione delle traiettorie per una squadra di $N = 6$ robot che convergono al baricentro $\mathbf{p}_{bar,d} = [1\ 1]^T$ con varianza desiderata di $0.2m^2$ mantenendosi ad una distanza di $20cm$ è mostrata in Fig. 21.

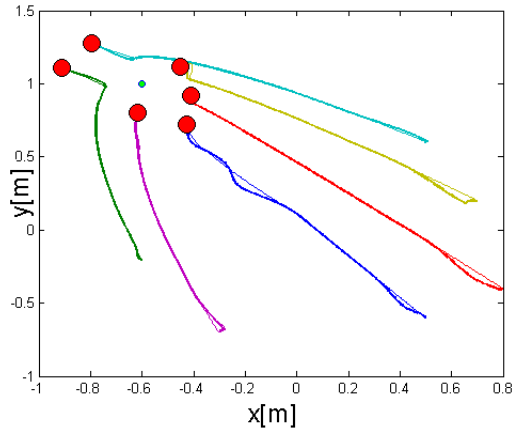

Figura 21: *Grafico* (x, y) *delle traiettorie dei riferimenti e dei veicoli simulati (linee più spesse) per il controllo NSB obstacle avoidance e raggiungimento del baricentro desiderato con varianza* $0.1m^2$.

Movimento in formazione rigida

Anche per questo *task* dobbiamo soddisfare in struttura gerarchica tre condizioni:

- Evitare collisioni.
- Muovere il baricentro della squadra con una determinata legge oraria.
- Portare i veicoli in una determinata formazione rispetto al baricentro (ad esempio, disporli in cerchio).

Lo schema di controllo è del tutto analogo a quello di Fig. 20. Abbiamo già visto come controllare i veicoli per soddisfare le prime due condizioni; analizziamo ora il raggiungimento di una determinata formazione rigida rispetto al baricentro. A tale scopo consideriamo la variabile di *task*

$$\sigma_{form} = \begin{bmatrix} \mathbf{p}_1 - \mathbf{p}_{bar} \\ \vdots \\ \mathbf{p}_N - \mathbf{p}_{bar} \end{bmatrix} \tag{43}$$

che considera per ogni veicolo lo scostamento rispetto all'attuale baricentro. La jacobiana associata è

$$\mathbf{J}_{form} = \begin{bmatrix} \mathbf{A} & 0 \\ 0 & \mathbf{A} \end{bmatrix} \in \mathbb{R}^{2N \times 2N} \tag{44}$$

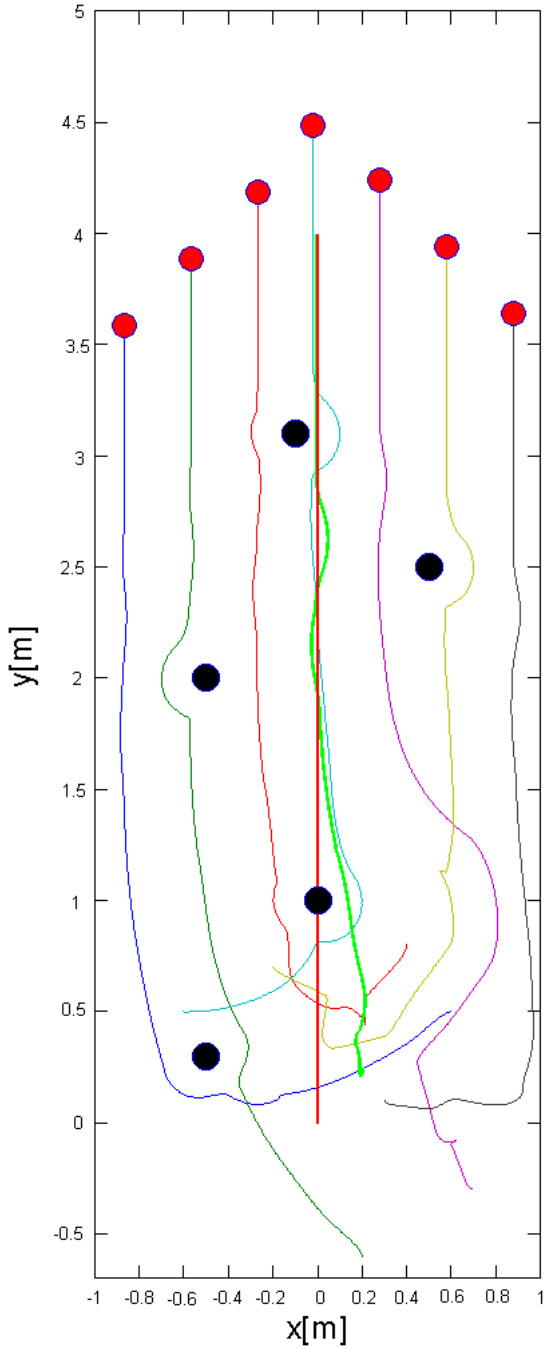

Figura 22: *Grafico* (x, y) *delle traiettorie dei veicoli simulati per il controllo NSB obstacle avoidance e inseguimento del baricentro di riferimeno in formazione.*

diagonale a blocchi, con

$$\mathbf{A} = \begin{bmatrix} 1-\frac{1}{N} & -\frac{1}{N} & \dots & -\frac{1}{N} \\ -\frac{1}{N} & 1-\frac{1}{N} & \dots & -\frac{1}{N} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{1}{N} & -\frac{1}{N} & \dots & 1-\frac{1}{N} \end{bmatrix} \quad (45)$$

Il parametro $\sigma_{form,d} \in \mathbb{R}^{2N}$ rappresenta le posizioni desiderate di ogni veicolo rispetto al baricentro. La velocità di controllo ottenuta è quindi

$$\mathbf{v}_{form} = \mathbf{J}^{\dagger}_{form} \mathbf{\Lambda}_{form} \left(\sigma_{form,d} - \sigma_{form}\right) \quad (46)$$

In questo caso abbiamo supposto la formazione rigida, dunque la componente $\dot{\sigma}_{form,d}$ è nulla; tuttavia grazie alle semplicità con cui è possibile specificare le posizioni relative desiderate rispetto al baricentro

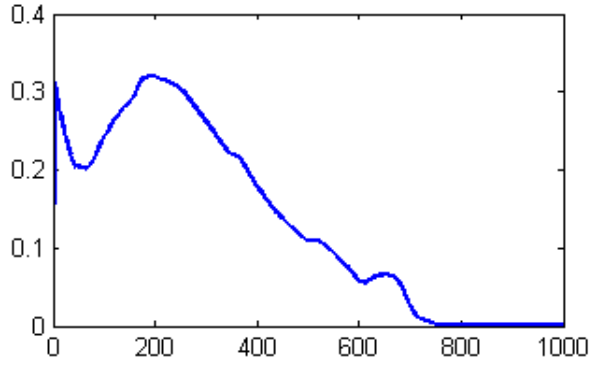

Figura 23: *Andamento nel tempo dell'errore quadratico di inseguimento rispetto al baricentro di riferimento.*

(soprattutto nel caso di formazioni regolari quali circonferenze, linee rette, poligoni, etc.), è possibile cambiare dinamicamente la formazione anche durante il controllo, ad esempio escludendo o aggiungendo veicoli alla formazione o allargando, restringendo o ruotando la formazione stessa. Proponiamo per questa applicazione un controllo NSB basato su un sistema di riferimento in coordinate polari, considerando il nuovo vettore di posizione

$$\hat{\mathbf{p}} = \begin{bmatrix} \rho_1 & \phi_1 & \dots & \rho_N & \phi_N \end{bmatrix}^T \quad (47)$$

In questo modo è possibile operare sui singoli parametri di guadagno della posizione radiale ρ e angolare ϕ. La simulazione proposta riguarda il movimento in formazione di $N = 7$ robot con inseguimento del baricentro di riferimento a velocità costante; i veicoli, oltre ad evitare collisioni tra di loro, sono tenuti ad evitare gli ostacoli presenti sul percorso, e questo comporta dei disturbi nella traiettoria del baricentro, necessari per asservire al *task* primario di *obstacle avoidance*. Abbiamo utilizzato valori unitari sia per $\mathbf{\Lambda}$ di controllo sia per i parametri proporzionali e di velocità nel controllo in *feedback linearization*. In Fig. 23 è mostrato l'andamento nel tempo dell'errore quadratico di inseguimento del baricentro di riferimento: si notino le perturbazioni dovute alla presenza di ostacoli nel percorso.

Risultati sperimentali

Per le verifiche sperimentali abbiamo utilizzato un calcolatore centrale per l' elaborazione di una versione centralizzata dell' algoritmo di controllo, alcuni robot e-Puck dotati di motore a passo e di comunicazione bluetooth con il computer centrale, una pedana di lavoro $3.20m \times 2.40m$ e una telecamera per la localizzazione dei veicoli (per una trattazione più completa dell'apparecchiatura hardware rimandiamo all'appendice A). Durante le prove sperimentali si sono risolti alcuni problemi legati alla coordinazione dei robot E-puck, derivanti principalmente dalle consuete differenze rispetto alla fase di simulazione.
Dato che l'algoritmo utilizzato si basa su un ciclo di controllo fondamentale comprendente l'analisi e il filtraggio dei dati provenienti dalla telecamera, l'elabo-

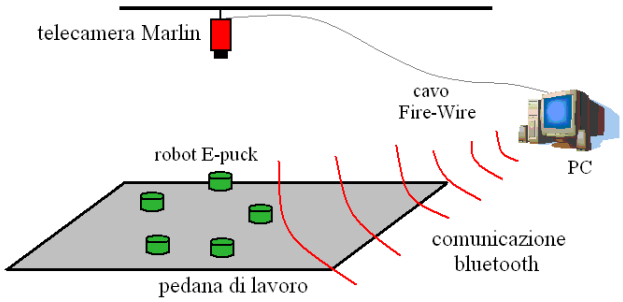

Figura 24: *Illustrazione del sistema di lavoro*

razione e l'attuazione dei comandi, il primo problema consiste nel fatto che il tempo di campionamento risultante tra un ciclo e l'altro è piuttosto elevato ($0.1s$).
Il secondo problema invece riguarda la presenza di riferimenti poco *smooth*: questo è dovuto sia al fatto che sono possibili errori sulla localizzazione che portano ad avere misure rumorose (in media $1cm$ sulla posizione e $1°$ sull'orientamento) sia alla modalità con cui si implementa il *path planning*, cioè il procedimento con cui l'algoritmo *NSB* calcola il cammino geometrico da far seguire (cammino che è olonomo, perciò non tenente conto dei vincoli anolonomi propri dei robot utilizzati).
Questo secondo problema può portare alla presenza di cuspidi (cambi secchi di direzione e velocità) e quindi di errori numerici dovuti alle derivazioni a tempo discreto eseguite in fase di elaborazione dei comandi di *feedback-linearization*. Si nota tuttavia che gli errori e i malfunzionamenti dovuti a tali problematiche risultano minimi grazie all'utilizo di motori a passo.

Sono stati comunque effettuati alcuni accorgimenti: per prima cosa un *tuning* del controllore (*Null-Space-Based* e *feedback-linearization*), e successivamente una saturazione delle velocità di controllo (di $10cm/s$ e $2rad/s$ per lavelocità lineare e angolare rispettivamente); inoltre si sono proposti degli obiettivi da raggiungere dotati di una dinamica non troppo spinta.
In ogni caso per ottenere delle prestazioni migliori si può tentare di ridurre il tempo di campionamento dell'algoritmo di controllo, oppure si può implementare un *path planning* di tipo non-olonomo, in modo che tenga conto dei vincoli che i robot devono soddisfare.

Obstacle avoidance e convergenza con varianza unidimensionale

Nel primo esperimento si richiede che un plotone di 6 robot, a partire da condizioni iniziali casuali, si muova in modo che il proprio baricentro raggiunga una desiderata configurazione costante $\sigma_{bar} = [0.6\ 1.6]^T$ mantenendo una varianza unidimensionale arbitraria $\sigma_{var} = 0.2m^2$. A questi task secondari viene aggiunto il task per evitare gli ostacoli, che nel caso specifico di questa esperienza sono gli stessi robot attivi oltre che al punto di arrivo del baricentro, impostando il valore di sicurezza $d = 20cm$.

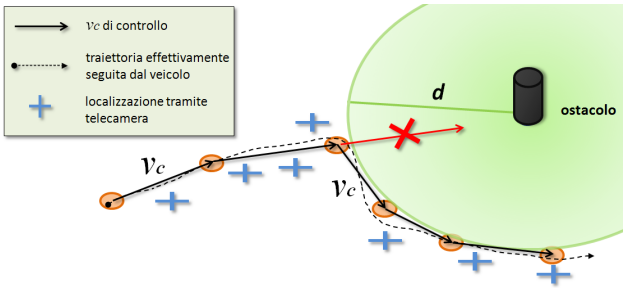

Figura 25: *Differenze tra cammino geometrico dato dalla concatenazione delle v_c, localizzazione della telecamera e percorso effettivamente seguito dal robot, in una comune situazione di Obstacle-Avoidance.*

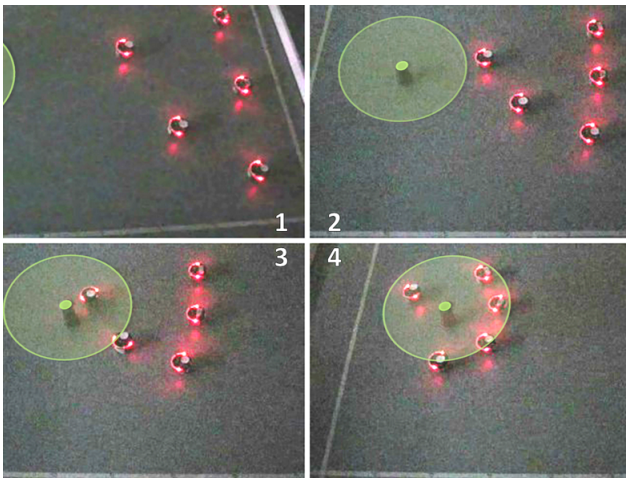

Figura 26: *Obstacle avoidance + convergenza del baricentro + varianza in 4 frames; l'area più chiara è rappresenta indicativamente la varianza.*

Non essendoci task riguardanti il singolo veicolo del plotone, non si ha il raggiungimento di una configurazione di equilibrio, e i robot si fermano quando la distanza tra la posizione del baricentro e la posizione desiderata è più piccola di una determinato valore critico. Questo è un significativo esempio di controllo di tipo comportamentale nel quale si sfruttano i moti interni del sistema, che complessivamente non cambiano la posizione del baricentro (poichè sono nello spazio nullo del task della posizione del baricentro), per soddisfare anche quei task che hanno priorità inferiore.

In Fig. 27 sono riportate le traiettorie (linee colorate) dei veicoli partiti da condizioni iniziali casuali, i riferimenti calcolati mediante l'algoritmo *NSB* (linee nere tratteggiate) e la posizone finale del baricentro (croce rossa) calcolato rispetto alle posizione finale dell'intera squadra (marker rossi). L'esperienza si è svolta con la seguente configurazioni dei parametri di controllo: guadagni dell'algoritmo di *feedback linearization* unitari, $\Lambda_{obs} = 0.8$ e $\Lambda_{bar} = \Lambda_{var} = 0.3$

Come si può osservare in figura il baricentro raggiunge la posizione desiderata e i robot vi si mantengono ad una distanza che in media non supera i $20cm$, il tutto evitando collisioni. E' da notare ad ogni modo che si hanno delle imprecisioni nell'inseguimento da parte dei robot nelle fasi finali del controllo, dovute alle problematiche introdotte nella sezione precedente.

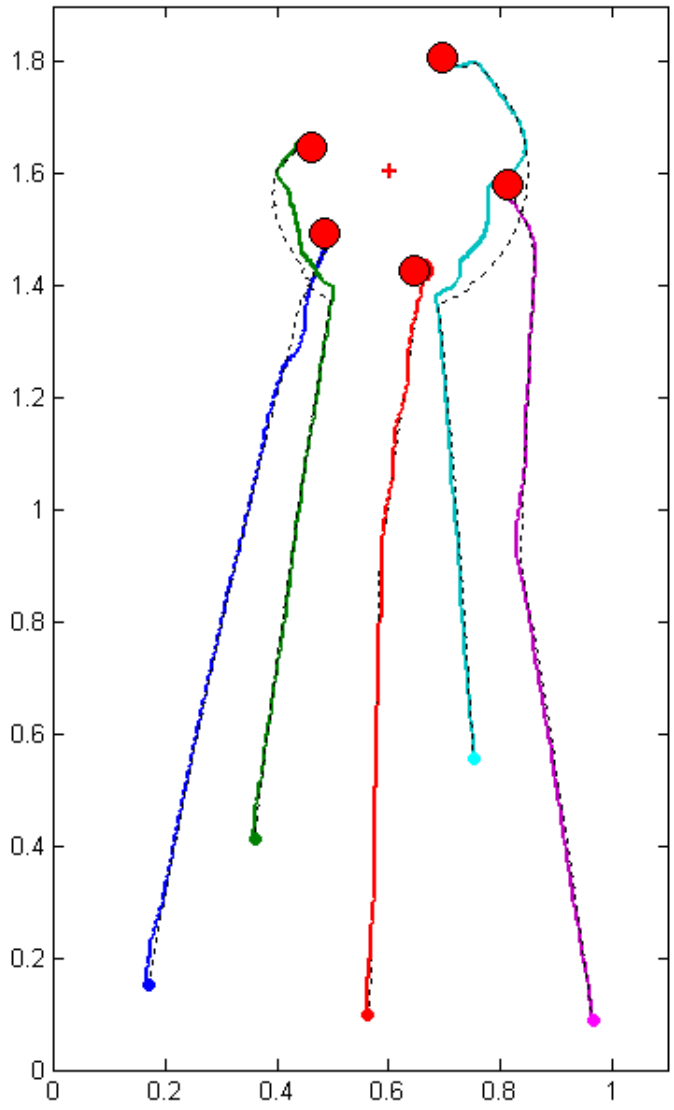

Figura 27: *Traiettorie (linee colorate) dei veicoli, riferimenti (linee nere tratteggiate) e posizone finale del baricentro (croce rossa) calcolato rispetto alle posizione finale dell'intera squadra (marker rossi).*

Obstacle avoidance e movimento in formazione

Come è stato anticipato precedentemente il controllo in formazione rigida di una squadra di robot è un problema particolarmente trattato in letteratura. Per dimostrare la bontà del controllo via *NSB*, ne mostriamo tre implementazioni per tre tipi diversi di formazione composta di cinque veicoli: *cerchio*, *linea* e *freccia*.

Nei casi trattati si richiede il mantenimento di una formazione rigida durante il moto della squadra, il quale viene definito calcolando una opportuna legge oraria per la posizione desiderata del baricentro. Per chiarezza espositiva verrà scelto un moto rettilineo uniforme e un moto circolare a velocità costante. L'estensione al caso più generale di formazioni dinamiche, e movimenti della squadra non uniformi è semplice e immediatamente implementabile, impostando

adeguatamente i valori della velocità della variabile di task $\dot{\sigma}_{for}(t)$ $\dot{\sigma}_{bar}(t)$.

Ovviamante il task per evitare le collisioni è costantemente presente in questa fase sperimentale, e quando attivo risulta essere sempre il task a più alta priorità.

In un primo caso imponiamo un moto al baricentro in modo che segua una traiettoria circolare attorno al punto $[0.8\ 1]^T$ e testiamo l'algoritmo *NSB* nel caso di movimento in formazione circolare prima e lineare poi. Le configurazioni di partenza della squadra non hanno rilevanza in quanto il sistema di visone ha il compito di localizzare i robot e come si può notare fin dalla Fig. 28 il successo del task sul baricentro viene soddisfatto in entrambi i casi.

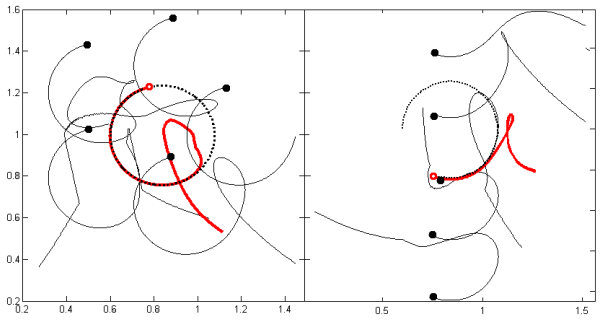

Figura 28: *Movimento desiderato del baricentro in nero tratteggiato, e moto del vero baricentro in rosso nei casi della formazione a cerchio e a linea. Le linee sottili nere sono i riferimenti calcolati per i singoli robot.*

Per quanto riguarda l'esperimento di movimento in formazione circolare, in Fig. $29-30$ sono riportate le traiettorie ottenute: i risultati sono soddisfacenti poichè la cooperazione dei tre comportamenti porta al compimento di tutti i task contemporaneamente, imponendo alla squadra di seguire un moto circolare in formazione rigida. Anche in questo caso bisogna però fare attenzione al tuning dei controllori (i valori dei parametri dei task comuni sono impostati come nel caso precedente e $\Lambda_{form} = 0.3$), in particolare al *feedback linearization* dal quale dipende l'effettivo inseguimento (guadagni unitari).

Analoghe considerazioni si possono fare anche nel

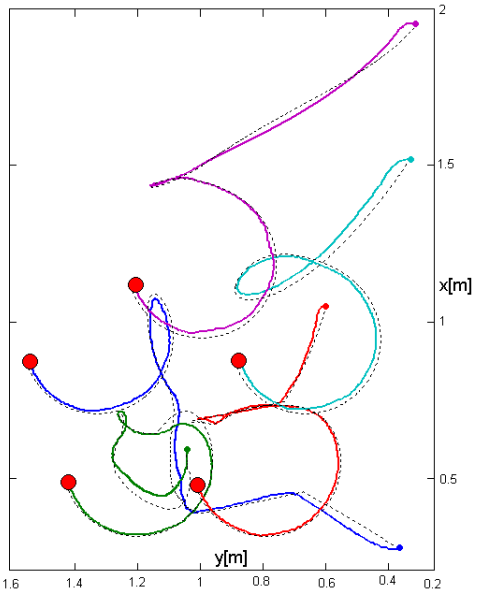

Figura 29: *Traiettorie (linee colorate) dei veicoli e riferimenti (linee nere tratteggiate).*

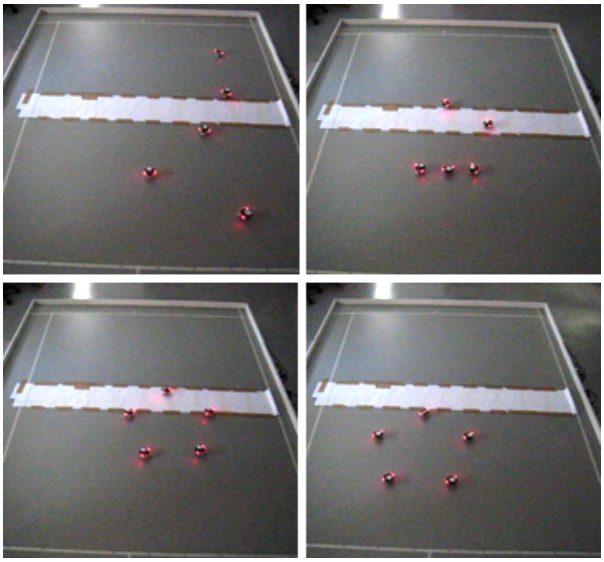

Figura 30: *Obstacle avoidance + convergenza del baricentro + formazione a cerchio in 4 frames.*

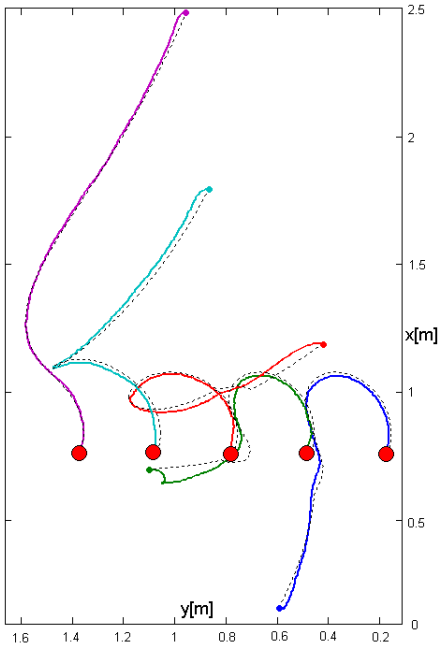

Figura 31: *Traiettorie (linee colorate) dei veicoli e riferimenti (linee nere tratteggiate).*

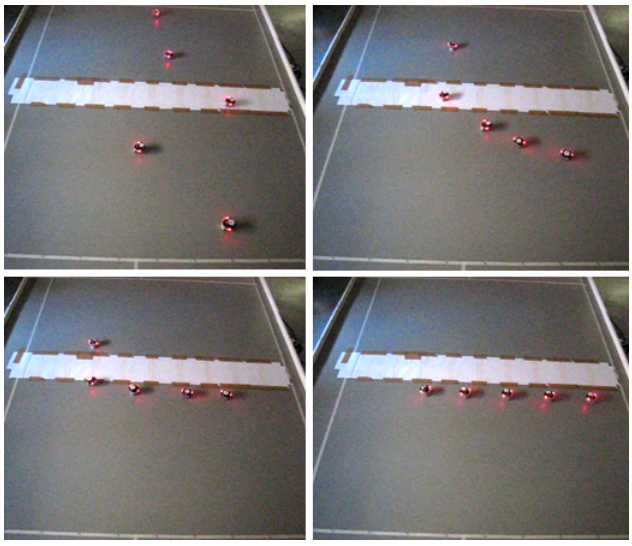

Figura 32: *Obstacle avoidance + convergenza del baricentro + formazione in linea in 4 frames.*

caso di moto in formazione in linea. Come nell'esperimento precedente il task sulla formazione è definito impostando le posizioni desiderate dei singoli robot relative al baricentro, al quale si impone un moto circolare. Le prestazioni risultano alquanto simili essendo infatti caratterizzate dai consueti errori di inseguimento nelle fasi iniziali del controllo, dovuti alla maggior frequenza di cambi di direzione del cammino geometrico elaborato. Una volta però stabilitisi sulle proprie traiettorie i robot reagiscono adeguatamente al *feedback linearization*, con un errore di posizione che tende asintotticamente a zero per la regolarità delle traiettorie stesse.

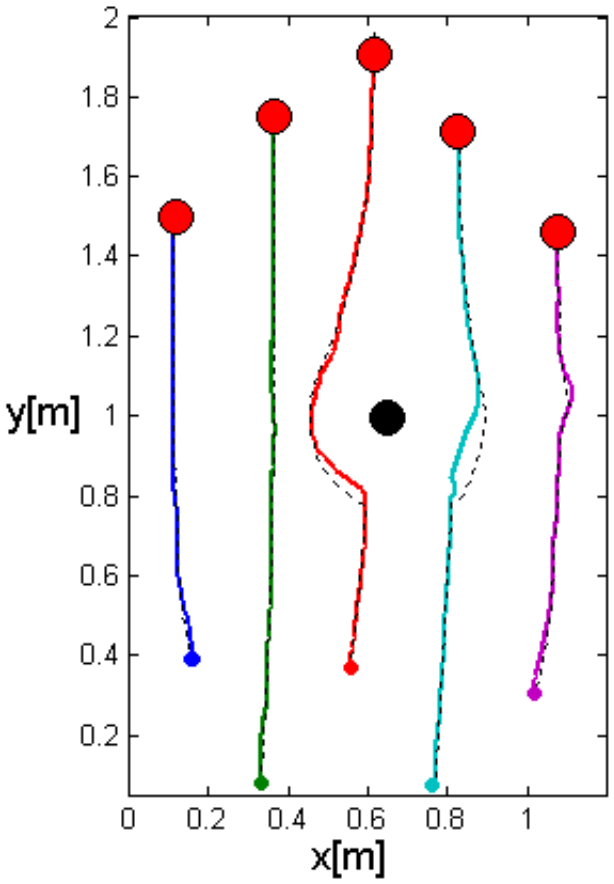

Figura 33: *Traiettorie (linee colorate) dei veicoli e riferimenti (linee nere tratteggiate); il cerchio nero è un ostacolo considerato come fosse puntiforme*

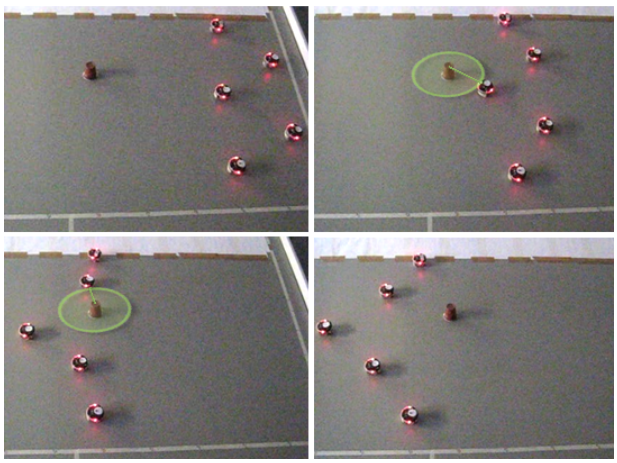

Figura 34: *Obstacle avoidance + convergenza del baricentro + formazione a freccia in 4 frames.*

Proponiamo infine una quarta esperienza, particolarmente significativa per l'analisi delle prestazioni dell'algoritmo di controllo comportamentale *NSB*. Una squadra di cinque veicoli in formazione a freccia deve stabilizzarsi lungo una traiettoria rettilinea in un'area in cui è presente un ostacolo, modellizzato come un punto materiale di posizione nota $[0.65\ 1]^T m$ da schivare.
L'*obstacole avoidance* viene utilizzato per evitare le possibili collisioni sia con gli altri componenti della squadra che con il suddetto ostacolo, dal momento che è stato sviluppato per ostacoli puntiformi, o ostacoli convenientemente rappresentabili da circonferenze. Più precisamante al controllo risultante calcolato per ogni singolo robot, coopera soltanto una componente di evasione determinata dall'ostacolo più vicino (veicolo o ostacolo). L'estensione al caso più generale e frequente di più ostacoli, statici o dinamici, simultanei è implementabile direttamente impostando un ordine gerarchico dipendente proprio dalla distanza degli ostacoli stessi. In Fig. $33 - 34$ sono presentate le traiettorie (linee colorate) effettuate dai singoli robot rispetto ai propri cammini geometrici calcolati (nero tratteggiato): il comportamento risultante, anche se non di priorità più alta, è il mantenimento della formazione a freccia con un moto rettilineo. L'*obstacle avoidance* oltre a salvaguardare l'integrità dei veicoli, si attiva amche quando due robot si avvicinano troppo all'ostacolo puntiforme. Per i commenti circa il corretto inseguimento delle traiettorie si rimanda alle esperienze precedenti, in quanto anche la configuraziona dei parametri dei controllori è rimasta invariata.

Un'ultima importante precisazione riguarda il caso di *task* in conflitto tra loro: se il controllo relativo ad un task secondario risulta essere perfettamente ortogonale allo spazio nullo del task di priorità superiore, il suo contributo si annulla.
Nel particolare caso di un robot che è diretto verso un goal e nella cui traiettoria rettilinea si interpone un ostacolo puntiforme, la componente relativa al raggiungimento dell'obiettivo si annulla come mostrato in Fig. 35. Questa è comunque una situazione che si presenta frequentemente soltanto in simulazione, in quanto il minimo locale che fa sì che il robot si fermi, risulta essere instabile grazie alla presenza di rumore sia nelle misure di localizzazione che soprattutto nei segnali di controllo di velocità.

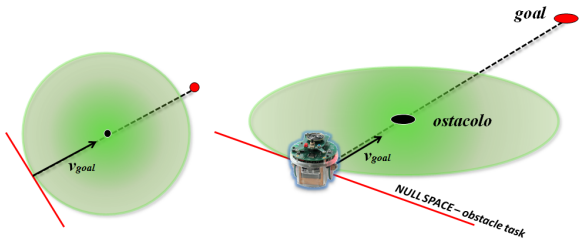

Figura 35: *Ostacolo puntiforme che si interpone esattamente tra robot e goal*

Conclusioni

Uno dei maggiori problemi che interessano la robotica coordinata è il controllo in formazione di una squadra di robot; tale controllo implica che i robot debbano muoversi all'interno dell'ambiente di lavoro mantenendo una determinata configurazione ed evitando ostacoli.
L'interesse scientifico in questo campo è notevolmente aumentato negli ultimi anni grazie al recente progresso tecnologico e ai numerosi vantaggi introdotti dall'uso di sistemi multi-robot (cioè sistemi che utilizzano più di un robot). L'impiego di sistemi multi-

robot permette infatti di aumentare l' efficienza del controllo e permette il raggiungimento di task molto complessi e variegati, rispetto a quanto si può aspirare utilizzando sistemi che utilizzano robot singoli. Un sistema multi-robot inoltre può tollerare e superare possibili guasti o malfunzionamenti dei veicoli, far sì che l'esecuzione del task sia piuttosto flessibile, e ridurre il costo del sistema utilizzando più robot specifici invece che un singolo robot più complesso.

Ci possono essere molteplici applicazioni per la robotica coordinata di sistemi multi-robot come l'esplorazione e il soccorso, la sorveglianza di vaste aree, le missioni di ricerca recupero, la mappatura di ambienti sconosciuti, il trasporto di oggetti grandi e pesanti etc..

In questo progetto è stato implementato l'approccio *Null Space Based* (NSB) per il controllo di un plotone di veicoli mobili (E-puck), il quale permette di poter soddisfare molteplici task, anche in conflitto tra loro ed elaborare la velocità di riferimento di ogni robot utilizzando una robusta struttura matematica. In particolare è stato introdotto il confronto tra i principali approcci disponibili in letteratura (il competitivo, il cooperativo e l' NSB) per sottolineare l'efficacia e i vantaggi offerti dall'approccio NSB. Tale approccio può essere applicato a differenti tipologie di missioni, come il controllo della distribuzione dei robot nell'ambiente di lavoro e il loro successivo moto in formazione fissa, definendo adeguatamente i task da soddisfare. Inoltre risulta robusto anche in presenza di misure rumorose, di disturbi esterni (che anzi spesso si rivelano necessari) e di cambiamenti dinamici dell' ambiente di lavoro.
Tutte queste considerazioni portano a considerare l'NSB come il metodo più efficace e robusto per il controllo di sistemi multi-robot, ma possono comunque essere presi in considerazione alcuni miglioramenti. L'NSB è stato implementato per il controllo dei robot, utilizzando limitate capacità di elaborazione, di performance meccaniche (i veicoli utilizzati sono non - olonomi e si muovono su una pedana di lavoro limitata) e di capacità di comunicazione (il Bluetooth non permette una comunicazione diretta e simultanea tra i vari robot e tra il calcolatore remoto e più di 7 robot). Potendo ottimizzare tutte queste condizioni, e creando subtask differenti e più avanzati per l'algoritmo NSB, molti di questi limiti possono essere superati.

In questo lavoro sono stati quindi illustrati i principali task, la cui esecuzione coordinata porta al compimento di obiettivi anche complessi. Ne è stata fatta un'analisi matematica e delle prestazioni, sia a livello simulativo che a livello sperimentale. Sono state introdotte inoltre migliorie e varianti rispetto al materiale presente in letteratura, tra i quali un nuovo metodo decisionale che incrementa le prestazioni del subtask *obstacle avoidance*, e più in generale si sono realizzati accorgimenti per ogni singolo tipo di controllo, atti a risolvere le problematiche specifiche incontrate in fase di verifica sperimentale.

Il particolare sistema multi-robot utilizzato, è composto da robot omogenei (cioè tutti dello stesso tipo) e l'implementazione effettuata è di tipo centralizzato (cioè i robot sono tutti controllati dallo stesso calcolatore che decide come devono agire); tutti i robot cooperano per portare a termine la medesima missione (come ad esempio portarsi in formazione a cerchio attorno ad un prefissato baricentro e successivamente ruotare in senso orario mantenendo la distanza relativa gli uni dagli altri).
Un passo successivo che si potrebbe approfondire in futuro è quello di un controllo centralizzato di robot eterogenei (vale a dire formalmente diversi tra loro, con diverse caratteristiche e potenzialità), per cui si possono prevedere *task* anche molto complessi.

Ancora più avanzate sono l'analisi e l'implementazione di un controllo di robot eterogenei decentralizzato, rendendo i robot più autonomi e riducendo o addirittura eliminando la comunicazione con il calcolatore remoto, cioè un controllo in cui il robot può decidere autonomamente come agire senza ricevere necessariamente l' ordine del calcolatore.
Quest'ultimo controllo è sicuramente il più robusto e completo poichÈ consente di sviluppare diverse problematiche sempre più utili e frequenti; inoltre in un sistema decentralizzato non è necessario l'utilizzo di un GPS o di una telecamera per ricavare la posizione dei robot ed inviarla all'algoritmo di controllo, ma grazie ai sistemi di sensori montati a bordo ogni singolo veicolo ricava la propria posizione relativamente alla posizione degli altri robot. Si potrebbe pensare infatti di utilizzare una robotica coordinata tramite controllo decentralizzato di robot eterogenei per far svolgere ad ogni robot un compito differente, al fine comunque di raggiungere uno scopo comune.
Un esempio di tipo accademico (già frequentemente

trattato in letteratura con diversi tipi di algoritmi) potrebbe essere quello di far giocare una partita di calcio ad una squadra di robot: ogni robot ha in questo caso un compito diverso a seconda del ruolo da ricoprire: il portiere deve evitare che la palla entri nella porta , il centrocampista deve costruire il gioco in modo da passare la palla all'attaccante, e l'attaccante deve far entrare la palla nella porta avversaria. Lo scopo comune della squadra in ogni caso è quello di vincere la partita. Queste diverse mansioni richiedono che i robot/giocatori siano eterogenei almeno nel tipo di task da compiere . Molte altre applicazioni possono essere realizzate nel campo industriale, militare e soprattutto in quell'area chiamata di *servizio* che sempre maggiore attenzione richiama a sè.

Appendice

A. Apparecchiatura Hardware

Ambiente per la sperimentazione

Per quanto riguarda l'area di lavoro su cui i robot si possono muovere, si è utilizzata una pedana appoggiata sul pavimento, leggermente rialzata, che misura $3.20m \times 2.40m$.

E-puck

L'oggetto del controllo utilizzato per questa esperienza è un veicolo mobile denominato e-Puck. Si nota che la meccanica di tale robot è molto semplice ed elegante: la struttura è basata su un corpo unico del diametro di $70mm$ a cui sono fissati il supporto del motore, il circuito e la batteria. La batteria, allocata al di sotto del motore, è collegata al circuito con due contatti verticali e può facilmente essere estratta per essere ricaricata.

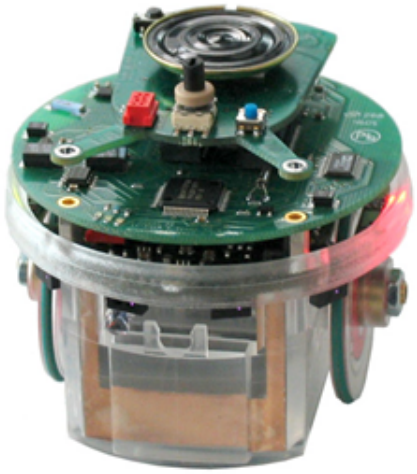

Figura 36: *Epuck*

La scelta del motore è stata orientata verso un piccolo motore a passo con riduttore di marcia. Tale motore realizza 20 passi per rivoluzione e la

marcia possiede una riduzione di $50:1$. Le due ruote hanno un diametro di circa $41mm$, e possiedono un sottile pneumatico di color verde, sono distanti circa $53mm$ e la massima velocità che possono raggiungere è di 100 passi al secondo, che corrisponde ad un giro completo effettuato in un secondo. La comunicazione avviene via bluetooth tramite un chip bluetooth , $LMX9820A$, potendo così comodamente accedere al robot come se fosse connesso ad una porta seriale. L'unica differenza consiste nel fatto che il $LMX9820A$ manda comandi particolari per la sua connessione e sconnessione (intervallo di comunicazione: min : $15cm$, max : $5m$). Il robot e-puck possiede inoltre tre microfoni (massima velocità di acquisizione: $33kHz$), un accelerometro 3D (strumento che misura la forte micro-accelerazione e micro-decelerazione che possiede il robot quando si muove), 8 sensori di prossimità IR (Infrared proximity sensors), una telecamera (risoluzione: $640(h) \times 480(l)$ pixel, a colori) e 8 led colorati (rossi) , tutti controllati utilizzando 24 segnali (Pulse Width Modulated) separati, generati da un microcontrollore posizionato internamente all' e-puck ($PIC18F6722$). Ogni led può essere controllato singolarmente, per una maggiore flessibilità, oppure in maniera sincronizzata, semplicemente settando pochi parametri (come la massima luminosità,il periodo di intermittenza etc.).

Telecamera

Per visualizzare i robot e acquisire la loro posizione in tempo reale si è utilizzata una telecamera AVT Marlin F-131B fissata al soffitto perpendicolarmente al piano di lavoro. Le specifiche tecniche della telecamera utilizzata sono le seguenti:

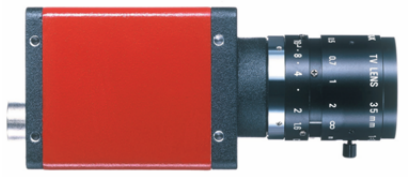

Figura 37: *Marlin-Cam*

Image Device	Typ 2/3 (diag $11mm$)
Effective Picture Elements	$1280(H) \times 1024$ (V)
Picture Size	1280×1024
Cell Size	$6.7\mu m \times 6.7\mu m$
Resolution Depth	8 bit
Lens Mount	C-Mount
Digital Interface	IEEE 1394; DCAM $V1.30$
Transfer Rate	100, 200, 400 Mbit/s
Frame Rates	up to 25 Hz in Format_7
Gain Control	Manual
Shutter Speed	$11\mu s$ to $67s$
External Trigger Shutter	Trigger_Mode_0
Power Requirements	DC $8V$-$36V$
Power Consumption	Less than 3 Watt
Dimensions	$58mm$ x $44mm$ x $29mm$
Mass	¡ $120gr$ (without lens)
Operating Temperature	$+5\deg C$ to $+45\deg C$
Storage Temperature	$-10\deg C$ to $+60\deg C$

Compito fondamentale della retroazione mediante telecamera è il rilevamento della posizione attuale del robot che, in tempo reale, verrà passata all'algoritmo di controllo ogni decimo di secondo. La velocità di acquisizione dell'immagine quindi, diventa di importanza fondamentale per il successo del progetto, e dovrà essere tenuta in considerazione anche negli algoritmi di visione implementati. La telecamera utilizzata (*AVT Marlin F-131B*) è stata scelta poichè considerata molto efficiente: è dotata infatti di un trigger esterno asincrono, che permette di acquisire le immagini in modo istantaneo, senza ritardi significativi. Tale strumento inoltre dà la possibilità di scandire l'immagine dell'intera pedana di lavoro permettendo di sfruttare tutto lo spazio disponibile per le misurazioni.

B. Sistema di visione

Le problematiche che si incontrano nel processo di visione interessa il processing dell'immagine ed in particolare l' interpretazione del significato dei dati che si

ottengono. è necessario quindi segmentare l'immagine, cioè scandire tutti gli elementi che la costituiscono, avendo precedentemente stabilito ciò che si vuole rilevare, cioè i criteri imposti dal tipo di *target*.
Un'accurata segmentazione consente di ottenere la suddivisione dei dati rilevati nel frame che si sta valutando. In condizioni reali di funzionamento, però, esistono alcuni problemi che rendono difficile la ricerca e la localizzazione delle proiezioni dei *target* sul piano dell'immagine:

1. *Agile motion*: si verifica quando il *target* è caratterizzato da una elevata velocità. Ne consegue che il processo di segmentazione risulta più lungo e complesso.

2. *Distraction*: si riferisce alla presenza nella scena di oggetti la cui immagine appare molto simile a quella di uno o più *target*, la cui identificazione diventa quindi più delicata.

3. *Occlusion*: si verifica quando un oggetto si interpone tra il *target* e la telecamera impedendo la visualizzazione di una parte, o dell'intera immagine.

La disciplina che sviluppa questo tipo di tematiche si chiama *data association*.
Il problema denominato come *distraction* è quello che più spesso si potrà verificare nel progetto qui sviluppato: il target da individuare infatti è costituito da alcuni led rossi che vengono rilevati dalla telecamera utilizzata come punti bianchi su sfondo nero. è molto facile che si verifichi la presenza di ulteriori punti bianchi indesiderati all'interno della pedana, come granelli di polvere, riflessi etc., che in ogni caso saranno di luminosità molto minore rispetto al *target*. Tale problema è stato gestito nell'algoritmo di visione senza particolari difficoltà.
I problemi di *agile motion* e di *occlusion* non devono invece essere gestiti dall'algoritmo in quanto si fa in modo, in fase di progettazione, che il target non abbia una velocità troppo elevata (cioè che i robot si muovano ad una velocità tale da essere visualizzati senza problemi) e che il campo visivo della telecamera sia completamente libero.

Per l'acquisizione e l'utilizzo dell'immagine nell'ambiente MATLAB in un primo momento sembrava una soluzione efficiente utilizzare il pacchetto "*imaqtool*", in modo da ottenere la matrice dell'immagine corrente tramite il comando "*getdata*". Non è stato possibile attuare questo procedimento in quanto tale pacchetto MATLAB non riconosce la telecamera connessa via Fire-Wire con il PC. Si è quindi scelto di utilizzare un file a disposizione in rete già compilato (autore Kazuyuki Kobayashi), scritto in linguaggio $C++$, e corredato dalla relativa libreria "*vcapg2.dll*", che permette di acquisire il frame in tempo reale ogni volta che lo si richiama dal prompt di MATLAB.
Con i mezzi hardware messi a disposizione non si sono riscontrati problemi di prospettiva dell'immagine poiché la telecamera è stata collocata e fissata sopra il rettangolo della pedana di lavoro in modo da essergli perfettamente perpendicolare. L'immagine acquisita quindi è corretta, perciò non sono stati necessari calcoli di matrici di trasformazione prospettiche per portarsi dal piano della telecamera al piano di lavoro.

La telecamera utilizzata acquisisce immagini in bianco e nero (scala di grigi) perciò, dopo numerose prove, si è scelto di utilizzare una luce molto ridotta in modo da visualizzare la pedana di lavoro come fosse nera, e di distinguere quindi i robot come fossero punti bianchi. Il pacchetto *vcapg2* viene chiamato dall'algoritmo e restituisce una matrice dell'immagine, di dimensioni pari alla risoluzione della telecamera ($960 \times 1280 \times 1$), che verrà analizzata per ottenere le informazioni necessarie.
Per riuscire ad individuare la posizione e la direzione relativa ad ogni robot e fare in modo che, durante tutto lo svolgimento della prova, ogni robot sia individuato univocamente, si è fatto uso di tre led rossi. Tali led (che vengono rilevati dalla telecamera come punti bianchi) sono posti in modo da formare un triangolo isoscele, la cui base è pari al diametro, e l'altezza corrisponde alla metà del diametro del robot. La posizione viene quindi rilevata calcolando il punto medio tra i due led, la cui distanza corrisponde al diametro (34 pixel), ottenendo in questo modo la posizione esatta (a meno di pochi pixel) del centro del robot. Il led restante invece (distante dagli altri due led di una quantità minore del diametro) viene utilizzato per il riconoscimento della direzione del robot: collegando tale punto a quello che indica il centro del robot si ottiene la retta necessaria per calcolare l'angolo di scostamento *theta* ad ogni passo.
Realizzando tale metodo di riconoscimento della posizione si può presentare un unico inconveniente: se

due o più robot si trovano ad una distanza minore o uguale di 34 pixel (diametro di ogni robot) l'uno dall'altro, allora l'algoritmo di visione non riesce più a distinguerli e si perde l'informazione desiderata. Ciò non viene considerato rilevante, in quanto uno dei più importanti *task* del progetto che si è sviluppato prevede che i robot non possano mai scontrarsi, perciò si può tranquillamente supporre che restino sempre ad una certa distanza tra loro (da impostare nell'algoritmo di controllo) e quindi che la posizione e la direzione di ognuno sia sempre nota e chiara.
La procedura che porta a far sì che tutti i robot vengano visualizzati ed individuati all'interno della pedana di lavoro può essere schematizzata come segue.
Finché sono presenti frame disponibili, cioè a seconda del tempo che si decide di avere a disposizione per far lavorare i robot:

1. Si acquisisce l'immagine;
2. Si cercano i punti diversi da zero (soglia che indica la presenza del colore nero) nella matrice dell'immagine, memorizzando sia il valore dei punti trovati, che le coordinate cartesiane degli stessi. Tali punti, considerati come "bianchi", sono certamente di più di quanti ci si aspetti (tre punti per ogni robot), ma in ogni caso non superano qualche decina (quantità decisamente irrilevante rispetto al numero di pixel che costituiscono la matrice).
   I punti che interessa rilevare sono certamente di un bianco più intenso rispetto a quelli che si vuole tralasciare, perciò, tra tutti i punti bianchi appena individuati, si ricercano ad uno ad uno quelli che hanno il massimo valore (intensità luminosa maggiore), finché non si è acquisito il numero di punti desiderato. Il vettore "*point*" conterrà quindi tutte le posizioni dei led rilevati, espresse in coordinate *(x,y)*;
3. Si richiama la funzione *findRobot* che, dato il vettore "*point*" delle posizioni dei punti bianchi, identifica il numero di robot stabilito, e ritorna un vettore (*pos_robot*) contenente il loro elenco (all'interno di ogni oggetto robot è contenuta l'informazione relativa alla sua posizione e alla sua direzione);
4. Se il numero di robot trovati corrisponde al numero reale, cioè se non si sono verificati errori, per il primo ciclo si associano al vettore dei robot ("*robot*") gli oggetti appena trovati con la funzione "*findRobot*". Dal secondo ciclo in poi invece, si associano le nuove posizioni ai robot precedentemente recuperati;
5. Se il numero di robot trovati non corrisponde al numero reale, viene visualizzata la stringa: "*ERROR: i robot trovati non corrispondono*", e semplicemente non considera il frame.

Riferimenti bibliografici

[1] G. Barbera, "Controllo del Khepera: modellizzazione, simulazione e sintesi di algoritmi di controllo per l'uniciclo" *Tesi di laurea triennale*, Università degli studi di Padova, Luglio 2006.

[2] G. Cosi, "Controllo del Khepera con localizzazione tramite rete di sensori wireless" *Tesi di laurea triennale*, Università degli studi di Padova, Marzo 2007.

[3] E. Piaggio, "Dinamica e controllo dei veicoli robotici" , Università degli studi di Pisa.

[4] J. R. T. Lawton, B. J. Young, R. W. Beard,"A decentralized approach to elementary formation maneuvers", 22 Settembre 1999.

[5] G. Oriolo, A. De Luca, M. Vendittelli, "WMR control via dynamic feedback linearization: design, implementation, and experimental validation" *IEEE Transactions on control systems technology, vol.10, n° 6, pagg. 835-852*, Novembre 2002.

[6] B. J. Young, R. W. Beard, J. M. Kelsey, "A control scheme for improving multi-vehicle formation maneuvers", Brigham Young University.

[7] R. W. Beard, J. Lawton, F. Y. Hadaegh, "A coordination architecture for spacecraft formation control" *IEEE Transactions on control systems technology*, Luglio 2000. Novembre 2002.

[8] T. Balch, R. C. Arkin, "Behavior-based formation control for multi-robot teams" *IEEE Transactions on robotics and automation*, 1999.

[9] G. Oriolo, A. De Luca, "Modeling and control of nonholonomic mechanical system", Dipartimento di Informatica e Sistemistica Università degli Studi di Roma ìLa Sapienzaî, Agosto 2001.

[10] Jean-Claude Latombe, "Robot motion planning" *The Kluwer International Series in Engineering and Computer Science*,Kluwer Academic Publishers, 1991.

[11] Jules M. Vleugels, Joost N. Kok, Mark H. Overmars, "Motion Planning Using a Colored Kohonen Network", ESPIRIT Basic Research Ation and Netherlands Organization for Scientific Research.

[12] Julien Burlet, Olivier Aycard, Thierry Fraichard, "Robust Motion Planning using Markov Decision Processes and Quadtree Decomposition", Institut National de Recherche en Informatique et en Automatique and Lab. Graphisme, Vision et Robotique 2004.

[13] L. Ricciato, E. Siego, D. Tamino, "Pianificazione del moto e controllo di un uniciclo", *Progettazione di Sistemi di Controllo*, Università degli studi di Padova - Dipartimento di Ingegneria dell'Automazione, 2006.

[14] S. Sekhavat, M. Chyba, "Nonholonomic Deformation of a Potential Field for Motion Planning", INRIA Rhone-Alpes, Montbonnot and Division Of Applied Sciences Harvard University, Cambridge.

[15] Richard M. Murray, S. Shankar Sastry, "Nonholonomic Motion Planning: Steering Using Sinusoids", *IEEE Transactions on Automatic Control*, Maggio 1993 .

[16] Alessandro Agnoli, "Controllo di veicoli anolonomi su ruota", *Tesi di laurea Specialistica*, Università degli studi di Padova - Dipartimento di Ingegneria dell'automazione, Ottobre 2007 .

[17] M. Caregaro Negrin, A. Nicolettis, P. Pucci, "Very-low sampling rate control of wheeled mobile robots using circles", *Technical report*, Università degli studi di Padova - Dipartimento di Ingegneria dell'automazione, Ottobre 2006.

[18] L. Consolini, F. Morbidi, D. Prattichizzo, M. Tosques, "On the Control of a Leader-Follower Formation of Nonholonomic Mobile Robots", *Collaborazione tra Dipartimanto di Ingegneria dell'Informaione dell'Università di Siena e dell'Università di Parma.*

[19] Wei Ren, Randal W. Beard, "Dynamic Consensu Seeking in Distributed Multi-agent Coordinated Control", *BYU MAGICC Lab Technical Report*, Department of Eletrical and Coordinated Control - Brigham Young University, Novembre 2003.

[20] P. Ogren, E. Fiorelli, N. E. Leonard, "Formation with a Mission: Stable Coordination of Vehicle Group Maneuvers", *Research sponsored by*

the Swedish Foundation for Strategic Research, Royal Institute of Technology of Stockholm and Princeton University.

[21] Lynne E. Parker, "ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation", *IEEE Transactions on Robotics and Automation, Vol. 14, No. 2*, Center for Engineering System Advanced Research Oak Ridge National Laboratory, Aprile 1998.

[22] P. Chiacchio, S. Chiaverini, L. Sciavicco and B. Siciliano, "Closed-loop inverse kinematics schemes for constrained reduntant manipulators with task space augmentation and task priority strategy", *The International Journal Robotics Research*, 10(4):410-425, 1991.