SAPIENZA
UNIVERSITÀ DI ROMA

Inseguimento di traiettorie per robot mobili con vincoli di sicurezza imposti tramite Control Barrier Functions: implementazione via Quadratic Programming

Facoltà: Ingegneria dell'informazione, informatica e statistica

Corso di Laurea in Ingegneria Informatica e Automatica

Candidato

Matteo Scuderi
Matricola 1937090

Relatore

Prof. Andrea Cristofaro

Correlatore

Anno Accademico 2022/2023

Tesi non ancora discussa

Inseguimento di traiettorie per robot mobili con vincoli di sicurezza imposti tramite Control Barrier Functions: implementazione via Quadratic Programming
Tesi di Laurea. Sapienza – Università di Roma

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

Email dell'autore: matteo.scuderi.ms@gmail.com

Sommario

Il presente lavoro di tesi ha lo scopo di formulare, analizzare e studiare il problema del "trajectory tracking" per robot mobili, in presenza di vincoli di sicurezza.

Lo studio riportato si limita a robot omni-direzionali in moto in uno spazio bidimensionale, ma è estendibile a spazi tridimensionali e/o a robot con vincoli cinematici.

Partendo da uno scenario ideale che vede il robot come oggetto puntiforme e con un unico vincolo di sicurezza (rappresentato da un singolo ostacolo), si procederà ad analizzare il problema e a fornirne una possibile soluzione. Successivamente, verranno eliminate entrambe le ipotesi semplificative appena descritte, avvicinando lo scenario simulativo ad un contesto più simile a quello reale.

Lo studio è fondamentalmente composto dai sei punti di seguito riportati:

1. Formulazione di un modello del sistema
2. Imposizione della riproduzione esatta di una traiettoria di riferimento
3. Imposizione del rispetto dei vincoli di sicurezza tramite **Control Barrier Function**
4. Estensione a scenario multi-ostacolo
5. Estensione a robot non puntiforme
6. Estensione a scenario con saturazione degli ingressi con ostacoli dinamici

Questi verranno trattati progressivamente e sequenzialmente nei Capitoli 2, 3 e 4.

Nel Capitolo 1 verranno trattati gli aspetti preliminari e di approccio al problema.

Nel Capitolo 2 si procede con la formulazione matematica del problema.

Nel Capitolo 3 si descrivono le scelte implementative.

Nel Capitolo 4 si discutono e commentano i risultati simulativi ottenuti.

Nel Capitolo 5 si traggono le conclusioni sulla metodologia di controllo proposta.

Indice

Capitolo 1

Introduzione

1.1 Introduzione al Problema

Lo scopo di questo studio è, in estrema sintesi, il seguente: dato un sistema dinamico con vincoli di sicurezza, mantenere l'evoluzione del sistema al di fuori di quelle regioni dello spazio di stato ritenute critiche, mentre si insegue una traiettoria di riferimento.

Il problema da affrontare si ripresenta in diverse aree della robotica mobile. Assegnata una traiettoria di riferimento ad un robot, potrebbero presentarsi ostacoli non conosciuti a priori, che vanno evitati nel rispetto dei vincoli di sicurezza.

Il metodo di soluzione proposto non è tuttavia ristretto al ***"collision avoidance"***, ma va a garantire più in generale l'evoluzione al di fuori delle zone *"unsafe"* dello spazio di stato.
È quindi possibile (tramite un'opportuna **Control Barrier Function (CBF)**), andare a imporre dei vincoli su qualsiasi componente dello spazio di stato del sistema preso in considerazione (limitazione della velocità, assetto, temperatura etc.).

Nel Paragrafo 1.2 definiremo il concetto e la metodolologia applicativa delle CBF che, nel caso di studio in questione, useremo per garantire *collision avoidance*.

1.2 Control Barrier Functions

Le Control Barrier Functions [1] (d'ora in avanti **CBF**), sono lo strumento principale utilizzato in questo studio per garantire il rispetto dei vincoli di sicurezza. Dato un sistema dinamico $\dot{x} = f(x) + g(x)u$, con $x \in \mathbb{R}^n$, definiamo C il *Safe-Set* t.c:

$$C \coloneqq \{\ x \in \mathbb{R}^n\ :\ h(x) \geq 0\ \} \tag{1.1}$$

$$con\ h\ :\ \mathbb{R}^n \to \mathbb{R}\ opportunamente\ definita$$

L'idea alla base è dunque quella di forzare lo stato $x(t)$ ad evolvere in maniera tale che: $h[x(t)] \geq 0,\ \forall\ t \geq 0$, ossia non abbandoni mai il *Safe-Set* C
Per fare questo, è necessario imporre la condizione:

$$\dot{h} + \alpha(h) \geq 0 \quad con\ \alpha \in K_\infty$$

1.3 Definizione del Problema

Lo scenario (bidimensionale) che vogliamo modellare e simulare è così articolato: Considerato un robot omni-direzionale (approssimato a punto materiale nel suo centro di massa), assegnata una traiettoria di riferimento, considerato un ostacolo la cui posizione non è nota a priori, vogliamo un **controllore in feedback** che garantisca:

- *Trajectory Tracking*
- *Collision Avoidance*

In particolare chiediamo che, assegnata la posizione $\bar{p}$ dell'ostacolo (unico, almeno per la prima fase dello studio), il robot si mantenga al di fuori della circonferenza di raggio δ con centro in $\bar{p}$.
Questo vincolo deve esser verificato anche qualora l'ostacolo si trovasse sulla traiettoria di riferimento e, una volta aggirato, il robot deve tornare ad inseguire il riferimento.

Una volta soddisfatte queste specifiche, estenderemo il caso di studio dapprima a uno scenario multi-ostacolo, per poi rimuovere anche l'approssimazione del robot a oggetto puntiforme.

Capitolo 2

Formulazione Matematica

2.1 Definizione del Modello del Sistema

Il modello del sistema che consideriamo è un **doppio integratore** del tipo:

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \end{pmatrix} = \begin{pmatrix} \tau_x \\ \tau_y \end{pmatrix} \tag{2.1}$$

i cui ingressi sono le accelerazioni lungo x ed y.

Riscrivendo il sistema con: $p = \begin{pmatrix} x \\ y \end{pmatrix}$; $\dot{p} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix}$; $z = \begin{pmatrix} p \\ \dot{p} \end{pmatrix}$; $u = \begin{pmatrix} u_x \\ u_y \end{pmatrix}$;

$$\dot{z} = \begin{pmatrix} 0 & I \\ 0 & 0 \end{pmatrix} z \; + \; \begin{pmatrix} 0 \\ I \end{pmatrix} u = Az + Bu \tag{2.2}$$

con $z \in \mathbb{R}^4$, $\;$ *Blocchi Matriciali*$\{0; I\} \in \mathbb{R}^{2\times 2}$

2.2 Controllo per Inseguimento di Traiettoria

Per garantire la riproduzione esatta della traiettoria di riferimento assegnata p_d, utilizziamo un controllore con feed-back e feed-forward (fig. 1.1)

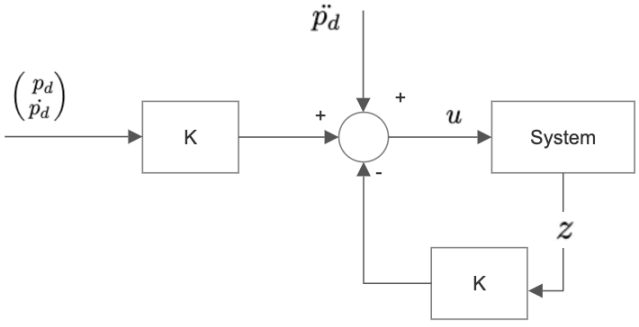

Figura 2.1. Schema a Blocchi del Controllore

Nel sistema ad anello chiuso applichiamo perciò il controllo

$$u^* = u_{ff}(t) + u_{fb}(t)$$

con:
i) $u_{ff}(t) = \ddot{p}_d(t)$

ii) $u_{fb}(t) = K \begin{pmatrix} p_d(t) - p(t) \\ \dot{p_d}(t) - \dot{p}(t) \end{pmatrix} \quad K = \begin{pmatrix} K_p & 0 & K_d & 0 \\ 0 & K_p & 0 & K_d \end{pmatrix}$, da cui:

$$u^* = \ddot{p_d} + Kd(\dot{p_d} - \dot{p}) + Kp(p_d - p) \tag{2.3}$$

Il sistema ad anello chiuso con feed-forward $\dot{z} = (A - BK)z + Bu_{ff}$ riproduce perfettamente la traiettoria, purché si assegnino opportunamente gli autovalori della matrice (A-BK). Si dimostra [2] che è sempre possibile trovare K_p, K_d che rendono il sistema asintoticamente stabile.

2.3 Imposizione dei vincoli tramite CBF

Per definire la regione *safe* dello spazio, consideriamo la CBF candidata:

$$h(x) = h \begin{pmatrix} p \\ \dot{p} \end{pmatrix} = \|p - \bar{p}\|^2 + \mu(p - \bar{p})\dot{p} \tag{2.4}$$

$$\mu > 0 \ : \ \textit{costante abbastanza piccola}$$

con:

- $\|p - \bar{p}\|^2$: termine tanto più positivo al crescere della distaza dall'ostacolo
- $\mu(p - \bar{p})\dot{p}$: contributo positivo in caso di allontanamento dall'ostacolo, negativo altrimenti. Proporzionale alla velocità.

Ricordiamo che il nostro obiettivo è quello di forzare l'evoluzione dello stato x affinché $h(x)$ si mantenga non negativa (ossia il robot rimanga nella zona safe).

Definendo perciò la *safety region* come $\Pi_{safe} = \{(p, \dot{p}) \in \mathbb{R}^{2n} : h(p, \dot{p}) > \delta\}$, dove δ è il termine di *penalty* che dipende dalla *clearance* dall'ostacolo (distanza minima dall'ostacolo per restare in sicurezza).

La nostra h, con l'aggiunta del termine di *clearance*, è una CBF finché vale la disequazione:

$$\dot{h}(t) + \alpha(h(t) - \delta) \geq 0 \tag{2.5}$$

Sviluppando il termine di derivata e chiamando $d = p - \bar{p}$, otteniamo:

$$2d^T\dot{p} + \mu\dot{p}^T\dot{p} + \mu d^T u + \alpha(h - \delta) \geq 0$$

Il nostro obiettivo sarà quello di trovare il controllo u che rispetti questo vincolo e, così facendo, imporremo la CBF $h(x) \geq 0$, i.e. forzeremo lo stato x ad evolvere all'interno della *safety region* Π_{safe}.

Approfondiremo il discorso nella sezione 3.1

Capitolo 3

Implementazione

Analizziamo le scelte implementative per ricavare un controllo modificato, a partire da quello nominale, che rispetti il vincolo della CBF.

3.1 Quadratic Programming

La CBF analizzata nella sezione 2.3, definisce il vincolo essenziale per mantenere il robot nella zona *safe*. L'imposizione di questo vincolo può avvenire tramite diverse tecniche che agiscono sul controllo u.
La metodologia proposta in questa dissertazione è quella del Quadratic Programming (QP).

Riprendiamo il controllo nominale introdotto nella sezione 2.2 (equazione 2.3): $u^* = \ddot{p_d} + kd(\dot{p_d} - \dot{p}) + kp(p_d - p)$ e il vincolo della CBF $\dot{h} + \alpha(h - \delta) \geq 0$.

Impostiamo quindi un problema di QP, o più precisamente di *"Least Squares"*:

$$\arg\min_{u \in \mathbb{R}^2} \|u - u^*\|^2 \quad \text{s.t.} \quad \dot{h} + \alpha(h - \delta) \geq 0 \tag{3.1}$$

Stiamo quindi cercando il controllo u con lo scostamento minimo da u^* tale che lo stato $x(t) \in \Pi_{safe}, \forall t \geq 0$. Se lo stato si trova in una regione non critica, avremo $u \equiv u^*$. Altrimenti, dovremo trovare un controllo u (tramite risoluzione del problema di QP) che forzi il sistema a restare nella *safe region*.

3.2 Risoluzione con Quadprog

Per risolvere il problema di QP appena descritto, ci serviremo della funzione quadprog di Matlab.
Questa prende come input quattro matrici che definiscono i problemi di QP generici (non nella forma *Least Squares*). Essendo i due problemi equivalenti, dovremo quindi ricondurci alla forma generica, calcolare le matrici descrittive del problema e poi fornirle in input alla funzione quadprog.
Partiamo col definire il problema di QP in forma generale:

$$\frac{1}{2} x^T H x + f^T x \quad s.t.\ Ax \leq b \tag{3.2}$$

La formulazione è facilmente riconducibile a quella del *"Least Squares"*:

$$\frac{1}{2}\|Rx - d\|^2 \quad s.t.\ Ax \leq b \tag{3.3}$$

con $H = R^T R$; $\quad f = -R^T d$
Considerando la nostra u come incognita del problema, riscriviamo in forma esplicita la nostra funzione obiettivo e il relativo vincolo:

$$\arg\min_{u \in \mathbb{R}^2} \|u - u^*\|^2 \quad \text{s.t.} \quad 2d^T\dot{p} + \mu\dot{p}^T\dot{p} + \mu d^T u + \alpha(d^T d + \mu d^T\dot{p} - \delta) \geq 0$$

con $d = p - \bar{p}$.

Osservando la forma della funzione obiettivo e con semplici passaggi algebrici, si ricavano le quattro matrici da fornire in input alla funzione quadprog:

$$H^{(q)} = I_{2\times 2}; \quad f^{(q)} = -2u^* = -2(\ddot{p_d} + kd(\dot{p_d} - \dot{p}) + kp(p_d - p));$$
$$A^{(q)} = -\mu d^T; \quad b^{(q)} = (2 + \alpha\mu)(d^T\dot{p}) + \mu\dot{p}^T\dot{p} + \alpha d^T d - \alpha\delta;$$

La soluzione trovata dall'algoritmo risolutivo di quadprog sarà proprio quel controllo che garantisce *collision avoidance* e che minimizza la "distanza" dal controllo nominale u^*, usato per garantire *trajectory tracking*.

Capitolo 4

Simulazione

4.1 Considerazioni preliminari e parametri di simulazione

I risultati di seguito riportati e le relative scelte di implementazione sono frutto dello studio di numerosi scenari simulativi, appositamente creati per testare il funzionamento della metodologia proposta.

La scelta di alcuni parametri è avvenuta perciò in maniera "empirica" con un processo di *trial and error*.
Non si garantisce dunque l'ottimalità dei parametri scelti, ma ne si garantisce l'efficacia nei numerosi contesti simulativi a cui il modello è stato sottoposto.

In tutte le simulazioni, si è usata alternativamente una traiettoria di riferimento circolare o ellittica, ma i risultati sono validi per qualsiasi tipo di traiettoria.

Di seguito si riportano i valori dei parametri di simulazione utilizzati in tutti i contesti dello studio proposto. Ne verranno aggiunti degli altri nelle sezioni successive.

$$\alpha = 5;\ \mu = 0.05;\ \delta = 2;\ t_{span} = 10s;$$
$$K_p = 100;\ K_d = 30;$$
$$R = 10;\ a_x = 20;\ a_y = 10;\ O_x = O_y = 20;\ \omega = \frac{\pi}{2} rad/s;$$

4.2 QP - Robot Puntiforme - Single Obstacle -

In questo primo assetto simulativo considereremo uno scenario ad ostacolo singolo e un robot approssimato a punto materiale.
Come primo passo andiamo a definire la traiettoria da assegnare e la sua derivata, necessaria per il feedforward.

$$p_d = \begin{pmatrix} a_x \sin \omega t + O_x \\ a_y \cos \omega t + O_y \end{pmatrix};\quad \dot{p_d} \begin{pmatrix} a_x \omega \cos \omega t + O_x \\ -a_y \omega \sin \omega t + O_y \end{pmatrix};\quad \ddot{p_d} \begin{pmatrix} -a_x \omega^2 \cos \omega t + O_x \\ -a_y \omega^2 \sin \omega t + O_y \end{pmatrix};$$
$$\textit{Per la traiettoria circolare, basta porre } a_x = a_y = R.$$

A questo punto abbiamo tutti gli strumenti necessari (fissati i parametri come detto alla sezione 4.1) per simulare il comportamento del robot.

In Matlab, simuliamo un sistema dinamico tramite **ode45** con: Matrici A, B (definite nella sez. 2.1), $z = (p, \dot{p}) \in \mathbb{R}^4$ vettore di stato, e infine:

$$u = quadprog(H^q,\ f^q,\ A^q,\ b^q)\ (dalla\ Sezione\ 3.2)$$

Si riportano due figure che testimoniano il corretto funzionamento della legge di controllo del robot.

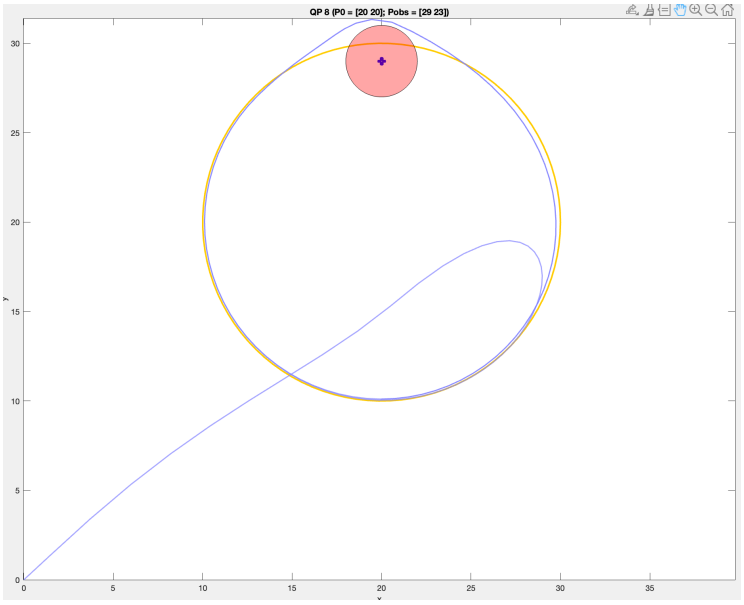

Figura 4.1. Ostacolo in $[20, 29]$

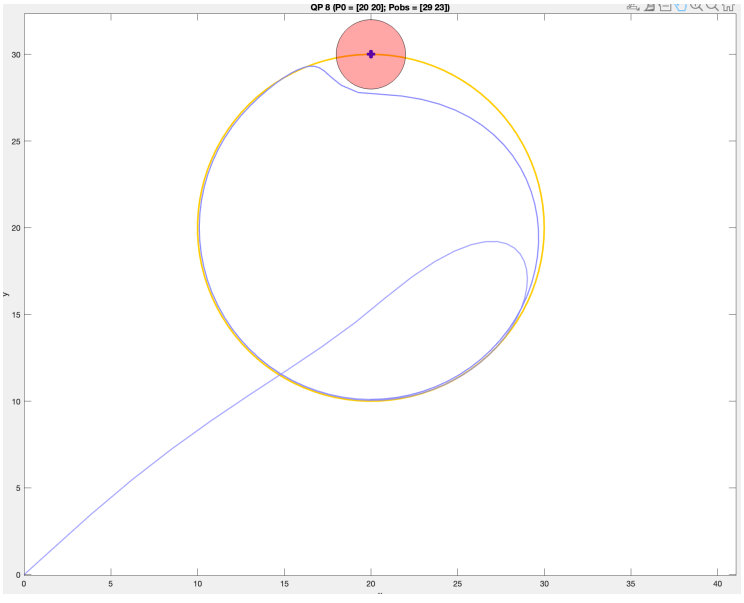

Figura 4.2. Ostacolo in $[20, 30]$

Vale la pena notare come il robot non solo evita l'ostacolo (in rosso), ma sceglie anche la traiettoria più conveniente, ossia impone quel controllo u che sia il più "vicino" possibile al controllo nominale u^*.

Questo comporta, almeno in questo caso, il passare internamente o esternamente all'ostacolo scegliendo il percorso che minimizzi la deviazione dal riferimento.

4.3 QP - Robot Puntiforme - Multiple Obstacles -

Il metodo risolutivo descritto si estende abbastanza facilmente ad uno scenario multi-ostacolo.
Consideriamo una configurazione a m ostacoli, per cui definiamo $\bar{P} \coloneqq \{\ \bar{p}_i \in \mathbb{R}^2, i = 1,\ ..., m\ \}$ e per cui vale sempre la seguente **assunzione di non-sovrapposizione**:

$$\|\bar{p}_i - \bar{p}_j\|^2 > 2\delta\ ,\forall\ (i,j) = 1, ..., m\ ;\ i \neq j$$

Questa assunzione fondamentale ci permette di considerare un solo ostacolo alla volta. Infatti, considerando l'*i-esimo* ostacolo e ignorando gli altri $m-1$, abbiamo la certezza che esistono infinite circonferenze centrate in $\bar{p}_i$ e di raggio $\delta + \epsilon$ su cui il robot può muoversi senza imbattersi in nessuno degli altri ostacoli.

Sviluppiamo allora un algoritmo che calcoli in tempo reale l'ostacolo più vicino (i.e. potenzialmente più prossimo alla collisione) e impostiamo il problema di QP usando le coordinate dell'ostacolo selezionato. Formalmente:

$$i^* = \arg \min_{i=1,...,m} \|p - \bar{p}_i\|^2 \tag{4.1}$$

Il nostro controllo sarà dunque: $u = quadprog(\ H^q,\ f^q,\ A^q(\bar{p}_{i^*}),\ b^q(\bar{p}_{i^*})\ )$.

Anche in questo caso riportiamo due figure per mostrare il corretto comportamento del robot.

Notiamo come il robot si destreggia discretamente sia in scenari in cui gli ostacoli sono sparsi nell'ambiente, sia in presenza di zone ad alta densità.

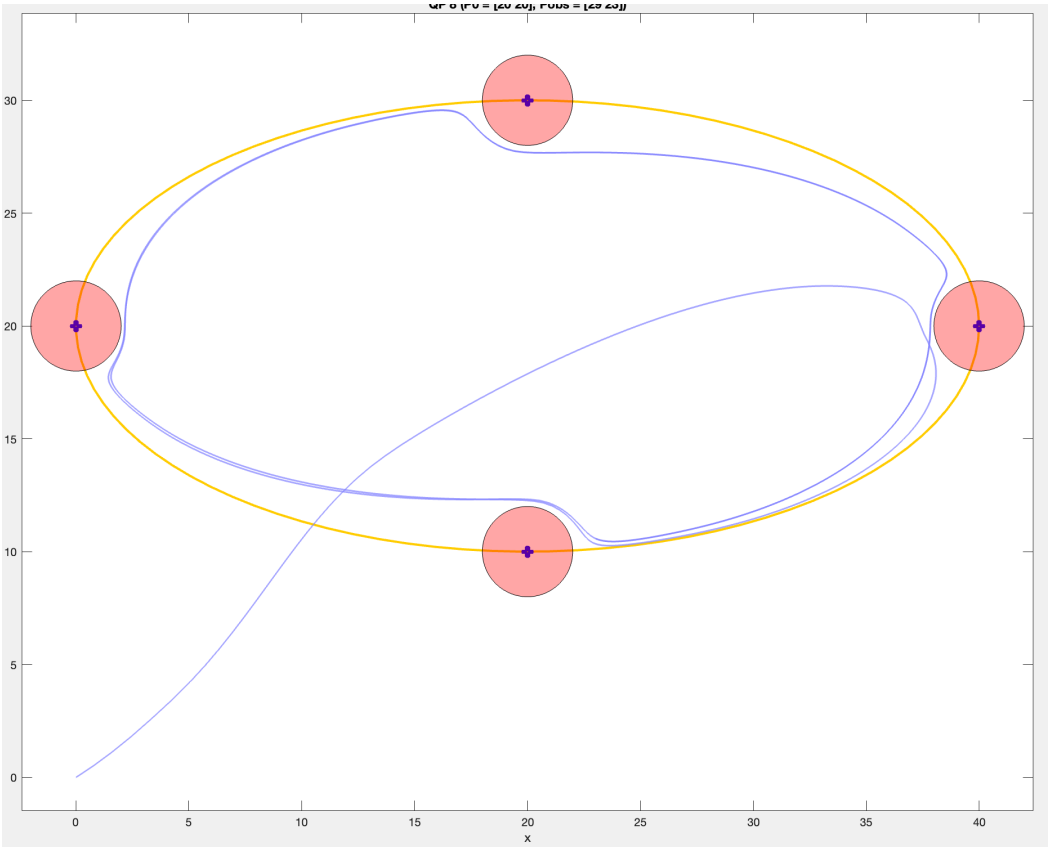

Figura 4.3. Comportamento con Ostacoli Sparsi

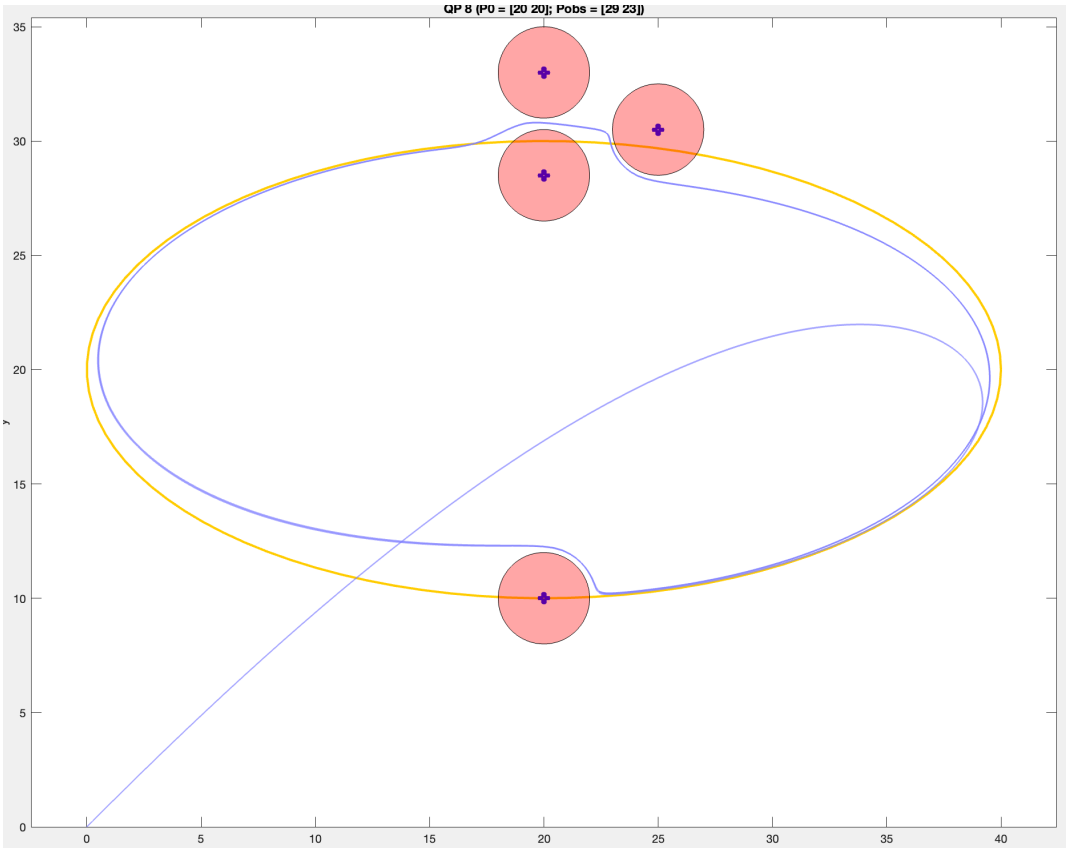

Figura 4.4. Comportamento con Cluster di Ostacoli

4.4 QP - Robot non puntiforme - Multiple Obstacls -

In questa configurazione, considereremo un robot circolare dotato di diametro $2r$, che opera in uno scenario multi-ostacolo.
Dovremo quindi aggiornare la nostra CBF per tenere conto di questo raggio, che considereremo come una *"clearance"* aggiuntiva. Inoltre, saranno necessarie delle accortezze per gestire quelle situazioni in cui lo spazio tra due ostacoli non è sufficiente a far passare il robot, ma di questo discuteremo nel seguito.

4.4.1 Modifica della CBF

Cominciamo col ridisegnare la nostra CBF per tener conto del raggio del robot r che, come già accennato, considereremo come *clearance* aggiuntiva.
Partendo da $\dot{h} + \alpha(h - \delta) \geq 0$, definiamo:

$$\dot{h} + \alpha(h - \gamma(\delta + r)) \geq 0 \qquad \gamma > 0 \tag{4.2}$$

il nuovo vincolo da imporre al nostro sistema. Il termine di penalty γ (costante e positivo) viene applicato alla somma tra raggio del robot e clearance dall'ostacolo. Questo coefficiente, opportunamente scelto, aumenta di fatto il peso del termine di "penalty" $r + \delta$ nella CBF.
Riscrivendo quindi la matrice $b^{(q)}$ da fornire in input alla funzione quadprog, con la modifica della CBF appena descritta, otteniamo:

$$b^{(q)} = (2 + \alpha\mu)(z^T \dot{p}) + \mu \dot{p}^T \dot{p} + \alpha z^T z - \alpha\gamma(r + \delta) \tag{4.3}$$

La necessità di questo termine aggiuntivo si è resa evidente dai risultati simulativi ottenuti. Mantenendo i parametri di simulazione inalterati, con l'aggiunta di $r = 3$; $\gamma = 4$, il robot si comporta egregiamente anche in scenari estremamente *challenging* (vedi fig. 4.5).
D'altro canto, in assenza della correzione γ, il robot entra in collisione in scenari particolarmente densi di ostacoli. È possibile che con un opportuno *re-tuning* dei parametri (α e μ su tutti) si riesca ad aggirare questo problema, ma i risultati ottenuti con γ sono abbastanza soddisfacenti da giustificare l'aggiunta di un parametro.

4.4.2 Ipotesi di *Non-Overlapping*

Va ora affrontato un problema venutosi a creare con la modifica della CBF. Con l'aggiunta del raggio r (inserito come *clearance*, può venire a mancare l'ipotesi di non-sovrapposizione descritta nel paragrafo (vedi sezione 4.3).
La violazione di quel vincolo avviene proprio ogni qualvolta lo spazio tra due ostacoli sia inferiore alla larghezza del robot.
Senza un opportuno intervento, il robot entra in una fase di stallo mentre tenta di "incunearsi" tra i due ostacoli.

La soluzione proposta è quella di modificare l'algoritmo di selezione dell'ostacolo più vicino nel modo seguente:

1. **Modificare selezione degli ostacoli**

$$i^* = \arg \min_{i=1,\dots,m} \|p - \bar{p}_i\|^2$$

$$j^* = \arg \min_{j=1,\dots,m;\ j \neq i^*} \|p - \bar{p}_j\|^2$$

Selezioniamo i due ostacoli più vicini al robot, $\bar{p}_{i^*}$, $\bar{p}_{j^*}$

2. **Controllo sulla distanza**

   Eseguiamo una verifica sulla distanza tra gli ostacoli e su quella dal robot.
   Andiamo quindi a modificare l'azione di controllo a seconda del verificarsi di certe condizioni.
   Tramite due **clausole** $C_1,\ C_2$, che devono essere entrambe verificate, definiamo proprio quelle condizioni che, se sussistenti, metterebbero a rischio la sicurezza del robot.
   Inseriamo quindi una logica del tipo:

   $$\begin{aligned} &IF(C_1\ AND\ C_2)\ \ THEN: && \textit{Applica Controllo Modificato} \\ &\quad ELSE: && \textit{Applica Controllo di Default} \end{aligned}$$

   Per alleggerire la notazione, definiamo:

   - $l \coloneqq \|\bar{p}_{i^*} - \bar{p}_{j^*}\|$ la distanza tra gli ostacoli,
   - $d \coloneqq \|p - \bar{p_{i^*}}\|$ la distanza del robot dall'ostacolo più vicino.

   Possiamo allora a definire

   - $C_1\ :\ \|z\| < l + \delta + 2r$
   - $C_2\ :\ l \leq 2(r + \delta)$

   La prima clausola è verificata quando la distanza tra il robot e l'ostacolo più vicino è minore di una distanza di sicurezza.
   La seconda (e più importante) clausola si attiva ogni qualvolta la distanza tra i due ostacoli più vicini è inferiore alla larghezza del robot.
   Quando entrambe le clausole C_1 e C_2 sono verificate, vuol dire che il vincolo di non sovrapposizione è violato e il robot è prossimo ad entrare in una condizione di stallo. È perciò necessario intervenire con un controllo modificato.

3. **Controllo Modificato**

   Per evitare lo stallo del robot, dobbiamo in qualche modo ripristinare artificialmente la condizione di non sovrapposizione.
   Consideriamo perciò un **ostacolo virtuale** $\bar{p}_v$ centrato nel punto medio tra i due ostacoli $\bar{p}_{i^*}$, $\bar{p}_{j^*}$ e con raggio tale da ricoprire entrambi gli ostacoli.

   $$\bar{p}_v \coloneqq (\bar{p}_{i^*} + \bar{p}_{j^*})/2; \qquad \delta_v \coloneqq l + 2\delta + 2r;$$

   Il nostro controllo modificato sarà perciò:

   $$u = quadprog(\ H^q,\ f^q,\ A^q(\bar{p}_v),\ b^q(\bar{p}_v)\ ), \qquad\qquad con\ \delta = \delta_v.$$

4. **Controllo di Default**

   Nel caso in cui una delle due clausole non sia verificata, possiamo applicare il controllo non modificato considerando l'ostacolo più vicino $\bar{p}_{i^*}$:

   $$u = quadprog(\ H^q,\ f^q,\ A^q(\bar{p}_{i^*}),\ b^q(\bar{p}_{i^*})\ )$$

4.4.3 Tracciamento Traiettorie

Osserviamo ora un paio di risultati simulativi a riprova del corretto funzionamento del nuovo algoritmo implementato, sia per il controllo di default che per quello modificato.

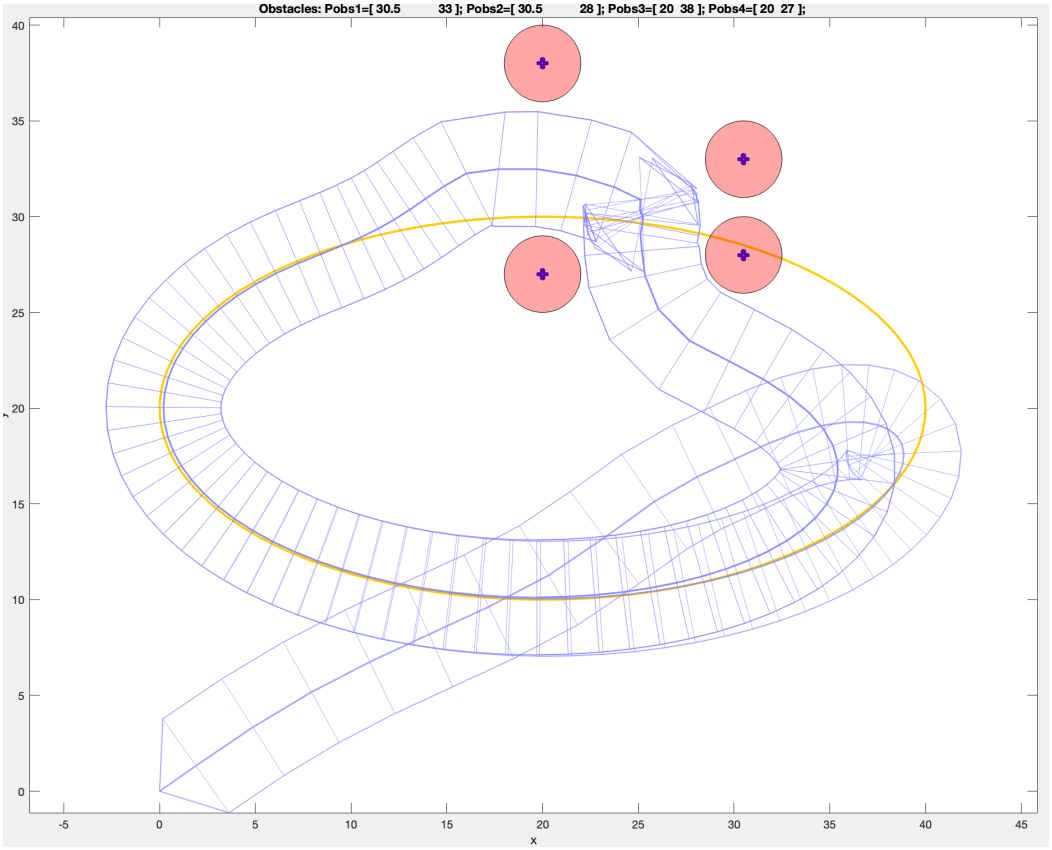

Figura 4.5. Scenario senza Overlapping, molto challenging

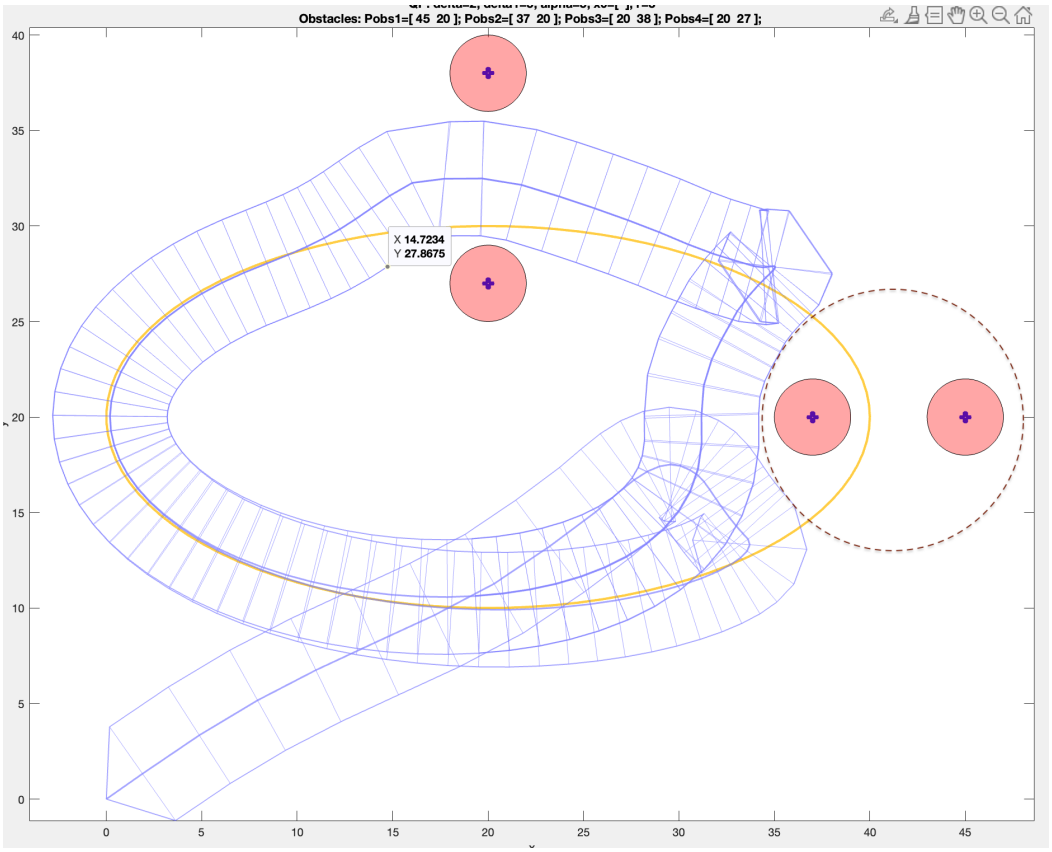

Figura 4.6. Scenario con Overlapping (ostacolo virtuale tratteggiato)

4.5 Simulazioni ulteriori

Prendiamo ora come caso di riferimento lo scenario con robot non puntiforme e ostacoli multipli. Vogliamo testare il nostro modello in situazioni ancora più *challenging*, più vicine ai casi di applicazione reale.
Ancora una volta, complichiamo il problema per step intermedi:

1. Saturazione degli ingressi
2. Ostacoli in movimento
3. Saturazione ed ostacoli mobili

Un video delle simulazioni è consultabile seguendo il link https://youtu.be/M6B1dvNh-hU.

4.5.1 Saturazione degli Ingressi

Osservando gli ingressi in input al nostro sistema (fig. 4.7), notiamo che per recuperare l'offset di traiettoria iniziale, vengono fornite accelerazioni fuori scala per la nostra simulazione.

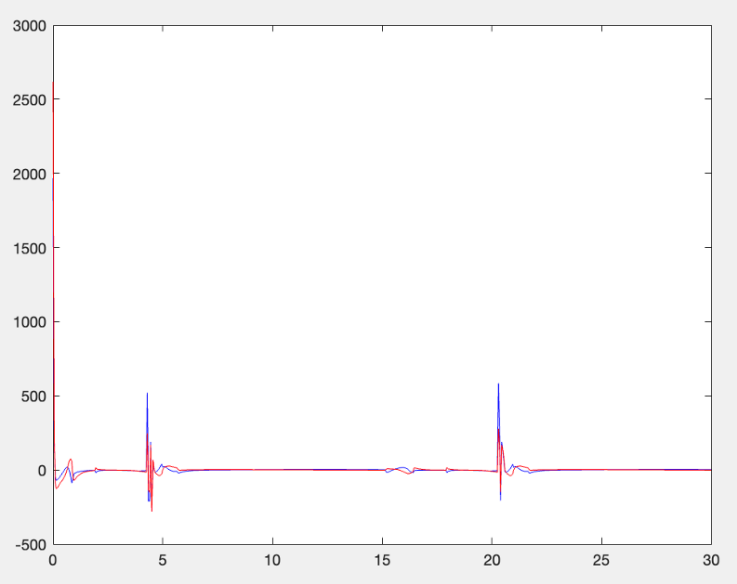

Figura 4.7. Sforzo di controllo per scenario in fig. 4.6

Consideriamo perciò delle saturazioni sulle accelerazioni in ingresso al nostro sistema che possano simulare le reali saturazioni presenti sui motori del robot.
Considerando che le singole accelerazioni di feedforward $u_{ff}^{(x)}$, $u_{ff}^{(y)}$ oscillano (dati i parametri di simulazione) tra -10 e 10, vediamo due scenari simulativi:

1. **Saturazione a $\pm$ 100**
   Con un ordine di grandezza superiore ai valori di feedforward, il comportamento è molto simile a quello senza saturazioni (fig. 4.6).

   Ha senso chiedersi però se sia sempre possibile fornire ingressi così tanto maggiori di quelli di feedforward, ed è quindi il caso di abbassare ulteriormente la saturazione.

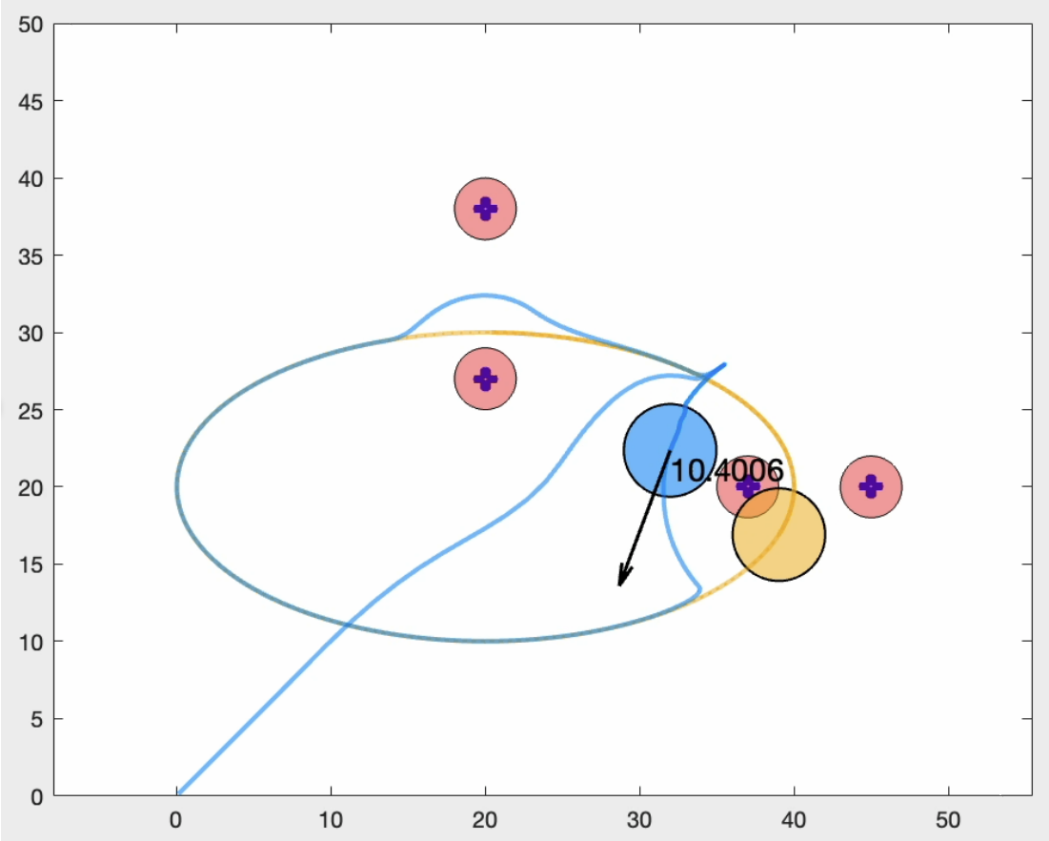

Figura 4.8. Traiettoria con saturazione ± 100 (scenario fig. 4.6)

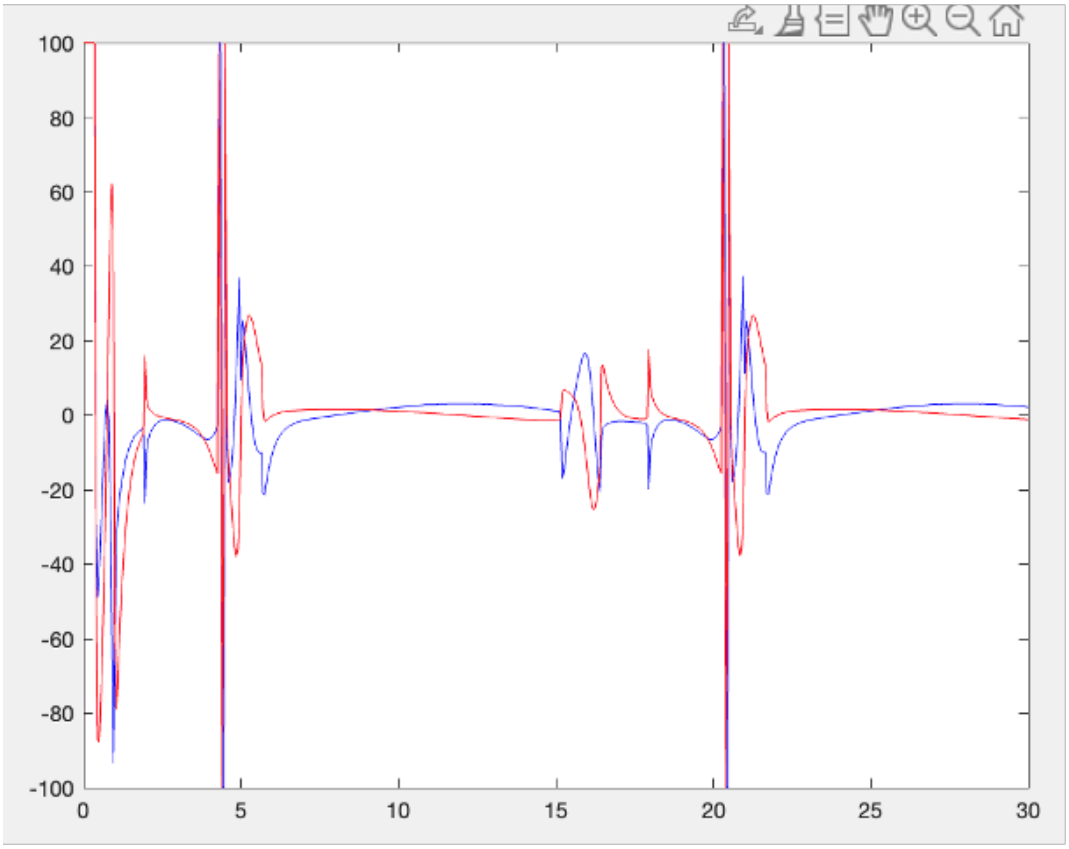

Figura 4.9. Sforzo di controllo con saturazione a ± 100

2. **Saturazione a ± 20**
   Con saturazione al doppio dei valori massimi di feedforward, si nota un peggioramento delle prestazioni che causa un *overshooting* della traiettoria quando si cerca di recuperare l'offset. Infatti, l'accelerazione negativa in saturazione non permette al robot di rallentare abbastanza prontamente per rimettersi in traiettoria.
   Una volta recuperato l'offset iniziale con relativo *overshooting*, però, il robot si comporta sempre discretamente, garantendo sia *trajectory tracking* che *collision avoidance*.

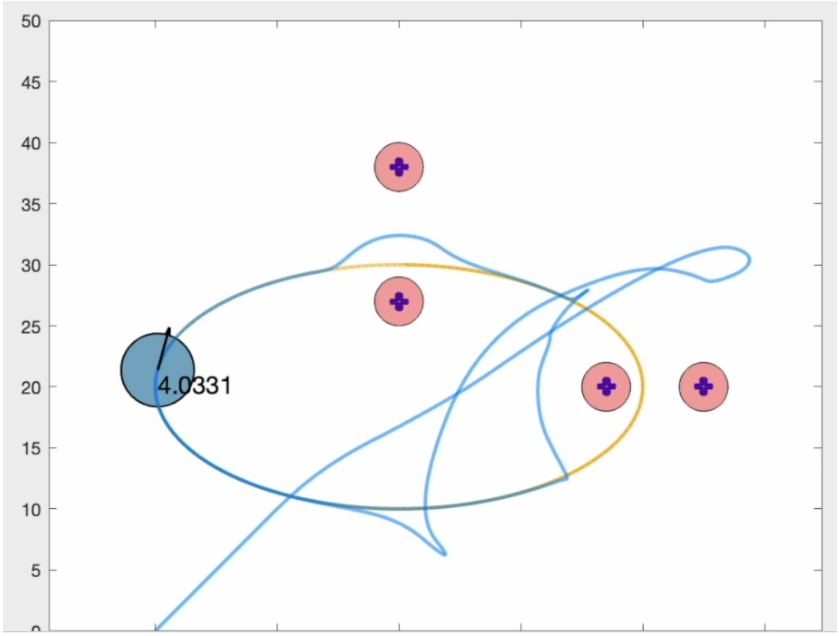

Figura 4.10. Traiettoria con saturazione ± 20 (scenario fig. 4.6)

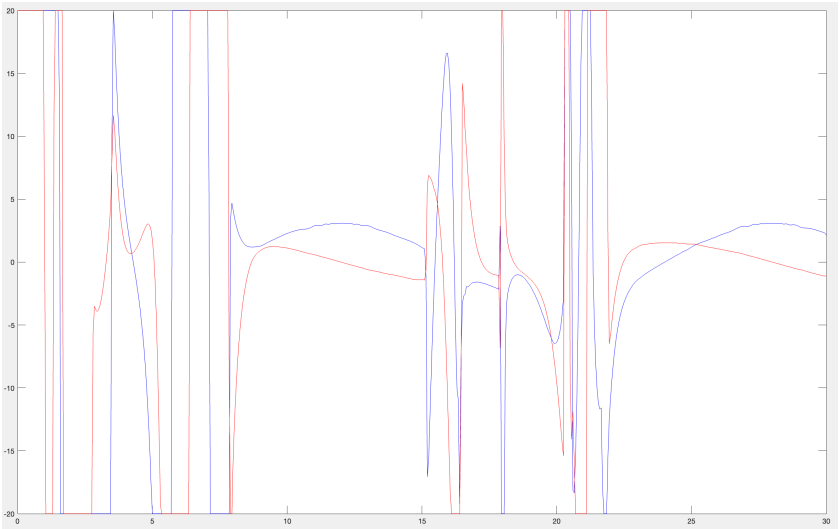

Figura 4.11. Sforzo di controllo con saturazione a ± 20

4.5.2 Ostacoli in Movimento

Consideriamo ora uno scenario non ideale, ma decisamente plausibile per un robot mobile.
Immaginiamo di avere sempre a disposizione (magari tramite un apposito sensore) la posizione degli ostacoli nell'ambiente. Supponiamo ora che gli ostacoli non siano statici, ma si trovino in movimento (nelle nostre simulazioni, proprio nell'intorno della traiettoria di riferimento).
Ci chiediamo allora, nell'ipotesi in cui **la velocità degli ostacoli sia NON nota**, può il nostro modello continuare a funzionare correttamente?

La risposta a questo quesito è: *dipende*. La mobilità degli ostacoli, con l'aggiunta della velocità non nota degli stessi, rende il collision avoidance una specifica più difficile da garantire.
Per **ostacoli abbastanza lenti** (per cui $v_{obs} \approx 0$), il modello è approssimabile a quello con ostacoli statici, e le simulazioni confermano il corretto funzionamento della legge di controllo..

Il problema si presenta su **ostacoli abbastanza veloci** (per cui $v_{obs} \gg 0$), dove l'azione di "evasione" del robot non è abbastanza repentina da evitare l'ostacolo.

Senza modificare il modello e la legge di controllo, possiamo però intervenire opportunamente sui parametri, in particolare sul parametro di penalty γ.

$$\dot{h} + \alpha(h - \gamma(\delta + r)) \geq 0 \qquad \gamma > 0$$

In questo modo stiamo aumentando la clearance dall'ostacolo, ma allo stesso tempo stiamo anticipando l'azione "evasiva" dall'ostacolo, al fine di evitare la collisione.
Ne conseguiranno delle traiettorie del robot più "ampie", in quanto esso si manterrà alla distanza opportuna dagli ostacoli (maggiore all'aumentare di γ).

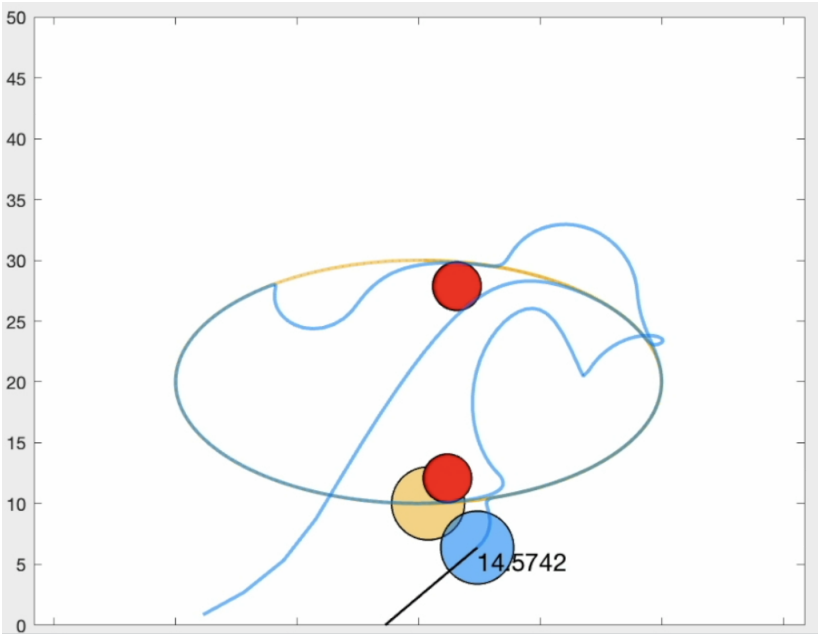

Figura 4.12. Collision avoidance garantito con opportuno γ

4.5.3 Saturazione ed Ostacoli Mobili

Unendo i due scenari appena descritti, ci troviamo davanti a un problema ancor più complesso: non solo il robot deve "reagire in tempo" così da evitare l'ostacolo, ma le accelerazioni di "evasione" devono essere abbastanza elevate da evitare le collisioni, cosa non scontata viste le saturazioni.

Ancora una volta, il modo più semplice di intervenire è quello di settare opportunamente il parametro γ a un valore abbastanza alto da garantire un comportamento adeguato (al prezzo di traiettorie più ampie e distanti da quella di riferimento, quando si evita un ostacolo).
Vediamo due casi (con saturazione a $\pm$ 20) con valori di γ rispettivamente di 8 e 16.

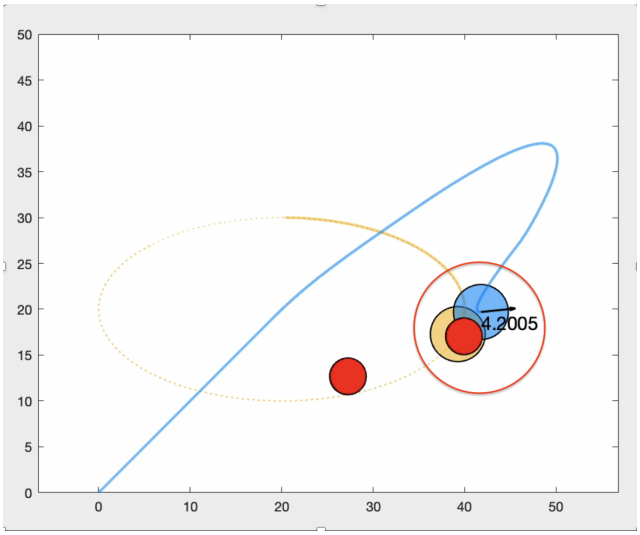

Figura 4.13. Collisione con $\gamma = 8$ (la velocità dell'ostacolo è troppo elevata e il robot "reagisce" troppo tardi)

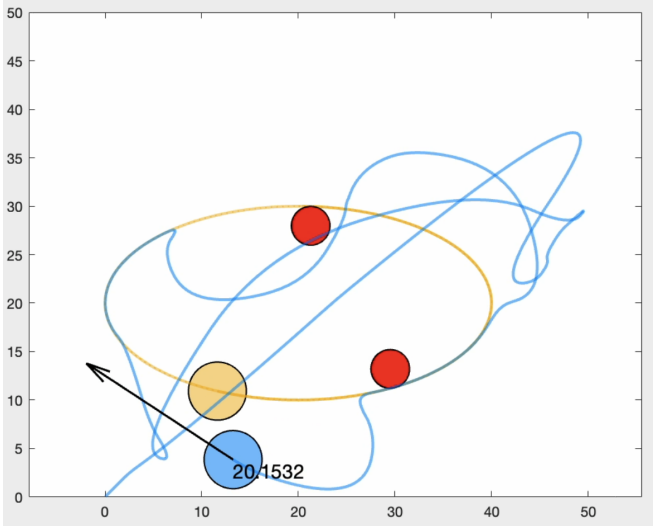

Figura 4.14. Con $\gamma = 16$ gli tutti gli ostacoli vengono evitati, al prezzo di traiettorie più ampie

Capitolo 5

Conclusioni

In quest'ultimo capitolo vogliamo analizzare i risultati ottenuti, riscontrando punti di forza e criticità della metodologia di controllo tramite CBF.

5.1 Ottimalità

Ricordiamo che il controllo applicato è soluzione di un problema di Quadratic Programming, cioè quello **ottimo** che minimizza lo "scostamento" dal controllo nominale.
La condizione di ottimalità è confermata dai risultati simulativi, dove lo sforzo di controllo si è sempre rivelato efficace per soddisfare le specifiche di progetto.

Ha senso dunque considerare il QP una tecnica molto valida per imporre il vincolo della CBF ed è difficile trovare dei lati negativi. L'apparente complessità computazionale è ammortizzata significativamente da algoritmi risolutivi efficienti (es. "interior-point-convex" con quadprog(...) in Matlab).

5.2 Tuning dei Parametri

Nel corso della presente relazione, abbiamo sottoposto il nostro modello a test sempre più *challenging*, alcuni dei quali vanno anche oltre lo scenario operativo per cui è stato progettato.
Basti pensare al fatto che abbiamo rimosso la staticità degli ostacoli e, pur non avendo alcuna informazione sulla velocità degli stessi, abbiamo ottenuto risultati significativi.
Abbiamo notato che i parametri vanno tarati accuratamente a seconda delle condizioni operative e dello scenario in cui opera il robot. Esiste un *trade-off* tra traiettorie più "strette" (con controllo più simile a quello nominale) e una maggiore robustezza in termini di *collision avoidance* (che comporta traiettorie più ampie e azioni di controllo anche molto distanti da quello nominale).

Abbiamo usato il **parametro di penalty** γ per bilanciare questo *trade-off* nella maniera più opportuna.

In caso di **ostacoli statici**, in cui si vuole magari minimizzare il consumo energetico, è conveniente impostare un $\gamma \approx 1$ (o comunque non troppo elevato). Così facendo le traiettorie saranno sempre il più possibile fedeli a quelle di riferimento, garantendo *collision avoidance* ma permettendo anche movimenti "agili" in presenza di cluster

di ostacoli (statici).

In presenza di **ostacoli dinamici**, soprattutto se dotati di una velocità non trascurabile, è necessario aumentare γ fino ad un valore opportuno, che minimizzi il rischio di collisione. Ovviamente il prezzo da pagare per garantire *collision avoidance* è un peggioramento delle traiettorie, che diventano più "larghe" (che comporta quasi sempre un maggior sforzo di controllo e consumo energetico).

La metodologia di controllo risulta essere dunque abbastanza flessibile e garantisce il rispetto delle specifiche anche in condizioni operative molto *challenging*, come quelle con ostacoli in moto a velocità ignota.
Allo stato attuale però, il ***tuning* dei parametri** va effettuato a priori. In caso di cambiamento dello scenario operativo a posteriori, il robot potrebbe continuare ad agire in maniera non ottimale (es. sovrastima di γ) o peggio, entrare in collisione (es. sottostima di γ).

Ha senso quindi implementare, in un'applicazione reale, un algoritmo di adattamento dei parametri alle condizioni operative (o quantomeno, del parametro γ).
Si potrebbe ad esempio adottare una politica di *switching* tra due valori γ_{low} e γ_{high} al verificarsi di opportune condizioni.
Avendo a disposizione la misura delle velocità degli ostacoli, si potrebbe invece rendere γ proporzionale ad essa, oppure considerare una nuova CBF che tanga conto di tali velocità.

5.3 Estensione a Modelli più Complessi

Per quanto i risultati simulativi siano soddisfacenti, va riconosciuta un'oggettiva approssimazione nel modello di sistema.
Il nostro sistema a doppio integratore non può che essere considerato la semplificazione di un controllore "in cascata" (ignoriamo completamente fattori come le azioni dei motori, il controllo in corrente ecc.)
Oltre a questa approssimazione, abbiamo considerato un robot omni-direzionale (assenza di vincoli cinematici), e abbiamo inoltre trascurato del tutto il controllo dell'**orientamento**.

Avrebbe senso estendere lo studio effettuato a modelli di robot mobili più complessi, magari in presenza di vincoli cinematici [3] (es. robot *car-like* o di tipo uniciclo) e/o in cui l'orientamento è significativo (es. robot asimmetrici per cui passare in mezzo agli ostacoli è possibile solo in una determinata configurazione).

Ciò non intacca in alcun modo la validità dei risultati ottenuti in questo primo studio semplificato, che possono essere usati come solida base per l'estensione del problema di controllo a modelli più complessi.

Bibliografia

[1] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada,"*Control Barrier Functions: Theory and Applications*" in 18th European Control Conference (ECC), 2019, pp. 3420–3431.

[2] A. Cristofaro, M. Ferro, and M. Vendittelli "*Safe trajectory tracking using closed-form controllers based on control barrier functions* ", in 2022 IEEE 61th conference on decision and control (CDC). IEEE, 2022, pp. 3329–3334.

[3] S. Fukuda, Y. Satoh and O. Sakata, "*Trajectory-Tracking Control Considering Obstacle Avoidance by using Control Barrier Function*", 2020 International Automatic Control Conference (CACS), Hsinchu, Taiwan, 2020, pp. 1-6.