SAPIENZA
UNIVERSITÀ DI ROMA

Inseguimento di traiettorie ed evitamento di ostacoli tramite Control Barrier Functions: implementazione via Switching Control

Dipartimento di Ingegneria Informatica, Automatica E Gestionale "Antonio Ruberti"

Corso di Laurea in Ingegneria Informatica e Automatica

Candidato

Carlo La Sala
Matricola 1956560

Relatore

Prof. Andrea Cristofaro

Anno Accademico 2022/2023

Inseguimento di traiettorie ed evitamento di ostacoli tramite Control Barrier Functions: implementazione via Switching Control
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: lasala.1956560@studenti.uniroma1.it

Sommario

L'obiettivo di questo lavoro è di ideare e progettare un controllore per garantire l'inseguimento di traiettorie per robot mobili con vincoli di sicurezza utilizzando la logica di controllo basata sullo Switching Programming che è un approccio di controllo basato su un controllore dinamico, il quale restituisce input di controllo differenti a seconda delle situazioni.
Al giorno d'oggi le tecniche di controllo maggiormente utilizzate per questo task utilizzano il Model Predictive Control, il Branch Informed Trees e algoritmi di controllo basati su problemi di ottimizzazione, come il Quadratic Programming (implementato parzialmente anche in questo lavoro per confrontarlo con lo Switching Programming).
Le applicazioni più comuni di queste leggi di controllo sono principalmente nell'ambito della robotica industriale, della domotica e della robotica marina e aerea.
Nonostante la grande efficacia di questi algoritmi, per quanto riguarda ambienti non strutturati e fortemente dinamici, la ricerca si propone di trovare leggi di controllo orientate ai vincoli sugli ostacoli e per questo motivo si introduce l'utilizzo delle Control Barrier Functions. Le CBFs sono un insieme di funzioni matematiche che permettono di imporre dei vincoli di sicurezza. Definita una CBF, possiamo utilizzarla per imporre al robot mobile di rimanere all'interno di un insieme di sicurezza stabilito e, di conseguenza, la CBF si comporta come una vera e propria funzione di barriera.
Prima di iniziare il progetto del controllore è necessario definire un modello del robot mobile per poter individuare le matrici che definiscono la dinamica del sistema. Partiamo dall'equazione differenziale che governa il moto di un punto materiale e in seguito definiamo il vettore di stato del sistema come l'insieme delle coordinate x, y e delle velocità lungo i due assi. Possiamo considerare quindi il robot mobile come un doppio integratore.
A questo punto possiamo implementare il controllore basato su Switching Programming. Definiamo innanzitutto un input di controllo nominale basato su feedforward+PD per inseguire la traiettoria di riferimento. Successivamente dividiamo lo spazio di stato in due insiemi: all'interno del primo la priorità è seguire la traiettoria di riferimento, all'interno dell'altro la priorità è evitare gli ostacoli. Dopo aver definito la CBF appropriata, definiamo il controllo dinamico che sarà il controllo nominale quando il roobt si trova nel primo insieme, mentre sarà un controllo modificato opportunamente quando deve evitare gli ostacoli.
Il lavoro prosegue con l'implementazione in Matlab del controllore basato su Switching Programming e anche di un controllore basato su Quadratic Programming (utilizzando quadprog di Matlab) per effettuare dei confronti. La prima implementazione riguarda un robot mobile puntiforme per semplificare il problema, mentre la seconda implementazione estende la prima ad un robot di raggio RR.
Infine tramite delle simulazioni si è notato come il controllore di tipo Switching sia molto più attento a mantenere un errore di regolazione molto piccolo soprattutto quando il robot entra in modalità di evitamento ostacoli. Nonostante questa minima differenza, i due controllori si comportano in modo simile in ogni scenario, tranne in un paio di casi in cui l'approccio è leggermente diverso per via di alcuni parametri di tuning del controllore di tipo Switching, che se opportunamente modificati riportano i due controllori a lavorare in modo pressochè identico.

Indice

Capitolo 1

Introduzione

In questo capitolo introduttivo viene descritto il problema dell'inseguimento di traiettorie per robot mobili con vincoli di sicurezza e vengono illustrate le possibili applicazioni nella vita reale delle soluzioni trattate.

1.1 Descrizione del problema

L'automazione e la robotica stanno diventando sempre più importanti nella società moderna, in quanto le tecnologie robotiche sono impiegate in una vasta gamma di applicazioni. In questo contesto uno degli aspetti più critici della robotica mobile è l'inseguimento di traiettorie, ovvero la capacità del robot di seguire una traiettoria pianificata nel modo più preciso e rapido possibile. Tutto ciò deve essere ottenuto in maniera safe, quindi evitando possibili impedimenti e garantendo la sicurezza delle persone che lavorano a stretto contatto con questi robot.

Nonostante la larga diffusione dei robot mobili e la richiesta sempre maggiore di essi, il controllo della traiettoria dei robot mobili è una sfida complessa, poiché deve tener conto di numerosi fattori, come le dinamiche interne del robot, le condizioni ambientali e, soprattutto, i vincoli di sicurezza. Infatti, un robot mobile che si sposta in ambienti condivisi con esseri umani deve garantire la massima sicurezza per le persone e le attrezzature circostanti.

Al giorno d'oggi sono varie le principali tecniche impiegate per il controllo di traiettorie safe per i robot mobili. Alcune di esse sono: tecniche basate su algoritmi di pianificazione di traiettorie, tecniche di controllo ibrido, tecniche di filtraggio dello stato e tecniche di apprendimento automatico. In particolare, molto diffuse e affidabili sono le tecniche basate su Model Predictive Control e problemi di ottimizzazioni. L'utilizzo del MPC si basa sulla definizione di un modello matematico che potrebbe essere costituito dalle equazioni che regolano la dinamica e la cinematica del robot. Questo modello, può quindi, essere utilizzato per prevedere l'evoluzione del sistema nel futuro, ad esempio la posizione e l'orientamento del robot sulla base di una traiettoria di riferimento prefissata, tenendo conto di determinati vincoli di sicurezza.

Un'altra tecnica molto innovativa è quella del BIT* (Batch Informed Trees), che è stato proposto per risolvere il problema della pianificazione dei percorsi in ambienti dinamici e incerti. Inoltre BIT* punta a migliorare le prestazioni dell'algortimo RRT (Rapidly-exploring Random Tree), basato su una gestione batch degli eventi di collisione. L'idea alla base di BIT* è quella di dividere l'ambiente in diverse regioni e di eseguire una pianificazione dei percorsi in ciascuna di esse in modo separato. In questo modo, si riduce la complessità computazionale dell'algoritmo e si evita di dover ripianificare l'intero percorso ogni volta che si verificano delle collisioni. In

particolare, per rappresentare lo spazio di ricerca delle traiettorie, BIT* utilizza una struttura di dati ad albero. L'algoritmo procede creando iterativamente un nuovo albero, generando dei campioni casuali nell'ambiente e cercando di connetterli al resto dell'albero. Nel caso in cui il nuovo campione crei una collisione, BIT* non lo scarta immediatamente, ma lo conserva in una lista di "eventi di collisione" da risolvere in batch. Una volta che l'albero raggiunge una certa dimensione, BIT* esegue una fase di ottimizzazione del percorso, in cui cerca di trovare il percorso più corto tra il punto di partenza e la destinazione. In questa fase, BIT* utilizza l'elenco degli eventi di collisione per generare nuovi campioni che possano risolvere le collisioni esistenti.

Nonostante l'automazione di questi sistemi robotici sia già molto sviluppata, recenti studi mostrano la necessità di sviluppare nuovi approcci di controllo che si basano in prima istanza sulla gestione dei vincoli di sicurezza. Questa priorità va affiancata alle già consolidate tecniche di feedforward per garantire l'inseguimento del riferimento desiderato.

Questo lavoro propone due approcci e in particolare si sofferma sulla progettazione di uno di essi per risolvere il problema dell'inseguimento di traiettorie per robot mobili con vincoli di sicurezza, utilizzando la teoria delle Control Barrier Functions come base per l'implementazione delle leggi di controllo. In particolare trattiamo il caso di robot mobile a due gradi di libertà dotato di ruote omnidirezionali intento a inseguire una traiettoria circolare o ellittica in uno scenario popolato da ostacoli. Il primo approccio è basato sulla programmazione quadratica (*Quadratic Programming*) e quindi sulla risoluzione di un problema di ottimizzazione, grazie alla funzione *quadprog* di Matlab. L'altro approccio è trattato più approfonditamente ed è basato sul concetto di controllo Switching.

1.2 Possibili applicazioni

L'inseguimento di traiettorie per robot mobili con vincoli di sicurezza è un campo di ricerca molto attivo e già da molti anni ha trovato ampio spazio di sviluppo in molte applicazioni pratiche. In particolare delle applicazioni comuni sono nella robotica industriale, nella robotica aerea e marina e nella domotica. Nel caso della robotica industriale il robot di solito è impiegato per spostare materiali all'interno di fabbriche garantendo sicurezza agli operatori umani e alle attrezzature. Il caso della robotica marina e aerea è molto delicato in quanto, la navigazione di questi robot avviene all'interno di ambienti fortemente dinamici. Infine, nella domotica l'utilizzo più comune è applicato ai robot di pulizia domestica e anch'essi si muovono all'interno di scenari dinamici e non strutturati. A tal proposito è molto importante l'approccio mediante CBFs poichè garantiscono robustezza di controllo, sicurezza, velocità di calcolo e, soprattuto, flessibilità che è la propietà fondamentale per gestire al meglio la navigazione in questa tipologia di ambienti.

Capitolo 2

Implementazione

In questo capitolo è trattata l'implementazione, partendo da un modello semplificato del sistema Robot mobile, dei due algoritmi per l'inseguimento di traiettorie per robot mobili con vincoli di sicurezza, con maggiore interesse alla tecnica dello Switching Programming.
Inoltre viene introdotto il concetto di Control Barrier Functions, utilizzate per implementare i due controllori.
Successivamente è mostrato come ciascuna tecnica sia stata implementata su Matlab per delle simulazioni, prima nel caso semplificato di robot mobile puntiforme e in seguito nel caso di robot mobile di raggio RR.

2.1 Il concetto di Control Barrier Function

Per poter passare alla risoluzione del problema dobbiamo innanzitutto familiarizzare con il concetto di Control Barrier Functions, in quanto è fondamentale all'interno dell'implementazione dei controllori per risolvere questo task.

Le Control Barrier Functions (CBFs) sono un insieme di funzioni utilizzate per la progettazione di controlli che portino i sistemi dinamici a lavorare in condizioni di sicurezza.

Formalmente, dato un sistema dinamico descritto dalla seguente equazione differenziale:

$$\dot{x} = f(x, u)$$

dove $x \in \mathcal{D} \subset \mathbb{R}^n$ rappresenta il vettore di stato del sistema, $u \in \mathcal{U} \subset \mathbb{R}^m$ è il vettore degli ingressi e $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ è una funzione non lineare che descrive l'evoluzione del sistema, allora possiamo definire una CBF come una funzione continua e differenziabile $h : \mathbb{R}^n \to \mathbb{R}$ tale che:

$$h(x) > 0, \quad \forall x \in \mathcal{C}$$

$$\dot{h}(x) + \alpha(h(x)) \geq 0, \quad \forall x \in \mathcal{C}$$

dove $\alpha : \mathbb{R} \to \mathbb{R}$ è una funzione di classe $\mathcal{K}$ (cioè una funzione continua, crescente e strettamente positiva con $\alpha(0) = 0$), $\mathcal{C} \subset \mathbb{R}^n$ è l'insieme di sicurezza desiderato. In particolare possiamo definire formalmente l'insieme di sicurezza in questo modo:

$$\mathcal{C} = \{x \in \mathcal{D} \subset \mathbb{R}^n : h(x) \geq 0\}$$

In altre parole, le condizioni definiscono che la funzione di barriera $h(x)$ è sempre positiva all'interno dell'insieme di sicurezza stabilito.

In pratica, se si progetta un controllore che garantisce che la funzione di barriera $h(x)$ sia sempre positiva durante l'evoluzione del sistema, allora si può essere sicuri che il sistema non violerà mai i vincoli di sicurezza.

Inoltre, le CBFs consentono di progettare controllori robusti rispetto alla presenza di disturbi e incertezze nel sistema. Infatti, se la funzione di barriera $h(x)$ può essere garantita positiva in presenza di disturbi e incertezze, allora si può essere sicuri che il sistema rimarrà sempre all'interno dell'insieme di sicurezza desiderato, indipendentemente dalle perturbazioni esterne. Un'altra importante proprietà delle CBFs è che possono essere combinate tra loro per garantire la sicurezza di un sistema complesso costituito da più sotto-sistemi interconnessi. Infine, possono essere utilizzate anche per progettare controllori in grado di garantire la stabilità di sistemi non lineari.

Nel caso particolare del task di inseguimento di traiettorie in maniera safe per robot mobili, le Control Barrier Functions giocano un ruolo fondamentale in quanto permettono di creare una sorta di muro virtuale, che il robot non può attraversare, intorno agli ostacoli. Questo "muro virtuale" è rappresentato da una funzione di barriera che definisce una regione nello spazio di stato del sistema che il robot non può attraversare e con la quale non può entrare in contatto.

2.2 Modello del processo

Per poter sviluppare una legge di controllo in grado di far inseguire al robot traiettorie in condizioni safe, dobbiamo prima di tutto definire un modello del processo.
Consideriamo l'equazione differenziale di secondo ordine

$$\ddot{p}(t) = u(t), p \in \mathbb{R}^n, u \in \mathbb{R}^n. \tag{2.1}$$

Interpretando quest'equazione differenziale come la dinamica che regola il moto di un punto materiale, p rappresenta il vettore delle coordinate Cartesiane del centro di massa del robot mentre u rappresenta il vettore degli ingressi e quindi dell'input di controllo.
É utile notare come i sistemi di n dimensioni con integratori di ordine r disaccoppiati possono rappresentare la dinamica di sistemi non lineari esattamente linearizzabili attraverso un feedback statico o dinamico, tra cui una vasta gamma di robot mobili ed è quindi possibile utilizzare questo modello semplificato.
Il vettore di stato del sistema dinamico sopra citato è quindi $\mathbf{z} = [\mathbf{p}, \mathbf{v}]$, dove p è il vettore delle coordinate x e y e v è il vettore delle velocità lungo x e y.
Per ottenere la dinamica del sistema, deriviamo l'equazione rispetto al tempo e quindi $\dot{\mathbf{z}} = [\dot{\mathbf{p}}, \dot{\mathbf{v}}]$, da ciò otteniamo che $\dot{\mathbf{z}} = [\mathbf{v}, \mathbf{u}]$. Questa equazione deriva dalla definizione della dinamica del sistema, dove la posizione del robot è determinata dalla sua velocità e la velocità è determinata dall'ingresso di controllo u(t).
Siamo interessati a trovare un modello del processo del tipo $\dot{x} = Ax + Bu, \quad y = Cx$ e quindi sostituendo il vettore di stato a x otteniamo

$$\dot{z} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} z + \begin{bmatrix} 0 \\ I \end{bmatrix} u, \quad y = \begin{bmatrix} I & 0 \end{bmatrix} z \tag{2.2}$$

Abbiamo quindi ottenuto un modello del sistema nello spazio di stato con le matrici A, B e C esplicitate. Dato che stiamo considerando un ambiente 2D, ciascun elemento delle matrici sarà un blocco $\in \mathbb{R}^{2x2}$ e il vettore dello spazio di stato avrà dimensione 4×1.
Siamo inoltre interessati a lavorare con un sistema che ad anello chiuso abbia un errore a regime permanente nullo nei confronti della traiettoria desiderata. Per far ciò è necessario progettare un controllore PID e possiamo utilizzare le sole due azioni P e D. L'azione P aumenta la prontezza di risposta e riduce l'errore a regime mentre l'azione D anticipa l'errore, evitando che l'uscita si allontani dal riferimento a causa dell'accelerazione data dall'azione proporzionale e tende a stabilizzare il sistema, anche se rallenta la risposta del sistema.
Utilizzando il feedback del PD, quando il robot è lontano dalla traiettoria di riferimento tenderà ad avvicinarsi e a ridurre a zero l'errore di regolazione.
Quindi alle matrici A, B e C possiamo affiancare una matrice dei guadagni K tale che: $K = \begin{bmatrix} k_p & 0 & k_d & 0 \\ 0 & k_p & 0 & k_d \end{bmatrix}$.

2.3 La tecnica dello Switching Programming

Per affrontare il problema dell'inseguimento di traiettorie sono utilizzabili varie tecniche. In questa sezione approcciamo il problema mediante lo Switching Programming.
Lo Switching Programming è un approccio di controllo dinamico per i robot mobili che consente loro di inseguire traiettorie di movimento sicure. In particolare, lo Switching Programming consente ai robot mobili di cambiare modalità di controllo in modo dinamico durante il movimento, in modo da poter gestire diversi scenari di navigazione in modo efficace e sicuro.

Lo Switching Programming per i robot mobili utilizza la tecnica di controllo mediante Control Barrier Function (CBF) che viene utilizzata per definire un limite di sicurezza per il sistema. In pratica, utilizziamo la teoria delle CBFs per progettare il controllore che si baserà su una logica di tipo switching grazie all'uso di una CBF appropriata.

L'idea di base dello Switching Programming per i robot mobili è quella di definire un insieme di modalità di controllo, ognuna delle quali è progettata per gestire uno specifico scenario di navigazione. In questo caso definiamo due leggi di controllo:

- legge di controllo nominale u^* che permette al robot di inseguire la sua traiettoria di riferimento;
- legge di controllo u per garantire la sicurezza in prossimità di ostacoli.

Durante il movimento del robot mobile, il sistema di controllo utilizza la CBF per valutare le condizioni ambientali e decidere quale modalità di controllo utilizzare in base alla situazione corrente. Se il sistema rileva la presenza di ostacoli sulla traiettoria prevista, il sistema passa automaticamente alla modalità di controllo appropriata per evitare gli ostacoli e garantire la sicurezza del sistema.

In sintesi, lo switching programming mediante Control Barrier Function consente ai robot mobili di navigare in modo sicuro e efficiente in diversi scenari di navigazione, passando dinamicamente da una modalità di controllo all'altra in base alle condizioni

ambientali. Questo approccio di controllo dinamico è particolarmente adatto per applicazioni in cui la sicurezza del sistema è una priorità assoluta, come nell'ambito dell'automazione industriale, della logistica o della robotica collaborativa.

Consideriamo ora la funzione $h(p(t), \dot{p}(t))$ definita in questo modo:

$$h(p(t), \dot{p}(t)) = (p(t) - \overline{p})^T (p(t) - \overline{p}) + \mu (p(t) - \overline{p})^T (\dot{p}(t)) \tag{2.3}$$

Nell'equazione precedente, $\overline{p}$ rappresenta il vettore delle coordinate dell'ostacolo e μ è una costante positiva. L'equazione descrive la distanza tra il centro di massa del corpo e la posizione dell'ostacolo, con un termine aggiuntivo che tiene conto della direzione del vettore distanza rispetto alla velocità del sistema. L'obiettivo è garantire uno spazio libero dall'ostacolo, definendo il controllo u in modo che la funzione descritta dall'equazione sia sempre maggiore di una costante positiva δ che rappresenta il raggio dell'ostacolo. La CBF è quindi

$$\overline{h}(p(t), \dot{p}(t)) = h(p(t), \dot{p}(t)) - \delta \tag{2.4}$$

Vogliamo quindi trovare una legge di controllo u tale che: $\dot{h}(t) + \alpha(h(t) - \delta) \geq 0$. Il termine α è una costante positiva fissata. Quindi essendo $\overline{h}(t)$ una CBF, un controllore che la utilizza garantisce che una traiettoria che parte da un punto safe dello spazio resti all'interno dell'insieme safe.

Definiamo ora due operatori di proiezione per proiettare l'accelerazione del robot sul piano ortogonale alla direzione che punta verso l'ostacolo:

- $\prod_{p-\overline{p}} = (p - \overline{p})[(p - \overline{p})^T (p - \overline{p})]^{-1} (p - \overline{p})$;
- $\prod^{\perp}_{p-\overline{p}} = I - \prod_{p-\overline{p}}$.

Questi due operatori di proiezione, come vedremo in seguito saranno importanti per definire il controllore.

Definiamo la legge di controllo nominale u^* per inseguire la traiettoria desiderata $p_d(t)$.

$$u^* = \ddot{p_d} + k_d(\dot{p_d} - \dot{p}) + k_p(p_d - p) \tag{2.5}$$

Settando $u = u^*$, il robot insegue il riferimento che gli è stato dato.

Ora dobbiamo definire l'altra "modalità" del controllore, ossia quella che permette al robot di evitare in sicurezza gli ostacoli.
L'idea è di dividere lo spazio di stato $\mathbb{R}^{2n}$ in due sottoinsiemi D_{track} e $D_{\perp}$. Il primo è l'insieme in cui la priorità è di seguire asintoticamente la tariettoria di riferimento $p_d(t)$, mentre nel secondo insieme la priorità è di evitare in sicurezza gli ostacoli. In particolare $D_{track} = D_{u^*} \cup D_{\delta_1}$ e $D_{\perp} = \mathbb{R}^{2n} \setminus D_{track}$, dove:
$D_{u^*}(t) = \{(p, \dot{p}) : (p - \overline{p})^T (\mu u^* + 2\dot{p}) > 0\}$ e $D_{\delta_1} = \{(p, \dot{p}) : h(p, \dot{p}) > \delta_1 > \delta\}$. I termini δ e δ_1 rappresentano rispettivamente il raggio dell'ostacolo e un raggio di "sicurezza", dove il robot può passare, pur sapendo che si sta avvicinando all'ostacolo.

Abbiamo quindi definito in questo modo il controllore dinamico per l'inseguimento di traiettorie per robot mobili con vincoli di sicurezza con la tecnica dello Switching Programming:

$$u = \begin{cases} u^* & if \quad (p, \dot{p}) \in D_{track} \\ (-2/\mu) \prod_{p-\overline{p}} \dot{p} + \prod^{\perp}_{p-\overline{p}} u^* & if \quad (p, \dot{p}) \in D_{\perp} \end{cases} \tag{2.6}$$

2.4 Robot puntiforme

Dopo aver definito un modello del sistema e aver compreso come utilizzare al meglio le Control Barrier Functions e la tecnica dello Switching Programming, implementiamo la legge di controllo per un robot mobile puntiforme che si muove in uno spazio 2D in presenza di uno o più ostacoli. Inoltre assumiamo che gli ostacoli non possono essere sovrapposti e che il robot parta da una zona safe.

Implementeremo anche un controllore basato sulla tecnica del Quadratic Programming (QP). La tecnica del QP è una tecnica utilizzata per risolvere problemi di ottimizzazione vincolati, in cui l'obiettivo è minimizzare una funzione costituita da un termine quadratico soggetto a un insieme di vincoli lineari e non.

Nell'inseguimento di traiettorie per robot mobili in condizioni di sicurezza, la tecnica del QP viene spesso utilizzata per generare i comandi del robot in modo che si muova lungo una traiettoria desiderata, evitando al contempo di urtare gli ostacoli. Utilizzando la tecnica del QP, è possibile formulare un problema di ottimizzazione che cerca di minimizzare la distanza tra la posizione del robot e la traiettoria desiderata, soggetto a vincoli che impongono che il robot non urti gli ostacoli. In questo modo, la tecnica del QP consente di ottenere comandi di controllo per il robot che minimizzano l'errore rispetto alla traiettoria di riferimento, garantendo al contempo la sicurezza del robot e delle persone e degli oggetti che si trovano intorno ad esso.

In primo luogo in Matlab, per entrambe le tecniche definiamo il modello matematico del processo con le matrici A e B .
In seguito definiamo tutte le costanti fisse del problema: $\alpha = 5, \mu = 0.05, \delta = 2, \delta_1 = 5, k_p = 100, k_d = 3, \omega = \pi/2$.
Dobbiamo impostare una posizione di partenza e una velocità di partenza al nostro robot e questo non sarà altro che lo stato iniziale del sistema: $\mathbf{x_0} = [\mathbf{O_x}, \mathbf{O_y}, \mathbf{0}, \mathbf{0}]$.
Il robot parte quindi da posizione iniziale O_x e O_y e con velocità iniziale nulla.
Successivamente posizioniamo gli ostacoli all'interno del nostro scenario e per rendere il tutto più dinamico in modo che ad ogni simulazione possiamo decidere arbitrariamente quanti ostacoli posizionare, possiamo definire un vettore di ostacoli, dove ciascun elemento è una tupla così definita: $\mathbf{Obstacles_i} = [\mathbf{Obs_x}, \mathbf{Obs_y}]$.
A questo punto dobbiamo definire una traiettoria di riferimento da far inseguire al robot mobile.
Possiamo impostare una traiettoria circolare o ellittica e per generalizzare definiamo una traiettoria del tipo: $p_d(t) = [x_{0_{track}} + a * \sin(\omega t), y_{0_{track}} + b * \cos(\omega t)]$, da cui possiamo ottenere $\dot{p_d}(t)$ e $\ddot{p_d}(t)$ che serviranno per impostare la legge di controllo nominale. Questa traiettoria di riferimento ellittica diventa banalmente una traiettoria circolare impostando $a = b$.
Ora bisogna risolvere l'equazione differenziale che governa la dinamica del sistema e lo possiamo fare utilizzando la funzione *ode45* di Matlab che ci aiuta a risolvere la funzione ***systemControl()*** da noi definita.

La funzione ***systemControl()*** differisce a seconda della tecnica che decidiamo di adottare.
Nel caso QP, la parte cruciale è nell'impostare il problema di ottimizzazione. Prima di tutto definiamo il controllo nominale u^* e in seguito calcoliamo la distanza tra il robot e i vari ostacoli, scegliendo l'ostacolo che ha distanza minore dal robot. Questo ostacolo sarà identificato come *closest*. A questo punto implementiamo il problema di ottimizzazione utilizzando la funzione Matlab *quadprog*. La matrice E rappresenta la matrice dei vincoli di disuguaglianza, la matrice L è invece la matrice dei termini noti dei vincoli, mentre $|u - u^*|^2$ è la funzione obiettivo del

problema di ottimizzazione. Di seguito è riportata la sezione di codice della funzione ***systemControl()*** dove scegliamo l'ostacolo più vicino al robot e impostiamo il problema di ottimizzazione. Nel codice la posizione del robot è rappresentata dalla variabile *posR*.

```
uNominal = yRef2Dot + kd*(yRefDot - velR) + kp*(yRef - posR);
d = zeros(1, numObs);
for i=1:numObs
       d(1,i) = (Obstacles(:,i) - posR)'*(Obstacles(:,i) - posR);
end
index = find(d == min(d));
closest = Obstacles(:, index);

%Quadratic Programming
distResidua = posR - closest ;
I2 = eye(2);
E = mu*distResidua';
L = (2+alpha*mu)*(distResidua'*velR) + mu*(velR'*velR) + alpha*(
    distResidua'*distResidua) - 2*alpha*delta ;
options = optimset('Display', 'off');
u = quadprog(I2, -2*uNominal, -E,L,[],[],[],[],[], options);
```

Listing 2.1. Implementazione Matlab, Quadratic Programming, Robot puntiforme

Nel caso Switching Programming, la parte cruciale è nell'impostare la CBF e utilizzarla all'interno del controllore dinamico. Anche in questo caso definiamo la legge di controllo nominale u^* e troviamo l'ostacolo *closest*. Ora definiamo gli operatori di proiezione e in seguito definiamo le due CBF di nostro interesse.
Definiamo h_1, che è la CBF che definisce l'insieme D_{u^*}, in questo modo: $h_1 = (p - \overline{p})^T(\mu u^* + 2\dot{p})$. Definiamo invece h_2, che è la CBF che definisce l'insieme D_{δ_1}, come nell'equazione 2.3. La logica del controllore dinamico è molto semplice e riflette ciò che è stato definito nel sistema 2.6.
Di seguito è riportata la sezione di codice della funzione ***systemControl()*** dove scegliamo l'ostacolo più vicino al robot e implementiamo la logica dello switching. Nel codice la posizione del robot è rappresentata dalla variabile posR.

```
uNominal = yRef2Dot + kd*(yRefDot - velR) + kp*(yRef - posR);
d = zeros(1, numObs);
for i=1:numObs
    d(1,i) = (Obstacles(:,i) - posR)'*(Obstacles(:,i) - posR);
end
index = find(d == min(d));
closest = Obstacles(:, index);

%Definition of Projection Operators
po = ((posR-closest)*((((posR-closest)')*(posR-closest))^(-1))*(
    posR-closest)');
poPerp = eye(2) - po;

%Definition of CBF
h1 = ((posR- closest)' * (mu*uNominal + 2*velR));
h2 = ((posR-closest)' * (posR-closest)) + mu*((posR-closest)' *
    velR);

%Switching
if h1>0 || h2>delta1
    u = uNominal;
```

```
20 else
21     u = ((-2/mu)*(po*velR)) + (poPerp*uNominal);
22 end
```

Listing 2.2. Implementazione Matlab, Switching Programming, Robot puntiforme

2.5 Robot di raggio RR

A questo punto possiamo estendere la trattazione al caso di un un robot mobile di raggio RR non fissato che si muove in uno spazio 2D in presenza di uno o più ostacoli. Inoltre assumiamo anche in questo caso che gli ostacoli non possono essere sovrapposti e che il robot parta da una zona safe.
La parte iniziale del codice Matlab resta invariata anche per il caso di robot di raggio RR, inclusa la scelta dei vari parametri fissati. Cambia invece la parte di plot dato che in questo caso bisogna graficare anche i bordi del robot.
Nella funzione ***systemControl()*** invece ci sono delle variazioni evidenti da applicare. In questo caso, ci sono due problemi principali da risolvere:

- bisogna tener conto dei bordi del robot e che questi ultimi non devono urtare gli ostacoli;
- nel caso in cui la traiettoria di riferimento passasse tra due ostacoli, il robot deve riuscire a capire se è in grado di passare tra gli ostacoli o deve aggirarli diversamente.

Per risolvere il primo probema è sufficiente inserire sia per il controllore QP che per il controllore SP un termine correttivo quando calcoliamo le ditanze tra robot e ostacoli. Termine correttivo che va aggiunto anche nel controllo sulla CBF h_2 per quanto riguarda lo Switching Programming. Questo termine correttivo non è altro che il raggio RR del robot.
Per quanto riguarda la seconda problematica, l'idea è di identificare i due ostacoli più vicini al robot e calcolare la distanza tra di loro. Successivamente, il robot confronta la distanza tra i due ostacoli con il suo raggio e passa tra di loro se è in grado di farlo in maniera safe, altrimenti crea un modello virtuale di un ostacolo che ha centro nel punto medio della distanza tra i due ostacoli e ha un raggio δ calcolato in questo modo $\delta_{new} = 2 * ((distanza_{Ostacolo1-Ostacolo2}/2) + \delta + RR)$.

Nel caso QP, a questo punto basta aggiornare la distanza residua tra robot e "ostacolo fantasma" e calcolare le nuove matrici dei vincoli E e L da passare alla funzione *quadprog*.
Di seguito è riportata la sezione di codice della funzione ***systemControl()*** dove scegliamo l'ostacolo più vicino al robot e impostiamo il problema di ottimizzazione. Nel codice la posizione del robot è rappresentata dalla variabile *posR*. Le coordinate dell'ostacolo fantasma, il raggio dell'ostacolo fantasma e la distanza tra robot e ostacolo fantasma sono definiti rispettivamente dalle variabili *ghost*, *deltaFunc* e *distanzaResidua*,

```
1 %Nominal control input in Dtrack
2 uNominal = yRef2Dot + kd*(yRefDot - velR) + kp*(yRef - posR);
3 %Find two closest Obstacles
4 vettDist = zeros(1, numObs);
5 for i=1:numObs
6     vettDist(1,i) = (Obstacles(:,i) - posR)'*(Obstacles(:,i) -
          posR);
```

```
end
index1 = find(vettDist == min(vettDist));
closest1 = Obstacles(:, index1);

vettDist(index1) = inf;

index2 = find(vettDist == min(vettDist));
closest2 = Obstacles(:, index2);
vettDist(index2) = inf;

%Check the robot passes between the two obstacles
distResidua = posR - closest1;
dObs1Obs2 = sqrt((closest2 - closest1)'*(closest2 - closest1));

if (sqrt(distResidua'*distResidua) < dObs1Obs2+deltaFunc) && (
    dObs1Obs2 <= 2*(RR+deltaFunc))
    ghost = (closest2 + closest1)/2;
    deltaFunc = 2*(dObs1Obs2/2+deltaFunc+RR);
    distResidua = posR - ghost;
end
%Quadratic Programming using CBF
I2 = eye(2);
E = mu*distResidua';
L = (2+alpha*mu)*(distResidua'*velR) + mu*(velR'*velR) + alpha*(
    distResidua'*distResidua-RR^2) - 2*alpha*(2*(deltaFunc+RR));
options = optimset('Display', 'off');
u = quadprog(I2, -2*uNominal, -E,L,[],[],[],[],[], options);
```

Listing 2.3. Implementazione Matlab, Quadratic Programming, Robot di raggio RR

Nel caso Switching Programming definiamo due parametri correttivi $\beta = 4$ e $\gamma = 12$ che modificano rispettivamente l'espressione che calcola di quanto il robot può avvicinarsi all'ostacolo virtuale e il controllo sulla CBF h_2.
A questo punto calcoliamo nuovamente i due operatori di proiezione e le due CBF utilizzando le coordinate dell'ostacolo virtuale. Infine come per il caso del robot puntiforme implementiamo la logica del controllore dinamico.
Di seguito è riportata la sezione di codice della funzione ***systemControl()*** per implementare lo Switching Programming nel caso di robot di raggio RR. Nel codice la posizione del robot è rappresentata dalla variabile *posR*. Le coordinate dell'ostacolo fantasma, il raggio dell'ostacolo fantasma e la distanza tra robot e ostacolo fantasma sono definiti rispettivamente dalle variabili *ghost*, *deltaFunc* e *distanzaResidua*.

```
%Nominal control input in Dtrack
uNominal = yRef2Dot + kd*(yRefDot - velR) + kp*(yRef - posR);
%Find two closest Obstacles
vettDist = zeros(1, numObs);
for i=1:numObs
    vettDist(1,i) = (Obstacles(:,i) - posR)'*(Obstacles(:,i) -
        posR);
end
index1 = find(vettDist == min(vettDist));
closest1 = Obstacles(:, index1);
vettDist(index1) = inf;
index2 = find(vettDist == min(vettDist));
closest2 = Obstacles(:, index2);
vettDist(index2) = inf;
%Check the robot passes between the two obstacles
beta = 4;
gamma = 12;
distResidua = posR - closest1;
dObs1Obs2 = sqrt((closest2 - closest1)'*(closest2 - closest1));
```

```
ghost = closest1;
if (sqrt(distResidua'*distResidua) < dObs1Obs2+deltaFunc) && (
    dObs1Obs2 <= 2*(RR+deltaFunc))
    ghost = (closest2 + closest1)/2;
    deltaFunc1 = beta*(dObs1Obs2/2+deltaFunc1+RR);
end
%Definition of Projection Operators
po = ((posR-ghost)*((((posR-ghost)')*(posR-ghost))^(-1))*(posR-
    ghost)');
poPerp = eye(2) - po;
%Definition of CBF
h1 = ((posR- ghost)' * (mu*uNominal + 2*velR));
h2 = ((posR-ghost)' * (posR-ghost)) + mu*((posR-ghost)' * velR);
%Switching
if h1>0 || h2>deltaFunc1+(gamma*RR)
    u = uNominal;
else
    u = ((-2/mu)*(po*velR)) + (poPerp*uNominal);
end
```

Listing 2.4. Implementazione Matlab, Switching Programming, Robot di raggio RR

Capitolo 3

Osservazioni finali

In questo capitolo prenderemo in esame i risultati delle varie simulazioni svolte confrontando il modo in cui i due controllori gestiscono i vari scenari. Alcuni scenari prevedono una traiettoria di riferimento ellittica per il robot mobile e altri una traiettoria circolare. É importante anticipare come entrambi i controllori eseguono in maniera pressochè ideale il proprio task di mantenere il robot in una zona safe e inoltre l'input di controllo permette al robot di allontanarsi di poco dalla traiettoria di riferimento.

3.1 Confronto delle simulazioni per robot mobile puntiforme

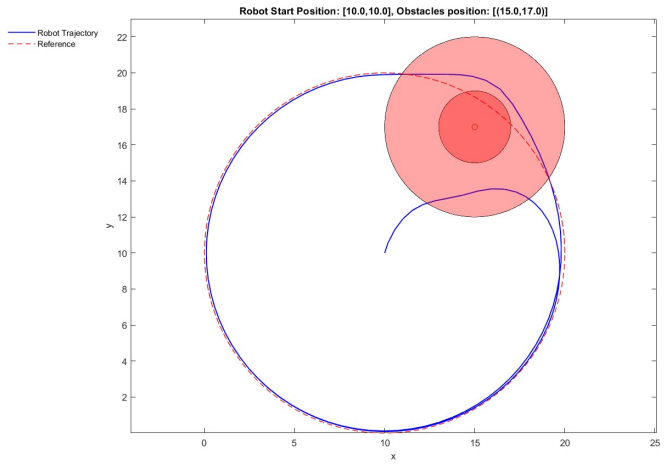

Figura 3.1. QP - scenario 1, robot puntiforme

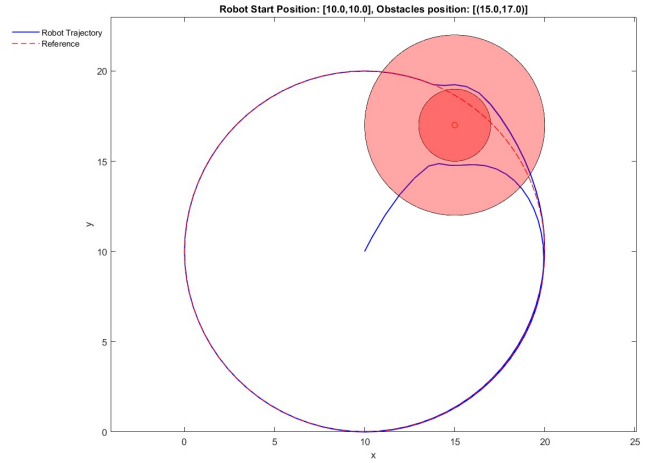

Figura 3.2. SP - scenario 1, robot puntiforme

In questo primo scenario entrambi i controllori utilizzano lo stesso approccio per evitare l'ostacolo, in particolare il controllore basato su **SP** permette al robot di passare molto più vicino all'ostacolo, percorrendo meno strada.

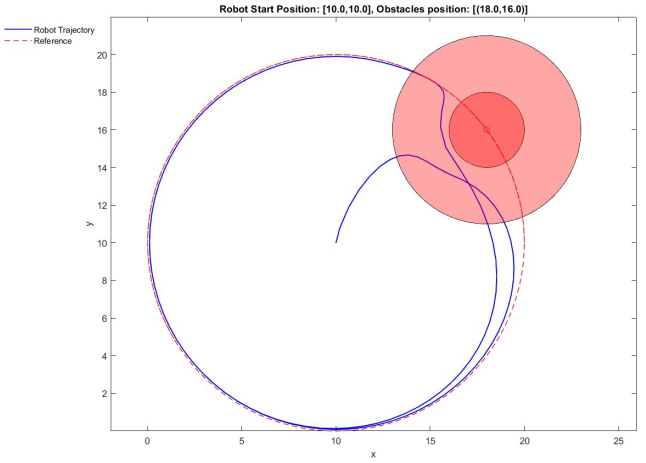

Figura 3.3. QP - scenario 2, robot puntiforme

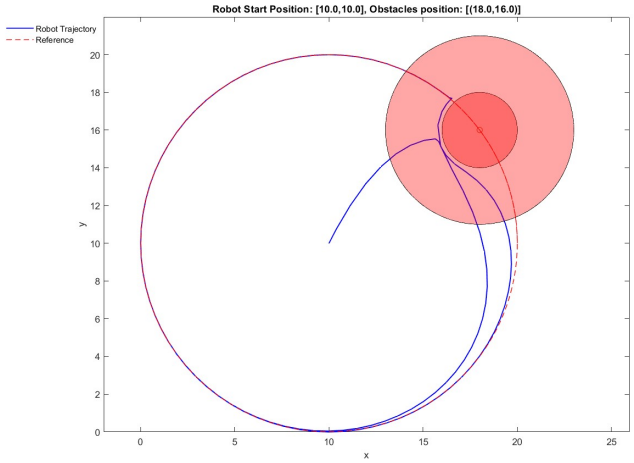

Figura 3.4. SP - scenario 2, robot puntiforme

Anche nel secondo scenario il risultato è lo stesso, con una maggiore precisione del robot controllato mediante **SP**.

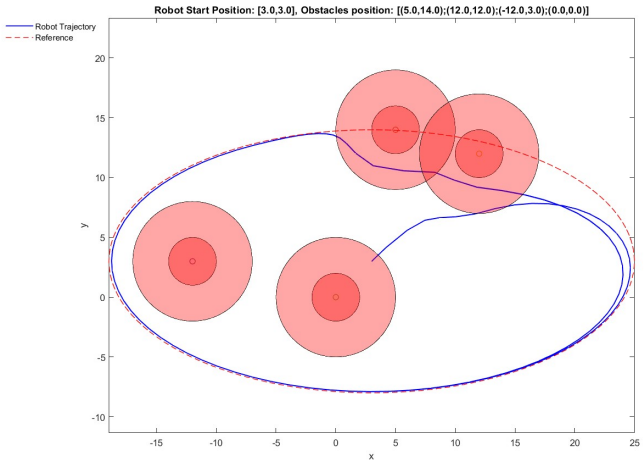

Figura 3.5. QP - scenario 3, robot puntiforme

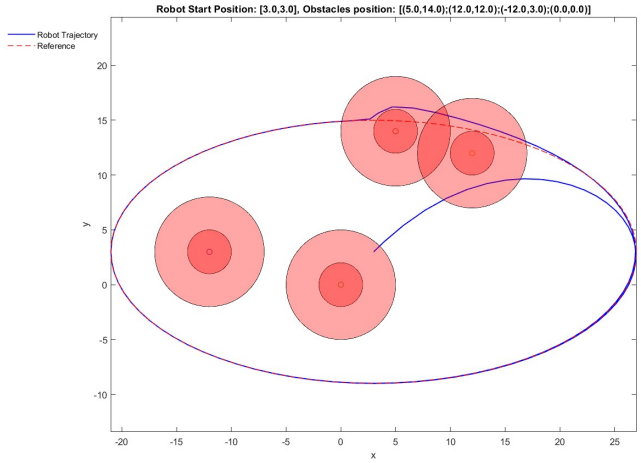

Figura 3.6. SP - scenario 3, robot puntiforme

Per lo scenario numero 3, i due controllori si comportano in modo abbastanza differente tra di loro. Il controllore basato sul **QP** da al robot il comando di aggirare gli ostacoli dal basso anche se in questo modo si allontana abbastanza dal riferimento, mentre il controllo mediante lo **SP** resta il più fedele possibile alla traiettoria di riferimento.

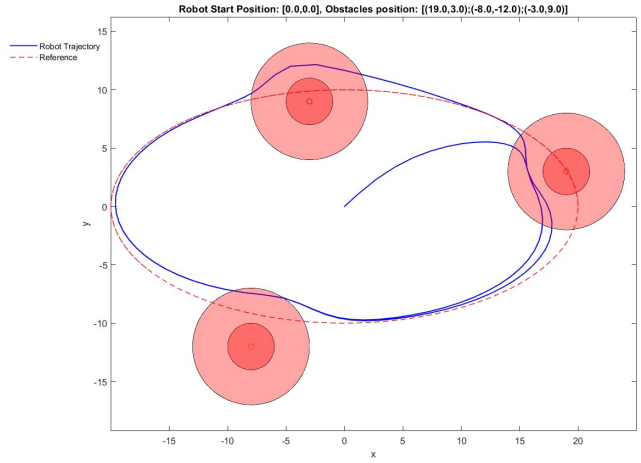

Figura 3.7. QP - scenario 4, robot puntiforme

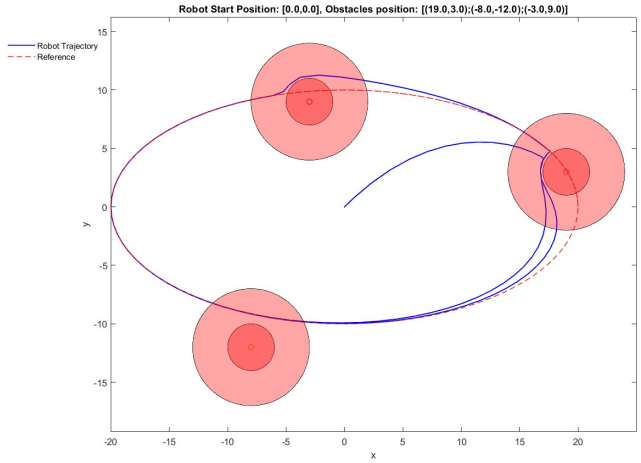

Figura 3.8. SP - scenario 4, robot puntiforme

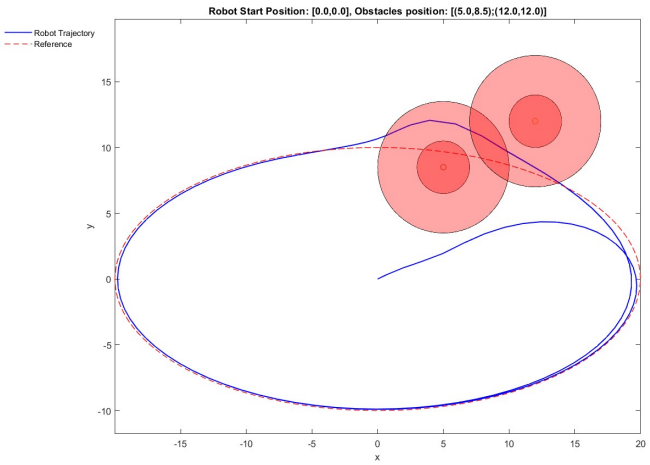

Figura 3.9. QP - scenario 5, robot puntiforme

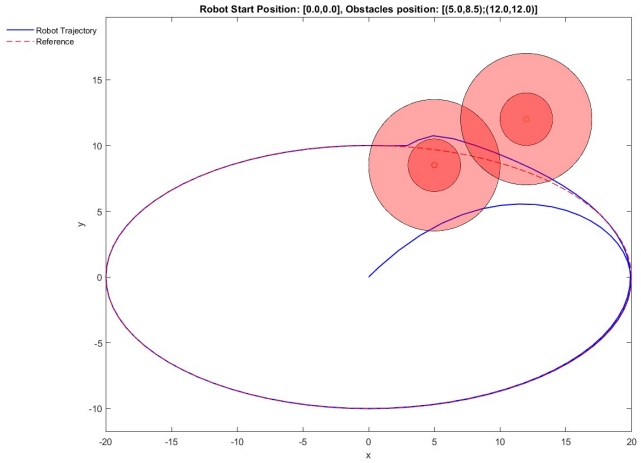

Figura 3.10. SP - scenario 5, robot puntiforme

Negli scenari numero 4 e numero 5, la situazione dei primi due scenari si ripete, con il robot che approccia gli ostacoli in maniera identica con entrambe le tecniche, ma con la traiettoria del robot controllato mediante **SP** sempre più vicina alla traiettoria ideale.

3.2 Confronto delle simulazioni per robot mobile di raggio RR

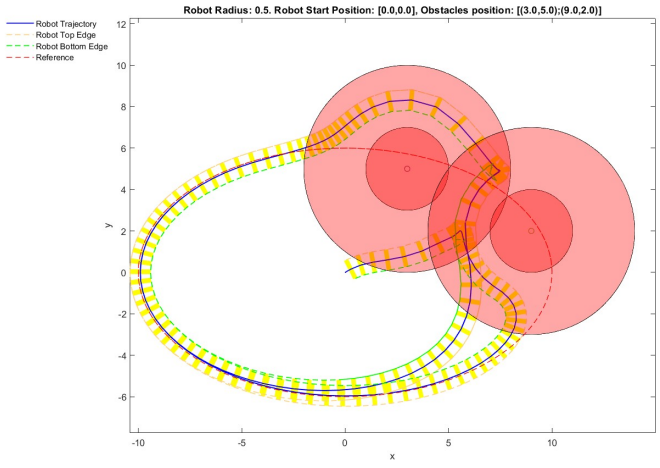

Figura 3.11. QP - scenario 1, robot di raggio 0.5

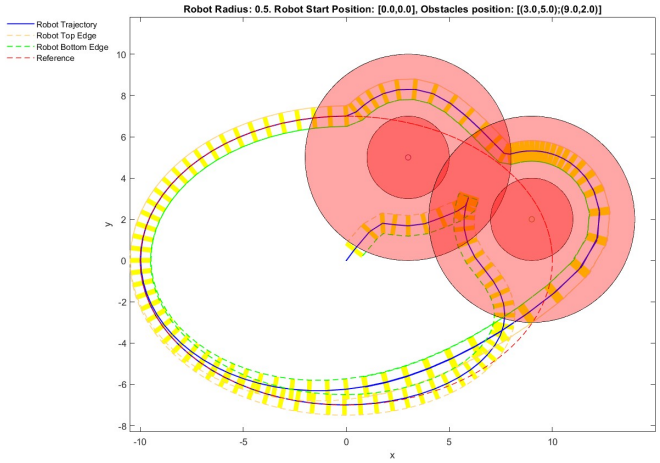

Figura 3.12. SP - scenario 1, robot di raggio 0.5

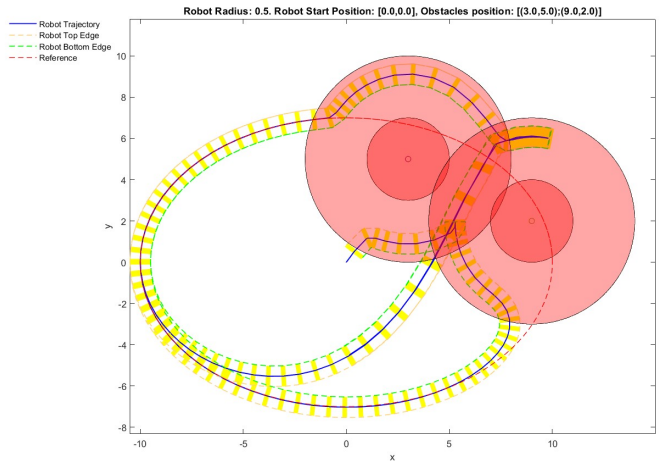

Figura 3.13. SP - scenario 1, robot di raggio 0.5

Nel primo scenario notiamo subito una differenza sostanziale d'approccio. Il robot controllato mediante **QP**, si rende subito conto della possibilità di passare tra i due ostacoli senza problemi e infatti li aggira in questo modo, mentre il robot controllato mediante **SP** (fig:3.12) ha un comportamento strano. In prima istanza sembrerebbe voler passare tra i due ostacoli, ma successivamente ha un cambio di approccio e crea l'ostacolo virtuale e lo aggira. Questa differenza potrebbe essere causata dal settaggio dei due parametri correttivi β e γ che durante la computazione, condizionano gli input del controllore. Infatti, raddoppiando i valori di questi due paramentri il robot (fig:3.13), nonostante un momento di indecisione (in questo caso stava per utilizzare l'approccio dell'ostacolo fantasma), decide giustamente di passare tra i due ostacoli.

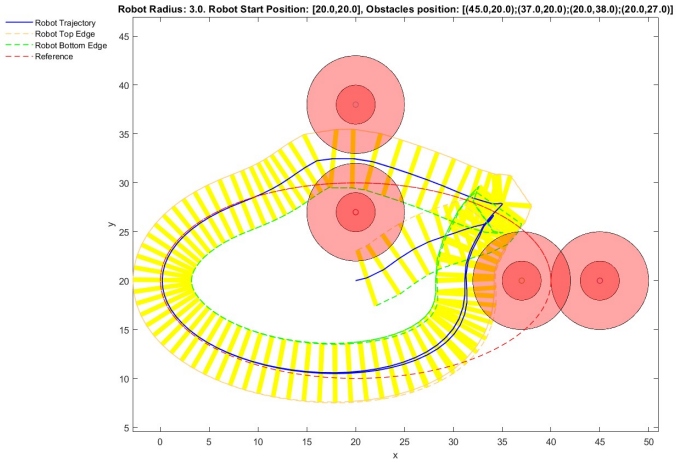

Figura 3.14. QP - scenario 2, robot di raggio 3

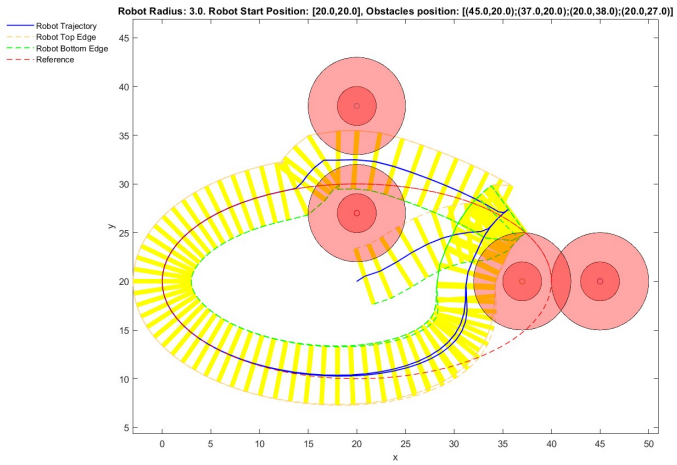

Figura 3.15. SP - scenario 2, robot di raggio 3

Questo è il primo scenario più sfidante per il robot mobile, in quanto sono presenti due coppie di ostacoli vicini tra loro: in un caso il robot, inseguendo la traiettoria di riferimento potrebbe passarci in mezzo, mentre nell'altro a causa del suo raggio, passando tra di loro li urterebbe. Entrambi i controllori riescono a gestire in maniera impeccabile la situazione con il controllore basato su **QP** che fa allontanare un pò di più il robot dal riferimento.

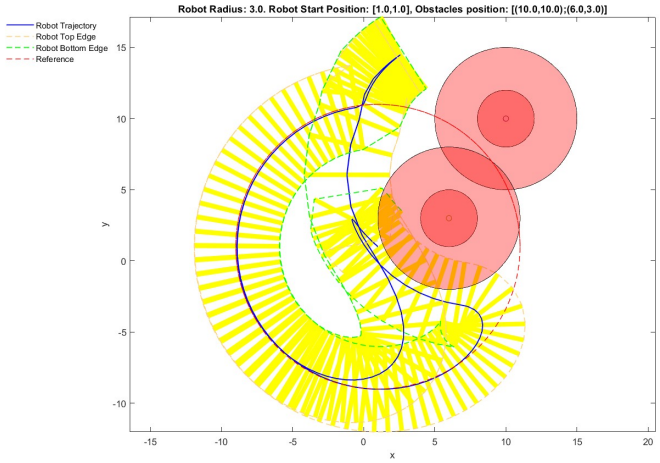

Figura 3.16. QP - scenario 3, robot di raggio 3

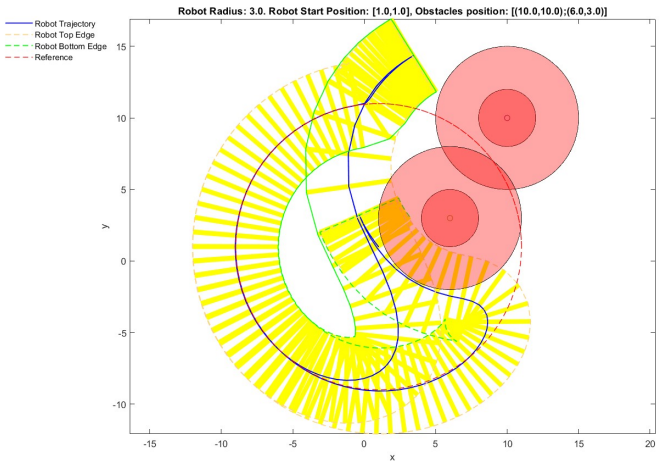

Figura 3.17. SP - scenario 3, robot di raggio 3

In questo caso la traiettoria di riferimento passa esattamente tra i due ostacoli, ma il raggio del robot che anche in questo caso è di 3, non gli permette di passare tra essi e con entrambe le tecniche si comporta in modo safe.

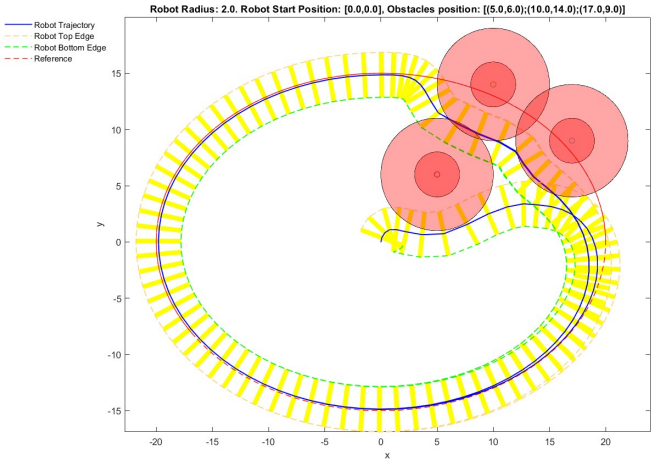

Figura 3.18. QP - scenario 4, robot di raggio 2

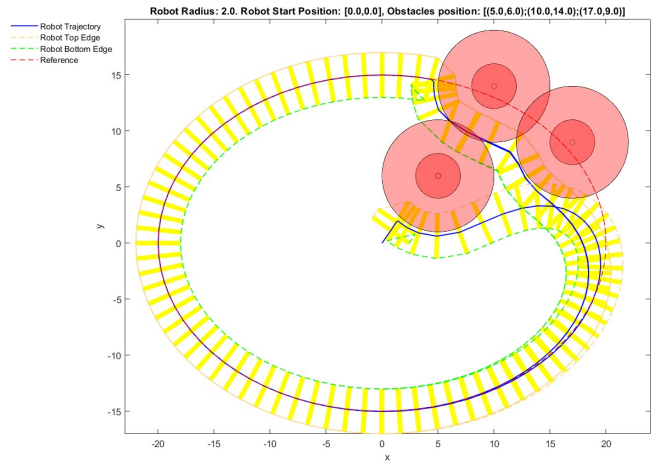

Figura 3.19. SP - scenario 4, robot di raggio 2

Nello scenario 4 sono presenti 3 ostacoli molto ravvicinati tra loro e il robot in entrambi i casi, inseguendo una traiettoria ellittica riesce ad evitarli con un errore rispetto al riferimento molto basso e senza incertezze.

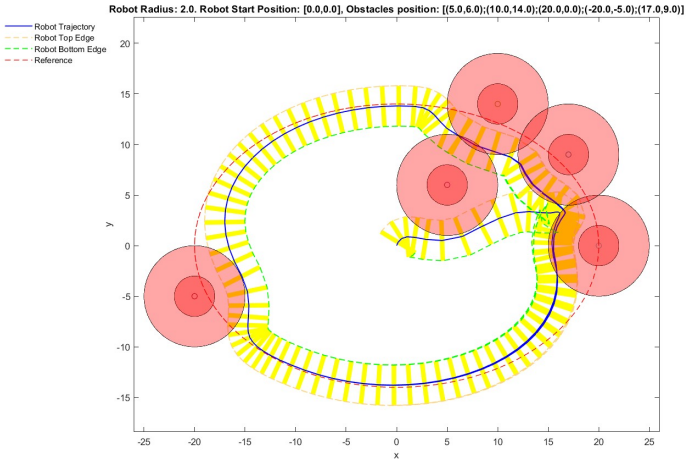

Figura 3.20. QP - scenario 5, robot di raggio 2

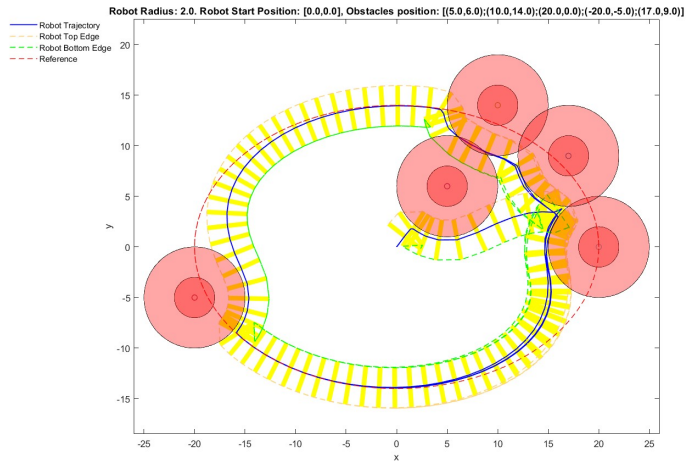

Figura 3.21. SP - scenario 5, robot di raggio 2

La sfida finale del robot mobile è di superare indenne lo scenario 5, popolato da 5 ostacoli, tra cui 4 molto vicini tra loro che costringono il controllore a dare continuamente input correttivi al robot. Anche in questo caso però il robot esegue correttamente e in sicurezza il task disegnando una traiettoria un pò più "spigolosa" e di conseguenza precisa nello scontornare gli ostacoli con il controllore basato su **SP**.
Nonostante gli ultimi due scenari siano molto challenging per il robot mobile, quest'ultimo si comporta in modo impeccabile.

3.3 Conclusioni

In conclusione, i due controllori sono molto efficaci sia nel caso semplificato di robot puntiforme e sia nel caso di robot di raggio RR. Inoltre, quasi in ogni scenario i due controllori si comportano in modo identico, fatta eccezione per lo scenario 3 (Robot puntiforme) in cui le due leggi di controllo portano il robot ad approcciare l'ostacolo in modo opposto. Nello scenario 1 (Robot di raggio 0.5) invece la differenza è dettata dal settaggio dei parametri, infatti cambiandoli, le differenze tra **SP** e **QP** si annullano.
Una differenza si nota nel tempo di esecuzione della simulazione che è leggermente più alto per il controllore basato su **SP**, che nel momento in cui il robot si avvicina agli ostacoli, rende la computazione più lenta. Ciò accade probabilmente a causa dei calcoli maggiori che svolge il controllore **SP**. Nel caso **QP**, la maggior parte dei calcoli vengono svolti all'interno della funzione *quadrprog* di Matlab che ovviamente riesce a svolgerli in modo tale da ottimizzare il costo computazionale dello script.

Bibliografia

[1] Ames, A. D., Coogan, S., Egerstedt, M., Notomista, G., Sreenath, K., & Tabuada, P. (2019, June). Control barrier functions: Theory and applications. In 2019 18th European control conference (ECC) (pp. 3420-3431). IEEE.

[2] Cristofaro, A., Ferro, M., & Vendittelli, M. (2022, December). Safe trajectory tracking using closed-form controllers based on control barrier functions. In 2022 IEEE 61st Conference on Decision and Control (CDC) (pp. 3329-3334). IEEE.

[3] Xu, P., Wang, N., Dai, S. L., & Zuo, L. (2021). Motion planning for mobile robot with modified BIT* and MPC. Applied Sciences, 11(1), 426.

Ringraziamenti

Al termine di questo percorso, vorrei ringraziare tutte le persone che mi sono state accanto e che hanno reso questi ultimi tre anni, molto importanti per me, ancora più speciali.

Innanzitutto, vorrei ringraziare i miei genitori e mia sorella, sempre presenti e sempre attenti a ciò che stavo facendo. Li ringrazio per avermi supportato sia economicamente, ma soprattutto moralmente, in questi anni da fuori sede. Mi dispiace per tutte le volte che vi ho fatto preoccupare perché troppo preso dagli esami. E grazie Monica per esserti presa del tempo per aiutarmi con le pratiche burocratiche!

Ringrazio tutti i miei zii e i miei cugini perché mi sono stati sempre vicino e hanno sempre mostrato interesse e attenzione in quel che ho fatto, dimostrandomi continuamente il loro affetto.

Ringrazio i miei nonni, anche loro sempre presenti e affettuosi nei miei confronti. Sono stati sempre desiderosi di capire come stesse andando e anche loro hanno atteso impazientemente, come me, questo momento così importante. Spero che questo traguardo li renda felici e orgogliosi di me.

Ringrazio tutti i miei amici dell'università, con i quali ho trascorso tre anni fantastici. Grazie a loro sono riuscito a conciliare lo studio e il divertimento, il che mi è stato di grande aiuto.

Ringrazio tutti gli amici di Vaglio che vivono a Roma, con i quali ho potuto creare una piccola famiglia: presenti sia nei momenti belli che in quelli meno felici. Grazie per le infinite serate che ci hanno fatto divertire, ma grazie anche delle intere giornate di studio.
Allo stesso modo ringrazio tutti gli altri miei amici che, pur vivendo lontani, sono riusciti a farmi sentire la loro vicinanza. Sono felice che siamo riusciti a mantenere un rapporto genuino e intenso, nonostante la distanza.

Infine, vorrei ringraziare in modo particolare Chiara. Ha dovuto sopportare tutte le mie continue lamentele e i miei capricci quando qualcosa non andava bene con lo studio. La ringrazio per avermi supportato in tutte le mie scelte e per esserci stata sempre, nonostante i chilometri fisici che quasi costantemente ci hanno separato. Grazie per avermi spronato quando non riuscivo a concentrarmi e per aver condiviso attivamente con me ogni minima gioia e ogni minima delusione di questo percorso. La sua presenza è stata un tassello fondamentale per me.

Questo primo e piccolo traguardo lo dedico a tutti voi e a tutti coloro che mi sono stati vicino e che non ho nominato esplicitamente!