POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria dell'Automazione

Simulazione ibrida di sistemi meccatronici

Relatore: Prof. Matteo G. ROSSI

Rafael C. BUENO

Matr. 779528

Anno Accademico 2012-2013

Alla mia famiglia e

a tutti che hanno di

qualche forma aiutato

Abstract

I sistemi meccatronici hanno spesso situazioni dove il fallimento non è accettabile. Questo progetto propone a testare l'uso di un co-simulatore come metodo per simulare le situazioni di rischio prima di eseguire nella pratica, usando anche di questo co-simulatore come aiuta per la proiettazione del sistema.

Il co-simulatore scelto è MCA, che è stato sviluppato precedentemente nel Politecnico di Milano [1]. Lo MCA usa il linguaggio Modelica per modellare l'ambiente e Zot per la parte del sistema, attraverso un linguaggio logico. Il lavoro è stato quello di capire il co-simulatore e il linguaggio logico, modellare un sistema meccatronico scelto e verificare la facilità o difficoltà del metodo.

Si mostrerà nella relazione che il metodo si è rilevato una buona alternativa alla progettazione dei software embedded per sistemi meccatronici, senza mettere in rischio questi sistemi. E alla fine il metodo si è mostrato pure molto utile anche nella scrittura delle regole e di facile apprendimento per un utente non esperto, dato che si è imparato a usarlo in meno di 2 mesi.

Indice

Indice delle figure

Indice delle tabelle

Indice dei grafici

1. Introduzione

La progettazione di sistemi di software embedded richiede estrema attenzione perché si tratta spesso di variabili che non possono essere oggetto di fallimento. Così, per la progettazione di tali sistemi, di solito si ricorre a un software di simulazione per riprodurre l'ambiente al fine di provare come lavori il software nel suddetto ambiente.

Ma i modelli sono di natura diversa: l'ambiente lavora nel continuo e il sistema funziona nel discreto, e ciò comporta un problema. Si vuole eseguire una simulazione in modo che l'andamento del sistema (come spesso sono chiamati i componenti software) vada in parallelo con quello dell'ambiente (dove la componente software è incorporata).

D'altra parte questa diversa natura comporta delle difficoltà nell'ottenere un modello complessivo che riesca a mettere insieme entrambi modelli. Così, la soluzione generalmente adottata è l'uso dei co-simulatori, che sono software ideati con l'obiettivo di creare questo collegamento tra l'ambiente e il sistema.

Nei sistemi meccatronici queste situazioni di rischio accadano spesso, per esempio quando esiste la possibilità che un braccio robotico collida con una persona, un oggetto o ancora con un altro braccio che lavora nello stesso spazio.

L'obiettivo di questo lavoro è quindi comporre la modellazione di un sistema meccatronico semplice (una serie di bracci robotici che lavora nello stesso spazio operativo), in modo da mostrare i vantaggi di questo approccio e, se necessario, sottolineare parti del software su cui occorre effettuare migliorie.

Si vuole anche far notare qual è la difficoltà nell'imparare le basi necessarie a programmare le regole logiche, da parte di un utente non esperto in linguaggio logico.

Il software scelto è MCA, sviluppato in [1], un cosimulatore di Zot [9] e Modellica[3], dove l'ambiente è modellato su modelica e il sistema è modellato in Linguaggio logico su Zot.

L'idea iniziale era quella di creare tutti i modelli in 3D, ma ciò avrebbe comportato una complessità non interessante ai fini di questo progetto.

Alla fine bastano i modelli semplificati in 2D per provare l'utilizzo del programma e in questo modo il modello 3D risulta essere solo un possibile ampliamento del presente

progetto, in quanto i modelli non sono chiusi e possono espandersi nella descrizione di modelli più realistici o ancora si possono aggiungere altri componenti, usando la stessa metodologia.

Con questa idea le regole logiche sono state scelte seguendo il buon senso, ossia non presentano una teoria basata dietro ad esse ma sono quelle che sembrano più adatte al problema in esame, dato che l'obiettivo non è utilizzare le giuste regole ma vedere come esse funzionano.

Il lavoro è stato diviso in 3 parti, che corrispondono ai seguenti capitoli:

- Background
- Modellazione e Progettazione
- Prove Sperimentali / Simulazione

2. Background

Quando si intende controllare un sistema fisico (continuo) utilizzando un software embedded (discreto), si incontrano dei problemi nel trovare una soluzione unificata.

A causa della loro diversa natura, continuo/discreto, la maggior parte degli approcci per la progettazione si basa su modelli e strumenti di sviluppo diversi per l'ambiente e per il sistema.

A questo punto per unificare i modelli si usano dei co-simulatori, che sono responsabili di fare da ponte tra il modello fisico (l'ambiente) e il modello software (il sistema) e anche di valutare il loro comportamento coordinato. La co-simulazione consente l'utilizzo di diversi strumenti di simulazione che lavorano in contemporanea e che sono in grado di scambiare informazioni tra di loro in modo collaborativo.

L'obiettivo delle simulazioni è usare dei modelli del sistema e dell'ambiente in modo da costruire una traccia che sia accettata da entrambi i modelli, cioè che non violi qualche condizione su ognuno di essi.

Per eseguire le analisi e i test di co-simulazione che saranno provati in questo progetto, si utilizzerà il software sviluppato in [1], la MCA (the MADES Cosimulation Approach). Questo software mette insieme un approccio acausale dell'ambiente ad una rappresentazione che si serve di regole logiche sul sistema, in modo da consentire la verifica formale di queste regole nel sistema risultante, ad ogni istante di campionamento.

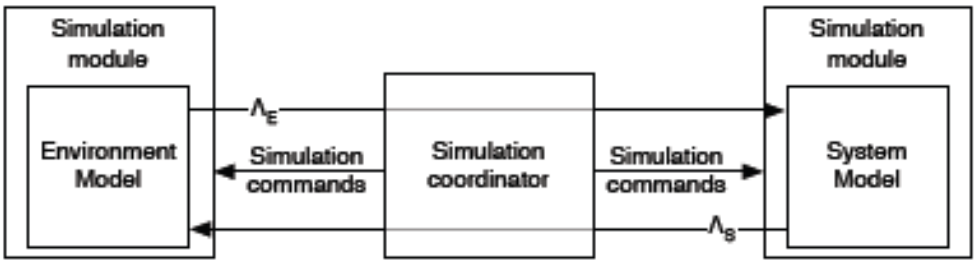


Figura 2.1. – Architettura del co-simulatore [1]

2.1. Il co-simulatore – MCA/Mades

Il software MCA [1] è stato scelto per permettere ai progettisti di combinare diversi formalismi complementari (equazioni differenziali, logica formule) in condizioni di continuità, invece di richiedere loro di soddisfare un singolo modello con notazioni uniche. In questo modo si sfruttano i rispettivi punti di forza dei formalismi coinvolti.

Il funzionamento del MCA è basato sulla progettazione del sistema attraverso un procedimento spiegato in [1] e [8] e con base nel linguaggio TRIO[5], che è una logica lineare e temporale del primo ordine. Alla fine la verifica della soddisfazione del sistema è fatta dal Zot [4], un programma anteriormente sviluppato nel Politecnico di Milano.

Nel MCA i modelli di sistema e ambiente comunicano tra di loro attraverso variabili condivise. Il sistema e l'ambiente hanno però anche delle variabili private che non sono visibili, come si vede nella figura 2.2.

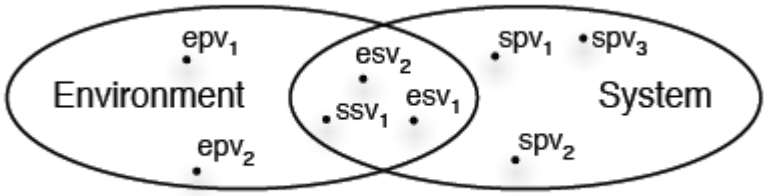


Figura 2.2. – Schema delle variabili del sistema di co-simulazione [1]

Queste variabili possono essere a valori reali o segnali finiti discreti, come variabili logiche. Le variabili dell'ambiente sono soggette ad equazioni differenziali o algebriche definite dal linguaggio Modelica e le variabili del sistema sono soggette alle regole logiche. Le prime hanno una nozione di tempo continuo e le altre di tempo discreto.

L'architettura del MCA, estratta dal [1], può essere espressa come:

- Il modulo di simulazione per l'ambiente, che si basa sull'interprete OpenModelica.
- Il modulo di simulazione per il sistema basato sullo strumento di Zot, che può funzionare come un risolutore di soddisfacibilità di formule logica temporale.
- Il coordinatore, un'applicazione stand-alone Java che prende come input i modelli di ambiente e di sistema e di alcuni parametri idonei (ad esempio, campionamento DELTA, lunghezza della traccia, ecc), quindi esegue l'algoritmo descritto sopra per produrre una traccia del sistema complessivo.

Per definire il legame tra le variabili si è creato un file nel formato XML dove si scrivono le variabili (figura 2.2) e se ne definisce il tipo, ambiente o sistema, privata o condivisa.

2.2. Il linguaggio OpenModelica

È un linguaggio orientato agli oggetti (object-oriented), quindi si definiscono le classi di modelli. È basato su equazioni, non su assegnamenti (ci sono gli assegnamenti ma anche questi sono trattati dal software come equazioni), ottenendo un approccio non-causale e dichiarativo.

Allora le equazioni fisiche importanti dei modelli possono essere definite semplicemente dal sistema di equazioni dei singoli componenti del modello, senza la necessità di "isolare" la variabile da definire nel software. Questo permette di definire e usare i componenti da soli e dopo metterli insieme nel modello, e ciò rappresenta una grande comodità.

Per mettere insieme i modelli di ogni componente, si arriva a ciò che è la forza del linguaggio: i connettori, che generano equazioni che legano questi modelli. Con il loro uso si riesce a semplificare i modelli ottenendone di più facili dal punto di vista della comprensione, e sembrano come gli schemi di modellazione fatti su carta.

Ogni componente (ad esempio un resistore) è specificato attraverso un sistema di equazioni differenziali algebriche e può essere definito separatamente, e si possono usare dei modelli già fatti e che sono contenuti in una libreria incorporata.

È a questo punto che si vede l'importanza delle connessioni, poiché corrispondono ai principi fisici che devono essere rispettati tra i due componenti modellati, definendo la conservazione delle variabili sforzo e di flusso.

2.3. Linguaggio Logico TRIO e il simulatore ZoT

Un linguaggio logico definisce gli stati del sistema basato in proposizioni logiche, per esempio: $A \rightarrow B$

Questa è una proposizione logica che è costante nel tempo.

Dall'altra parte un linguaggio logico temporale lineare, è come la logica proposizionale con l'aggiunzione della capacità di gestire proposizioni relative a tempi diversi. Permette quindi di esprimere delle proprietà legate al tempo.

Un linguaggio logico temporale ha gli operatori logici comuni (¬,∧,∨,→ ,↔) e in più gli operatori temporali di base:

- Next(p) : la proprietà p è vera nell'istante successivo;
- Finally(p): p è vera in alcun instante nel futuro;
- Always(p): p è sempre vera;
- Until(p,q): p è vera fino a quando q è vera;

Derivati da questa base, ci sono definite tante linguaggi, come LTL, MTL, TRIO, ecc. Non è scopo di questa relazione spiegare specificamente tutte le proprietà dei linguaggi, per questo basta riportarsi a [6].

Le logiche temporali hanno una capacità di esprimere le proprietà in un modo intuitivo conciso. TRIO è un linguaggio del primo ordine, con operatori temporali riferiti al passato e al futuro che permettono di esprimere proprietà il cui valore cambia nel tempo.

Ogni formula TRIO assume significato rispetto ad un istante di tempo che è lasciato implicito. TRIO ha in più come operatori basi Futr(p,t) e Past(p,t), che sono definiti come, supponendo che l'istante attuale sia T (implicito):

- Futr(p,t) : p è vera nell'istante T+t;
- Past(p,t) : p è vera nell'istante T-t;

Un gran numero di operatori temporali possono essere ottenuti a partire da Futr e Past. Di seguito ne sono elencati quelli che saranno importanti per questa relazione:

- AlwF(p) : p sarà vera in tutti gli istanti da T in poi;
- AlwP(p) : p è stata vera in tutti gli istanti fino a T;
- SomF(p): p sarà vera in alcun istanti da T in poi;

- SomP(p): p è stata vera in alcun istanti fino a T;
- Lasts(p,t): p sarà vera per i prossimi t istanti;
- Lasted(p,t): p è stata vera negli ultimi t istanti;
- WithinF(p,t): p sarà vera in alcun istanti da T fino a T+t;
- WithinP(p,t) p è stata vera in alcun istanti da T-t fino a T;

Alla fine Zot è un programma satisfiability checker delle regole logiche definite in TRIO, cioè, fa la verifica se tutte le regole imposte hanno una possibile soluzione e secondo le sue logiche proprie torna come risultato una di queste possibili soluzioni.

3. Progettazione

3.1. Modello dell'ambiente

Il modello dell'ambiente è rappresentato da un insieme di bracci robotici che si trovano nello stesso spazio di lavoro. Nella modellizzazione di ogni braccio ci mette insieme sia il modello fisico del braccio libero nello spazio sia il modello del motore che aziona le articolazioni flessibili al fine di muovere il generico braccio.

3.1.1. Modello generale di un braccio controllato

Per ogni braccio si riporta il seguente modello fisico (figura 3.1), composto dalle classi Arm, Motor, Controller e InverseKinematics.

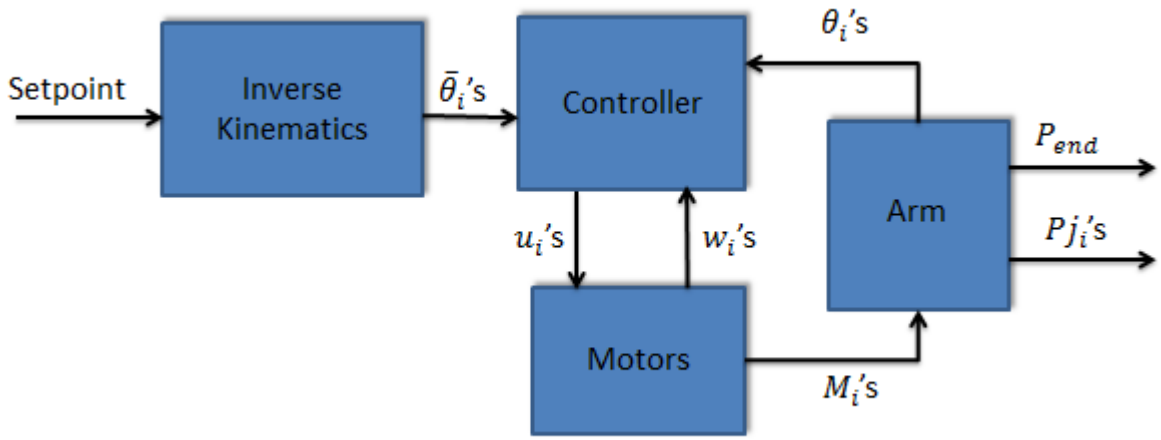


Figura 3.1. – Schema di modellazione di un braccio

La classe Arm definisce il braccio libero nello spazio, cioè senza i torque applicati ai giunti, definendo le relazioni cinematiche e dinamiche del braccio.

Ha come ingresso i torque applicati ai giunti, e come uscita gli angoli (θ_i), le posizioni spaziali (Pj_i) dei giunti e la posizione dell'end effector.

La classe Motor definisce i motori che generano i torque per la movimentazione dei link.

Possono essere normali DC brushless o motore passo-passo. L'altra classe è il controllore che prende l'errore angolare e genera le tensioni da applicare ai motori.

In ultimo è presente un sistema che si occupa della cinematica inversa (la classe InverseKinematics). Questa parte è necessaria per ottenere le posizioni angolari di ogni giunto dalla posizione finale, perché il setpoint dato è la posizione dell'end effector (mentre si possono controllare solo gli angoli di giunto).

Esistono due metodi per effettuare la cinematica inversa: calcolo numerico e calcolo analitico. Per sistemi complessi è più adatta la prima soluzione, attraverso il calcolo dello jacobiano [1]. Nel presente studio si userà il calcolo analitico perché il sistema è semplice.

3.1.2. Modello di un braccio planare con 2 g.d.l. controllato

Per uno studio che non porti ad una complessità eccessiva, è stato scelto un braccio dotato di 2 gradi di libertà (g.d.l) capace di compiere un movimento planare (senza gli effetti gravitazionali) per poter così ottenere analiticamente la cinematica inversa del braccio (figura 3.2).

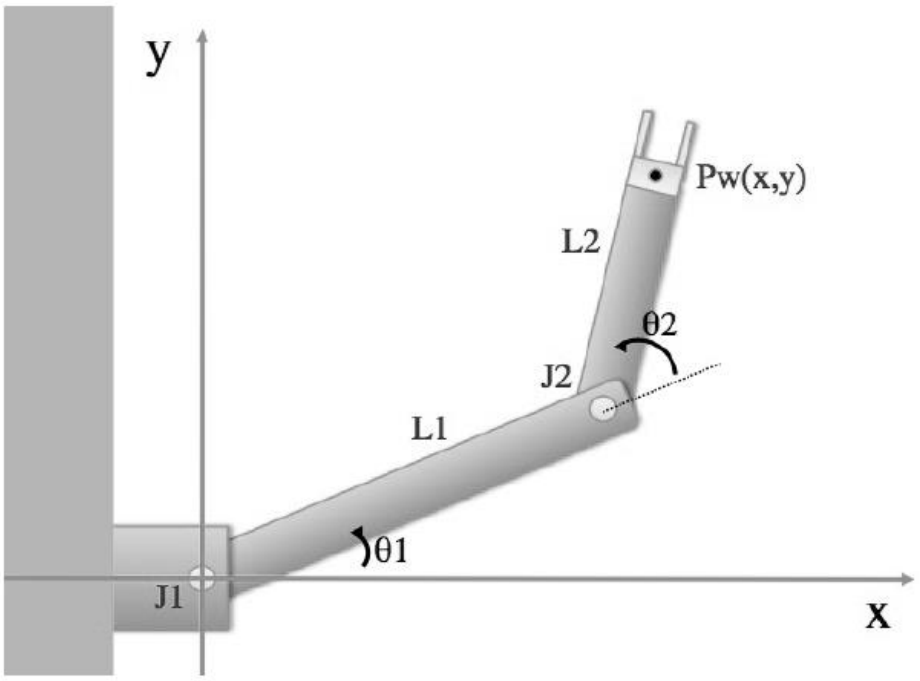


Figura 3.2 – Schema di un braccio articolato con 2 g.d.l. e moto planare

Il braccio è composto da due giunti (J1 e J2) dove entrano i torque aggiuntivi dei motori per far muovere i link L1 e L2 rispettivamente (facendo crescere θ_1e $\theta_2$della figura 2.2). I motori scelti sono normali DC Brushless, schematizzati in modo semplificato come un generatore di coppia, una costante di tempo in cascata al comando e una cassa di riduzione(gearbox), come si vede dall'ambiente modelica riportato in figura 2.3.

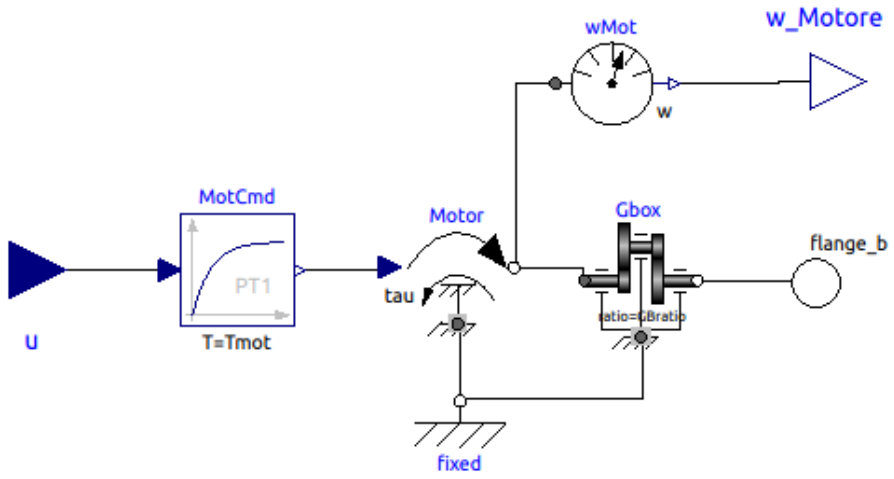


Figura 3.3. – Schema OpenModellica dei Motori

Data la semplicità, si può progettare il controllore come due controllori semplici desaccopiati sulla posizione angolare di ogni motore. Ogni controllore agisce in cascata come un semplice PI (Proporzionale-integrale) sulla velocità del motore e un P (Proporzionale) sulla posizione.

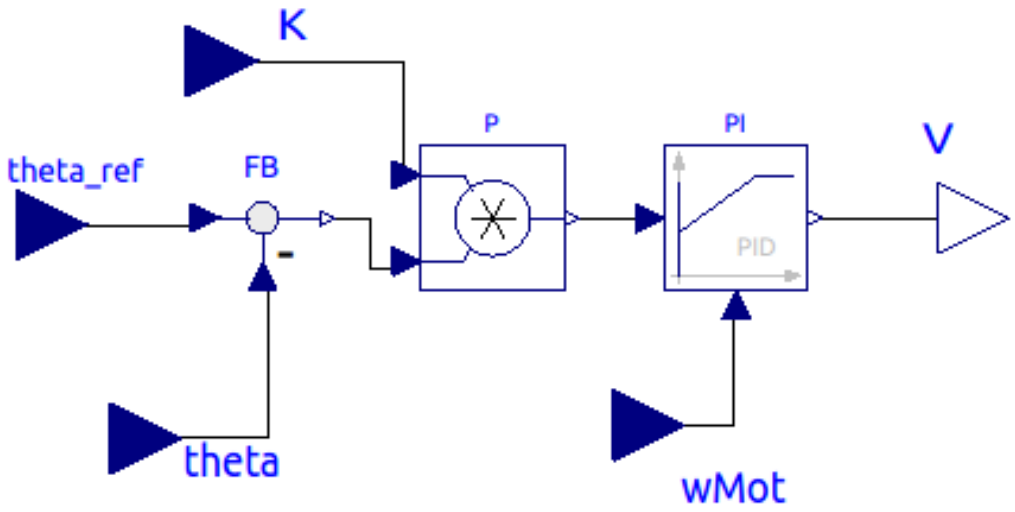


Figura 3.4. – Schema OpenModellica dei controllori

Progettato in questo modo, il sistema complessivo Controllore+Motori riesce a portare il braccio all'angolo desiderato. Si vuole che le prestazioni del regolatore siano tali da garantire una risposta veloce senza che sia presente alcuna sovraelongazione. Per rispettare questa specifica, sono stati scelti come parametri del regolatore PI i valori di $K_p = 20$ e $T_i = 0.05\ s$, di modo che inizialmente il guadagno K, del regolatore P, rimanga costante e uguale a 1.

Per il calcolo degli angoli desiderati, si usa la cinematica inversa analitica, che in questo caso si riduce alle seguenti equazioni (estratte da [2]):

$$B = \sqrt{(SP_x - O_x)^2 + \left(SP_y - O_y\right)^2}$$

$$q_1 = atan2\left(SP_y - O_y, SP_x - O_x\right)$$

$$q_1 = acos\left(\frac{L_1^2 - L_2^2 + B^2}{2.L_1.B}\right)$$

$$\bar{\theta}_1 = q_1 + q_2$$

$$\bar{\theta}_2 = -\pi + acos\left(\frac{L_1^2 + L_2^2 - B^2}{2.L_1.L_2}\right)$$

Dove:

- $\left(SP_x\ , SP_y\right)$ è il setpoint desiderato
- $\left(O_x\ , O_y\right)$ è l'origine del braccio (posizione del primo giunto, importante nel caso in cui si considerino più bracci insieme)
- L_1 e L_2 sono le lunghezze dei link 1 e 2

Alla fine abbiamo il modello proprio del braccio, la classe ARM che si vede nella figura 3.4.

In questo modello è presente una componente di traslazione (origin) per usare questa classe quando si mettono più bracci insieme. I giunti (Joint_1 e Joint_2) sono rotazionali sull'asse z, quindi non è presente l'azione della forza gravitazionale. I giunti sono caratterizzati anche da smorzamento e rigidezza, scelti secondo i valori solitamente usati [7].

Dopo aver messo insieme tutte le classi sopra elencate, come mostrato nella figura 3.1, si è giunti al modello del braccio controllato. In entrata al sistema fisico si hanno le seguenti variabili:

- Le posizioni dell' END EFFECTOR, ovviamente perchè è l'obbietivo del movimento del braccio.
- I guadagni dei controllori legati agli angoli dei giunti, perché con questo possiamo gestire bene il movimento del braccio, diminuendo l'azione di controllo o bloccando il movimento del braccio data qualche perturbazione non desiderata.

Le uscite sono tutte le posizioni dei giunti e dell'end effector che sono importanti per i calcoli degli errori e delle possibili collisioni.

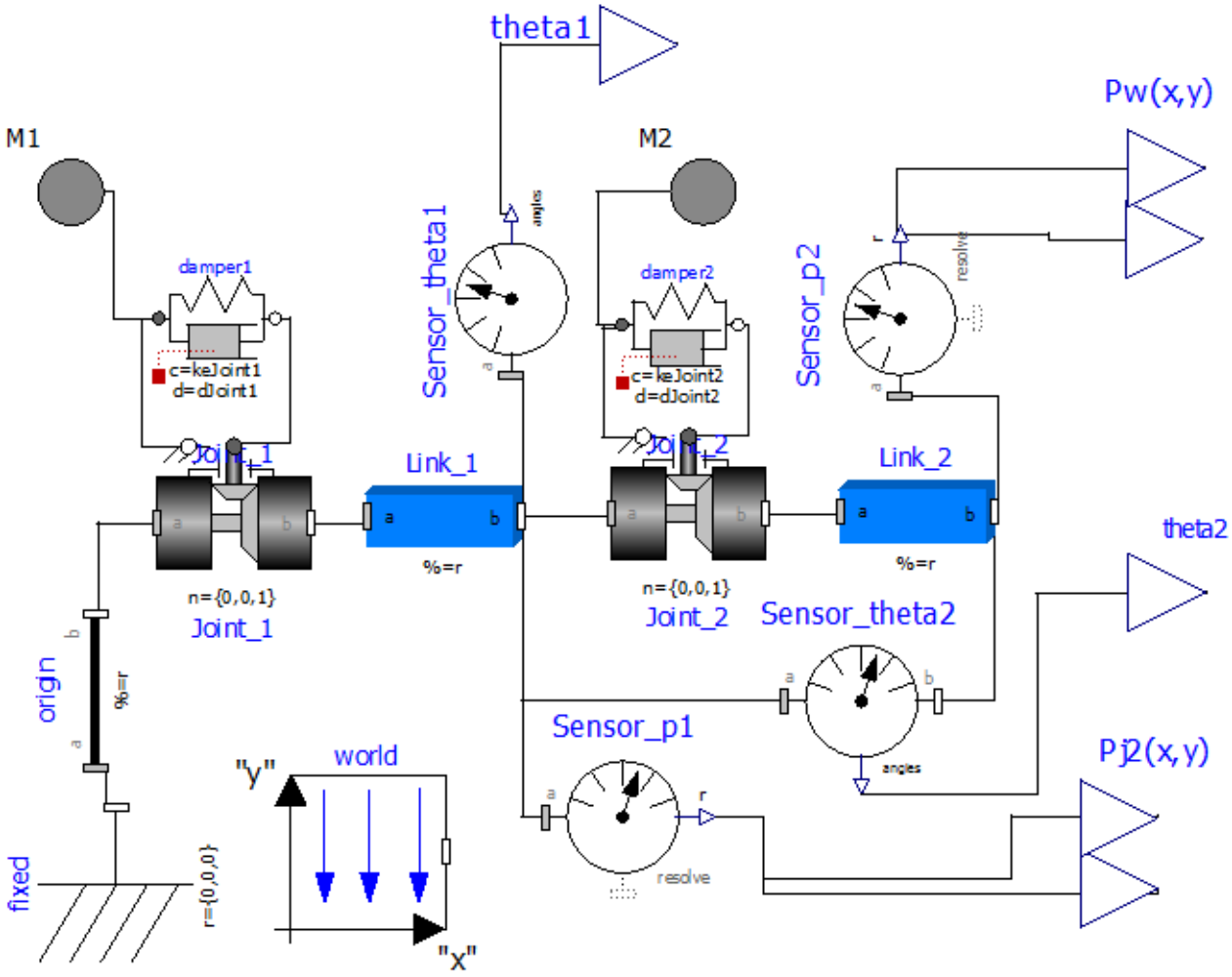


Figura 3.5. – Schema OpenModellica del braccio

3.1.3. Modello del sistema con più di un braccio lavorando insieme

Con il modello del braccio controllato descritto precedentemente, possiamo inserire 2 o più bracci nello stesso spazio di lavoro, e ciò è possibile posizionando le rispettive origini in punti diversi:

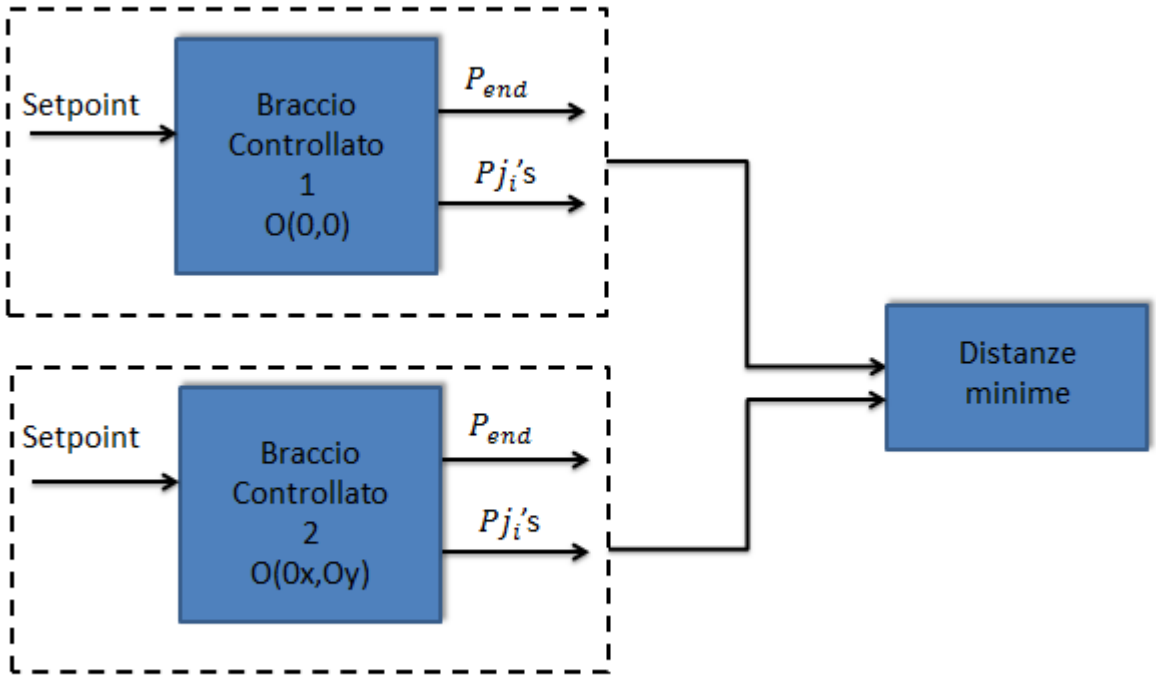


Figura 3.6. – Schema del sistema con 2 bracci

Si nota dalla figura che quando si decide di posizionare 2 o più bracci insieme è importante tenere conto della distanza tra di essi.

Per il calcolo della distanza si usano le usuali formule della distanza da un punto a una retta, e altre correzioni che saranno illustrate e spiegate nelle prove sperimentali.

3.2. Modello della parte logica

Il sistema del software interagisce con l'ambiente leggendo le misure, per esempio dell'errore, e definisce i guadagni del controllore ed i setpoint in accordo con le logiche interne definite.

Il modello del sistema è espresso come un insieme di restrizioni temporali, ma si lasciano liberi ancora alcuni punti di modo che il sistema risulti non deterministico. Per esempio, non si comunica al sistema che il guadagno debba essere uguale a un valore, ma che si debba trovare all'interno di un certo range di valori. Questo porta ad un vantaggio notevole, in quanto permette al software di scegliere la miglior soluzione secondo la sua logica interna.

È importante notare che le regole qui sottolineate sono state scelte secondo il buon senso, poiché l'obiettivo prefissato non era di definire precisamente le regole ma di mostrare il funzionamento complessivo e le opzioni che si offrono al progettista quando si usa il software di cosimulazione.

3.2.1. Vincoli su ogni braccio

Le prime regole ideate, e usate nella prima simulazione, sono quelle per fare in modo che il sistema sia un po' più veloce, cambiando una delle entrate del sistema fisico, i guadagni del controllore P (Proporzionale di posizione), d'accordo con la dimensione dell'errore, seguendo il principio:

Tabella 3.1. – Relazione qualitativa tra errore e guadagno

Errore	**Guadagno**
Grande	Grande
Piccolo	Piccolo

È stata quindi definita una serie di formule, le quali vincolano il guadagno di ogni regolatore (rappresentato dalla variabile g) di modo che questo rimanga entro un certo range a seconda del valore dell'errore (rappresentato dalla variabile err). Di seguito si riporta una tabella riassuntiva che riporta i range pensati:

Tabella 3.2. – Definizione della relazione tra errore e guadagno

Errore (range)	Guadagno (range)
err > 90°	K_{gde}< g < K_{max}
60° < err < 90°	K_{med}< g < K_{gde}
10° < err < 60°	K_{picc}< g < K_{med}
err < 10°	K_{min} < g < K_{picc}

I valori dei range di errore sono stati definiti in base a studi già realizzati. D'altra parte i valori dei guadagni sono stati definiti da piccole simulazioni sul modello del braccio, sapendo i valori limite di saturazione del regolatore, e che saranno riportati successivamente, all'inizio della parte sperimentale.

Per fare questi cambiamenti quando si ritiene che siano davvero necessari, si deve imporre l'intervallo di tempo in cui l'errore si trova all'interno del generico range, poiché agendo in altri modi si ha il rischio di effettuare cambiamenti inutili sul guadagno. Per esempio nel secondo caso, la decisione presa è la seguente: il guadagno cambia se l'errore si trova tra 60° e 90° negli ultimi $T_{err} = 5$ t.u.

In base a quanto già riportato, definiamo le seguenti formule che valgono in ogni istante di tempo (cioè, "sempre"):

$$Alw\big(Lasted(err > 90\,, T_{err}) \rightarrow K_{gde} < g < K_{max}\big) \quad (1)$$

$$Alw\big(Lasted(60 < err < 90\,, T_{err}) \rightarrow K_{med} < g < K_{gde}\big) \quad (2)$$

$$Alw\big(Lasted(10 < err < 60\,, T_{err}) \rightarrow K_{picc} < g < K_{med}\big) \quad (3)$$

$$Alw\big(Lasted(err < 10\,, T_{err}) \rightarrow K_{min} < g < K_{picc}\big) \quad (4)$$

Queste appena riportate definiscono le regole su cui si basa il funzionamento di un singolo regolatore, ma ovviamente queste valgono per entrambi i regolatori e devono essere quindi raddoppiate nella scritura del linguaggio logico.

Per una seconda simulazione si è introdotta un'altra serie di regole per controllare l'esecuzione della traiettoria del generico braccio.

Il primo vincolo necessario da definire è un vincolo di sicurezza. Questo esprime la necessità di modificare il moto del robot se nel suo raggio d'azione viene a trovarsi una persona o un generico ostacolo. Esso quindi non può continuare normalmente a seguire la

traiettoria pre-impostata. È importante notare che l'obiettivo di sicurezza cambia se nello spazio di lavoro del robot è presente una persona o un ostacolo, in quanto è ovvio come in presenza di una persona nello spazio di lavoro sia più importante la salvaguardia dell'incolumità della persona, mentre in presenza di un ostacolo è il robot ad essere soggetto del problema di sicurezza (potrebbe riportare danni nell'impatto).

A questo punto e in questa situazione occorre effettuare una scelta, difatti il robot può rallentare il suo movimento per non collidere fortemente con il corpo o può fermarsi fino a che la posizione dell'ostacolo non sia prossima ad esso. Rimanere un po' di tempo in questa condizione sembra essere l'opzione migliore, anche se il sensore non avverte più la vicinanza del corpo.

Si è scelta la seconda opzione, e si progetta la regola riportata di seguito, dove *DontMove* è un predicato che impone al braccio di non muoversi ed è stato scelto un tempo (T_{person}) di 5 t.u. per rimanersi ancora fermo:

$$Alw\left(PersonDetected \rightarrow Lasts\left(DontMove\ , T_{person}\right)\right) \qquad (5)$$

Analogamente possiamo definire un vincolo che esprime la seguente idea: se il robot è arrivato al target (setpoint), deve rimanere fermo in questa posizione per un intervallo di tempo definito. Questo serve ad esempio per permettere all'End effector di avere tempo per chiudersi su un eventuale oggetto la cui posizione coincide con il target. Questa regola è espressa dalla seguente formula logica temporale (7), dove *OnTarget* è un predicato che dice se il robot è arrivato al target:

$$Alw\left(OnTarget \rightarrow Lasts\left(DontMove\ , T_{ontarget}\right)\right) \qquad (6)$$

Le prossime regole riportate, in particolare quelle indicate dai numeri dall' (7) alla (9), definiscono i predicati di prima, dove *OnTgx e OnTgy* sono predicati creati per sapere se l'end effector è sufficientemente vicino al target in x e in y, e *eps* è la tolleranza del sistema:

$$Alw\left(OnTarget \leftrightarrow Lasted(OnTgx\ \wedge OnTgy\ , 1)\right) \qquad (7)$$

$$Alw\left(OnTgx \leftrightarrow (-eps < (Pw_x - SP_x) < eps\ )\right) \qquad (8)$$

$$Alw\left(OnTgy \leftrightarrow \left(-eps < \left(Pw_y - SP_y\right) < eps\ )\right)\right) \qquad (9)$$

La regola *DontMove* può essere definita in due modi diversi, con l'obiettivo di annullare la tensione che arriva al motore: annullando i guadagni o cambiando il setpoint, o ancora mettendo la posizione attuale come nuovo setpoint.

La prima è di facile definizione, ma risulta indispensabile aggiungere alle regole dall'(1) al (4) una condizione, la quale esprime la necessità di non essere nella situazione *DontMove*:

$$Alw\left(\neg DontMove \rightarrow \left(Lasted(err > 90\ , T_{err}) \rightarrow K_{gde} < g < K_{max}\right)\right) \qquad (1b)$$

$$Alw\left(\neg DontMove \rightarrow \left(Lasted(60 < err < 90\ , T_{err}) \rightarrow K_{med} < g < K_{gde}\right)\right) \qquad (2b)$$

$$Alw\left(\neg DontMove \rightarrow \left(Lasted(10 < err < 60\ , T_{err}) \rightarrow K_{picc} < g < K_{med}\right)\right) \qquad (3b)$$

$$Alw\left(\neg DontMove \rightarrow \left(Lasted(err < 10\ , T_{err}) \rightarrow K_{min} < g < K_{picc}\right)\right) \qquad (4b)$$

E infine la definizione della regola:

$$Alw\left(DontMove \rightarrow (g_1 = 0\ \wedge\ g_2 = 0)\right) \qquad (10)$$

Per quanto riguarda la seconda opzione occorre definire un'altra variabile che conserva il Setpoint finale che si vuole. Quindi abbiamo le seguenti regole:

$$Alw\left(DontMove \rightarrow \left(SP_x = Pw_x\ \wedge SP_y = Pw_y\right)\right) \qquad (11)$$

$$Alw\left(\neg DontMove \rightarrow \left(SP_x = Tg_x\ \wedge SP_y = Tg_y\ \right)\right) \qquad (12)$$

Dove:

- SP_x e SP_y sono i setpoint che saranno mandati al modello dell'ambiente
- Pw_x e Pw_y sono le posizioni dell'end effector
- Tg_x e Tg_y sono i setpoint definiti dall'utente

Si usano anche vincoli logici temporali per "guidare" la simulazione, limitando o definendo il comportamento di alcune variabili durante la simulazione.

Un vincolo ovvio è sul setpoint, che possiamo all'inizio lasciare costante in questo modo:

$$Alw\left(\ SP_x = 0 \wedge SP_y = 2\right) \qquad (13)$$

Nella seconda simulazione possiamo aggiungere alcuni cambiamenti riguardanti il setpoint, dove fino ad 20 t.u. il setpoint definito dall'utente sarà (0,2) e dopo per altri 20 t.u. il setpoint sarà (1,0):

$$Lasts\left(\left(SP_x = 0 \wedge SP_y = 2\right), 20\right) \wedge Futr\left(Lasts\left(\left(SP_x = 1 \wedge SP_y = 0\right), 20\right), 20\right) \quad (14)$$

Ovviamente se si sono usate le formule (11) e (12) , la formula (14) deve essere modificata mettendo le variabili Tg_x e Tg_y al posto di SP_x e SP_y.

Altri vincoli necessari saranno presentati quando necessario, come per esempio, per limitare l'azione della condizione *DontMove,* altrimenti si verificherà la possibilità del software scegliere questa condizione come verdadeira sempre e ovviamente il robot non si muove.

Allora se definiamo che questa condizione deve essere presente solo se il braccio è arrivato al target o se una persona/ostacolo è avvertita vicina al braccio del robot, avremo la seguenti condizione per esempio:

$$Alw\left(DontMove \leftrightarrow (OnTarget \vee PersonDetected)\right) \quad (15)$$

A secondo dalla necessità si saranno definite sucessivamente queste regole.

3.2.2. Vincoli su sistema con più bracci

Quando si mettono più bracci insieme si devono aggiungere altri vincoli per controllare l'iterazione dei bracci nello stesso spazio di lavoro. L'obiettivo è evitare infatti una collisione dei bracci, quindi è necessario trovare qualche algoritmo per calcolare le distanze tra di essi.

Attraverso i valori di queste distanze, possiamo definire le prossime regole che esprimono l'idea seguente: quando una data distanza è minore di una tolleranza i bracci si bloccano semplicemente, dove $dist_{12}$ è la distanza tra i bracci 1 e 2, $DontMove_1$ e $DontMove_2$ sono rispettivamente la condizione *DontMove* dei bracci 1 e 2:

$$Alw\left(dist_{12} < dist_{min} \rightarrow (DontMove_1 \wedge DontMove_2)\right) \quad (16)$$

4. Prove Sperimentali

4.1. Singolo braccio

Per le prove con un solo braccio sono state esaminate due situazioni di setpoint differenti, una dove cambia solo θ_1 mentre $\theta_2$rimane costante, e l'altra dove entrambi gli angoli cambiano e i controllori devono lavorare insieme.

Ricordando che la posizione iniziale P è P=(2,0), le prove scelte sono nella tabella 4.1.

Tabella 4.1. – Prove scelte per valutare le prestazioni con le regole sul guadagno

Target	**Setpoint (px,py)**	$\boldsymbol{\theta_1}$**(°)**	$\boldsymbol{\theta_2}$**(°)**
1	(0,2)	90	0
2	(0,1)	150	-120

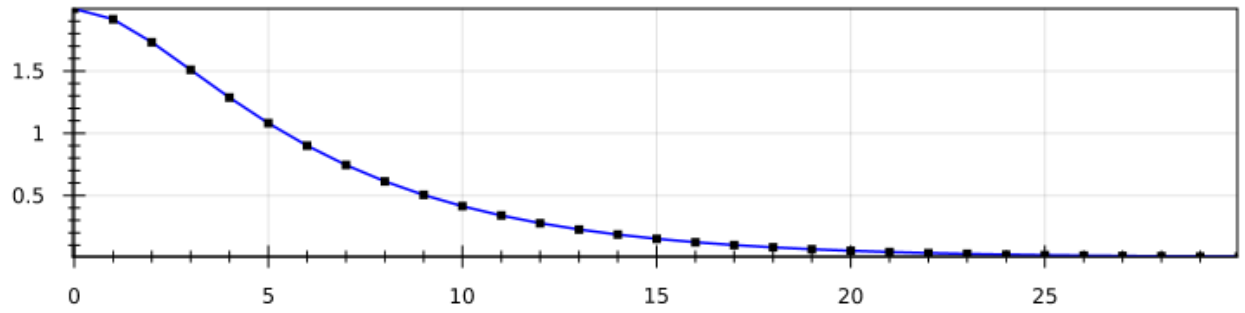
4.1.1. Prove senza controllo

Per testare la buona funzionalità del software e per ottenere i dati che serviranno in seguito per il confronto, si è dovuta eseguire una prova "senza" controllo, cioè solo con il controllo interno al braccio, senza aggiungere modifiche.

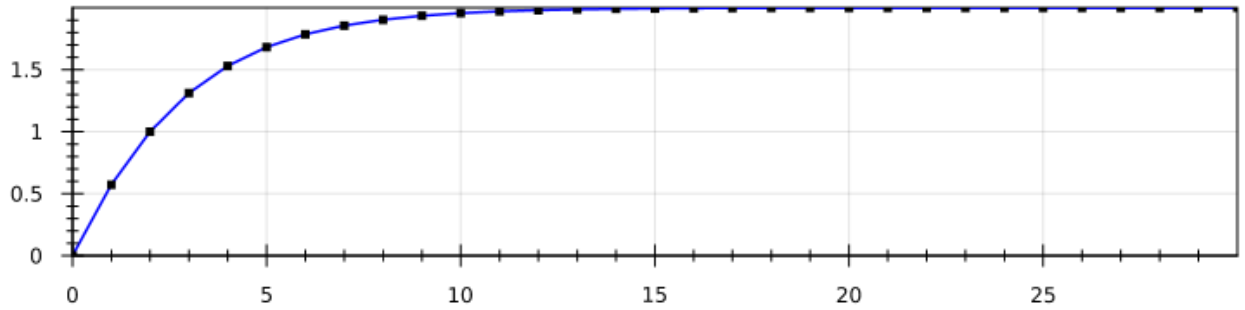
L'unica regola logica impostata, per una prima prova col 1°Target, è la seguente:

$$Alw\left(SP_x = 0 \wedge SP_y = 2 \wedge g_1 = 1 \wedge g_2 = 1\right) \qquad (17)$$

I grafici 4.1 e 4.2 mostrano i risultati di questa prima simulazione, che è stata effettuata con un periodo di campionamento di 1s per un periodo di 30 t.u. In particolare, il grafico 4.1.(a) mostra la coordinata x e il grafico 4.1.(b) mostra la coordinata y, entrambe del punto Pw (end effector).



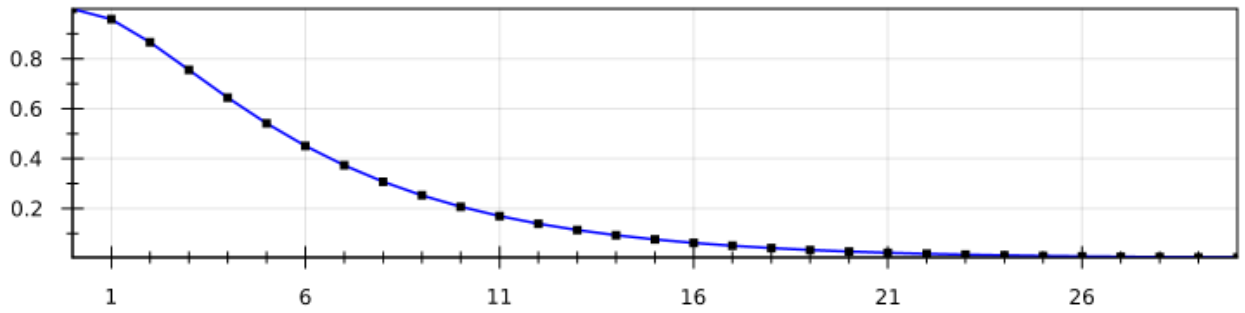
(a)



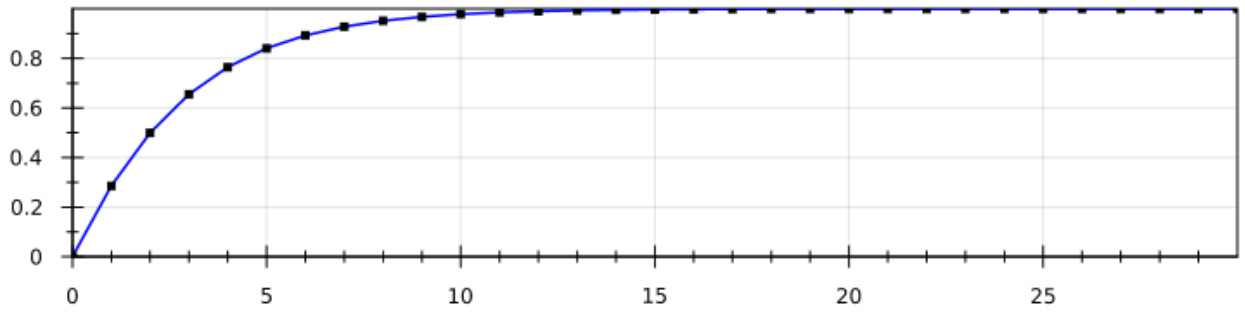
(b)

Grafico 4.1. – Posizione dell'end effector (Pw) – "x" nella figura (a) e "y" nella figura (b)

È importante tenere conto anche delle posizioni dei giunti, perchè è possibile che l'end effector sia già in una posizione stabile ma che i giunti presentino comunque delle oscillazioni, e non possiamo considerare questa condizione come una situazione stabile. Quindi per evitare ciò occorre guardare anche il grafico della posizione del giunto.



(a)

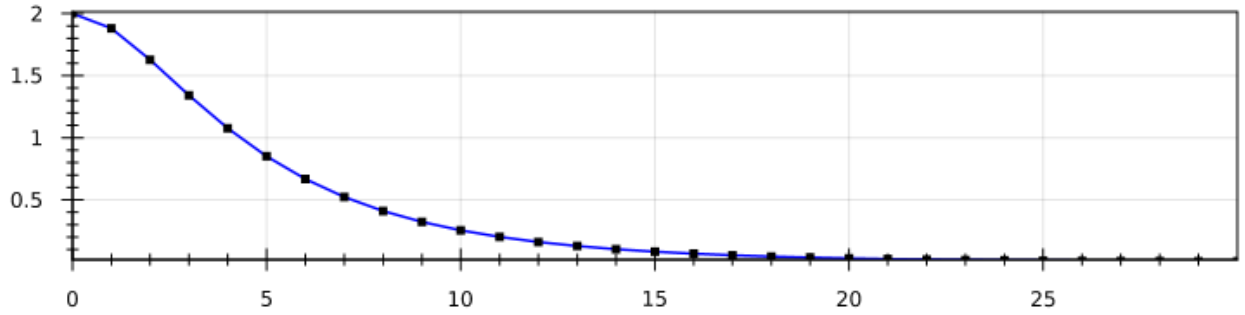


(b)

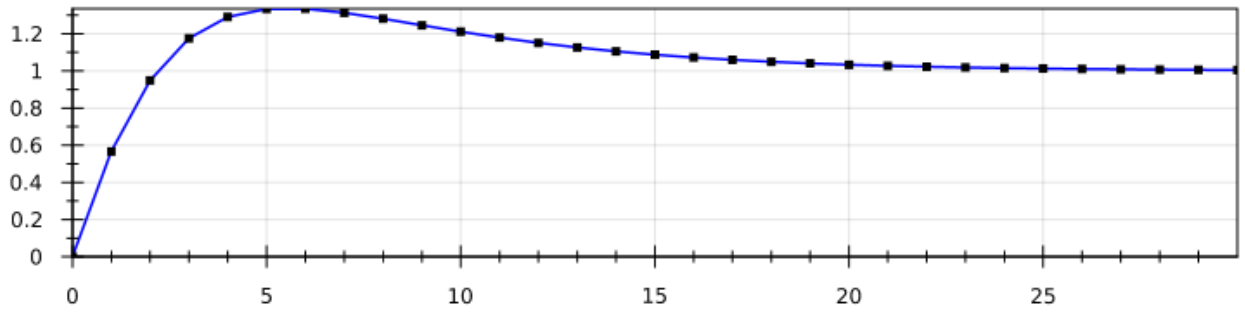
Grafico 4.2. – Posizione del giunto (Pj2) – “x” nella figura (a) e “y” nella figura (b)

Si nota che il tempo di assestamento della posizione di Pw e di Pj2 è praticamente lo stesso, circa 20 secondi. È un risultato aspettato perché il sistema ha un solo giunto che potrebbe oscillare quando Pw è già fermo, e questo è cinematicamente impossibile.

Le prossime figure mostrano i risultati della simulazione effettuata col 2° Target, che ci porta a verificare la robustezza del controllo. Come la precedente, anche questa simulazione è stata effettuata con un periodo di campionamento di 1s per un periodo di 30 t.u.

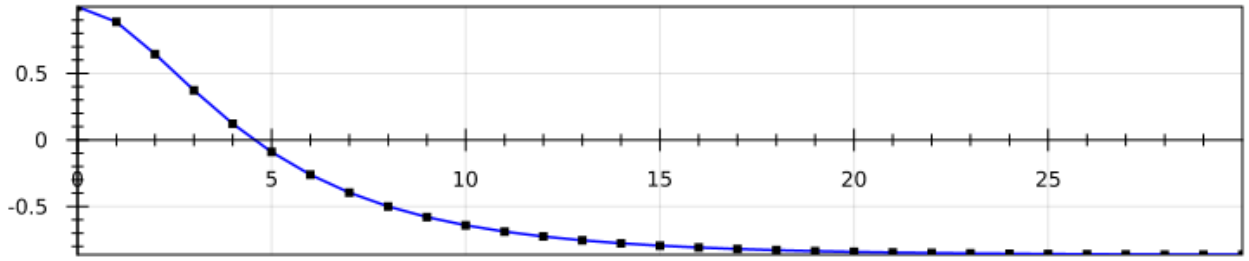


(a)

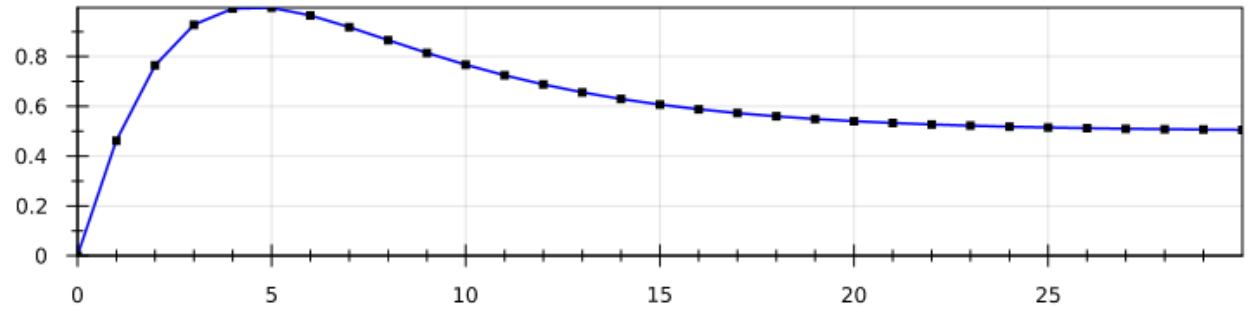


(b)

Grafico 4.3. – Posizione dell’end effector (Pw) – “x” nella figura (a) e “y” nella figura (b)



(a)



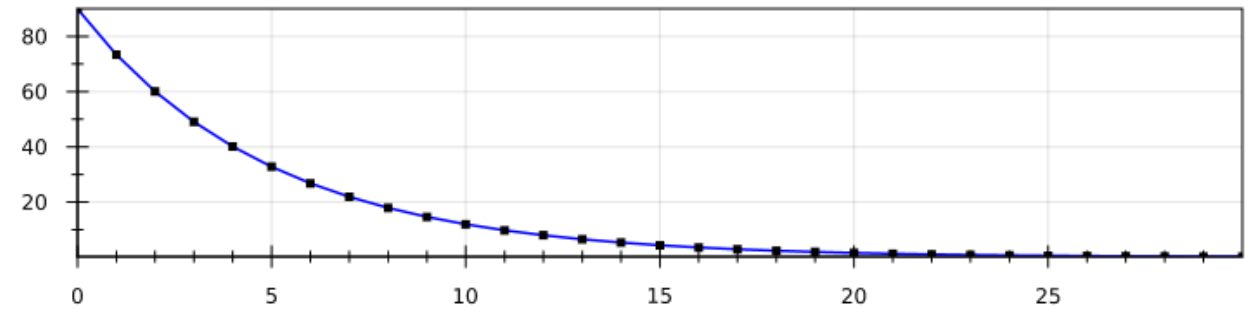
(b)

Grafico 4.4. – Posizione del giunto (Pj2) – “x” nella figura (a) e “y” nella figura (b)

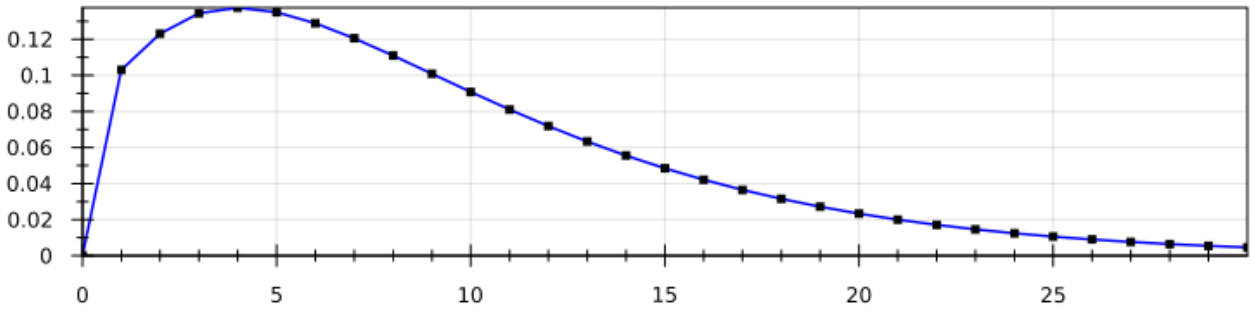
Per avere la conferma che il ragionamento presentato precedentemente sia giusto, si sono plottate di nuovo la posizione del giunto e si può notare come abbiano la stessa dinamica quando si confronta le risposte in x e in y.

Per il nostro sistema non bisogna dunque più guardare le posizioni dei giunti, e si vuole sottolineare l’importanza di questo ragionamento soprattutto per sistemi più complessi.

Si noti anche come il controllo sia buono e ben disaccoppiato, in quanto con questa prova (la quale prevede di attuare il controllo su entrambe le variabili) il sistema riesce ad avere quasi le stesse prestazioni. Le variabili y presentano una sovraelongazione dovuta al fatto che il sistema è controllato su θ_1 e θ_2, quindi se guardiamo queste variabili vediamo che il sistema rispetta le prestazioni. A questo punto, le variabili più importanti su cui focalizzare l’attenzione sono gli errori angolari, perché si conosce già che risultato ci si aspetta.

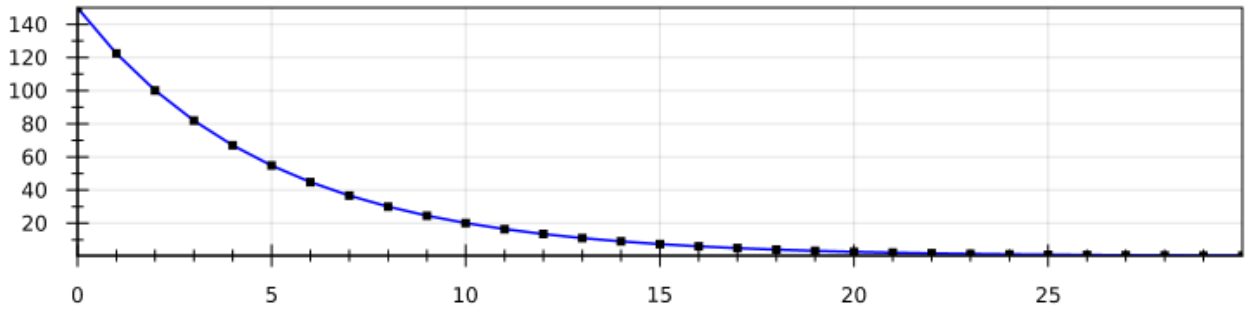


(a)

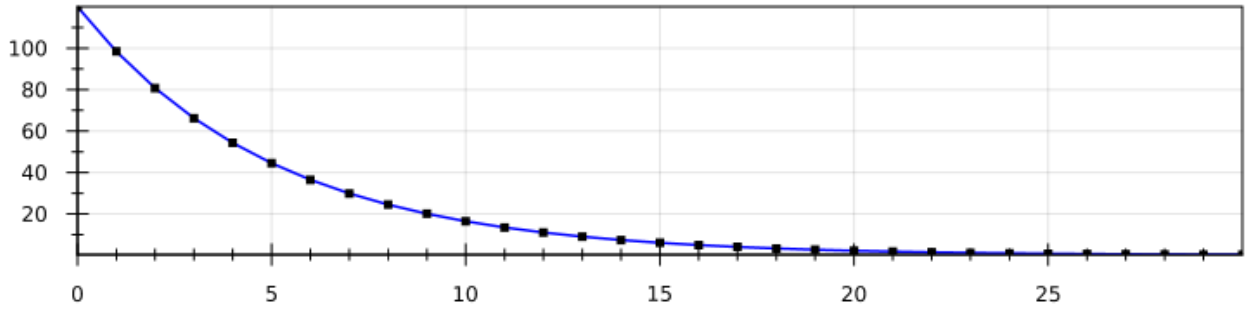


(b)

Grafico 4.5 – Errori angolari col 1° Target = (0,2) – (a) sul θ_1 e (b) sul θ_2



(a)



(b)

Grafico 4.6 – Errori angolari col 2° Target = (0,1) – (a) sul θ_1 e (b) sul θ_2

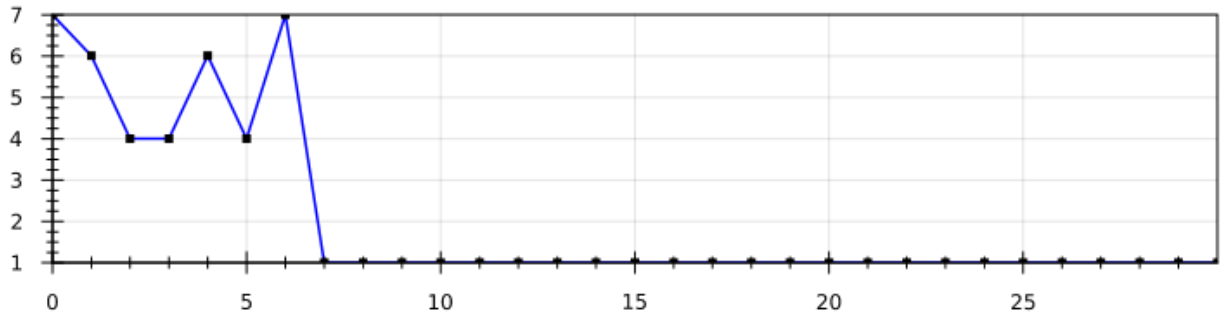
4.1.2. Prima simulazione: le regole logiche [1-4] sui guadagni

Per questa prova si sono impostate le prime regole che controllano i guadagni basati sulla variazione dell'errore. Nella definizione di queste regole si è lasciata libera scelta riguardo i valori dei guadagni limite. Sapendo che il massimo guadagno per il sistema è 10, a partire da questo si sono scelti progressivamente i valori degli altri guadagni:

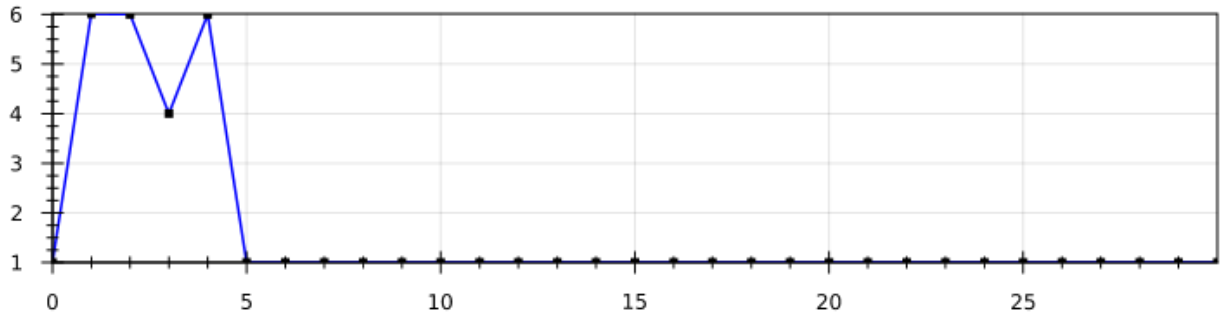
Tabella 4.2 – Valori dei guadagni limite

Guadagno	Valore
K_{max}	10
K_{gde}	6
K_{med}	4
K_{picc}	2
K_{min}	1

Per il test col 1° Target (2,0) ci si aspetta che il guadagno rimanga piccolo per il regolatore del giunto 2 e che cambi per il controllore 2. Come in precedenza, la simulazione è stata effettuata con un periodo di campionamento di 1s per un periodo di 30 t.u.

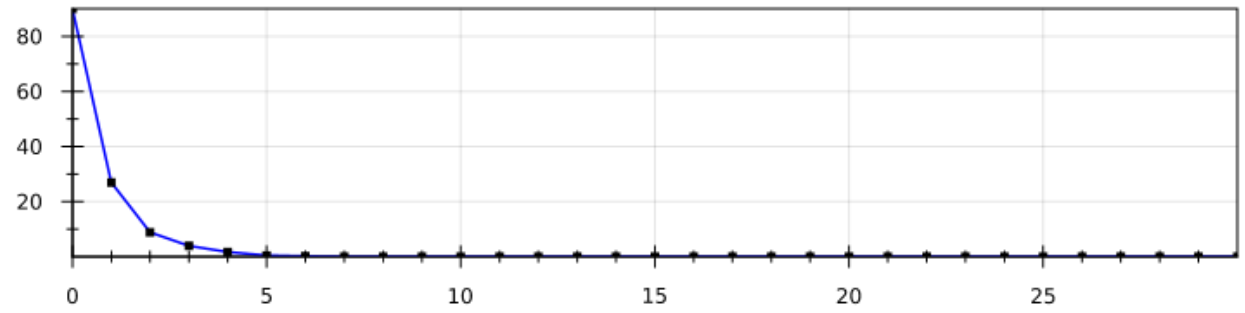


(a)

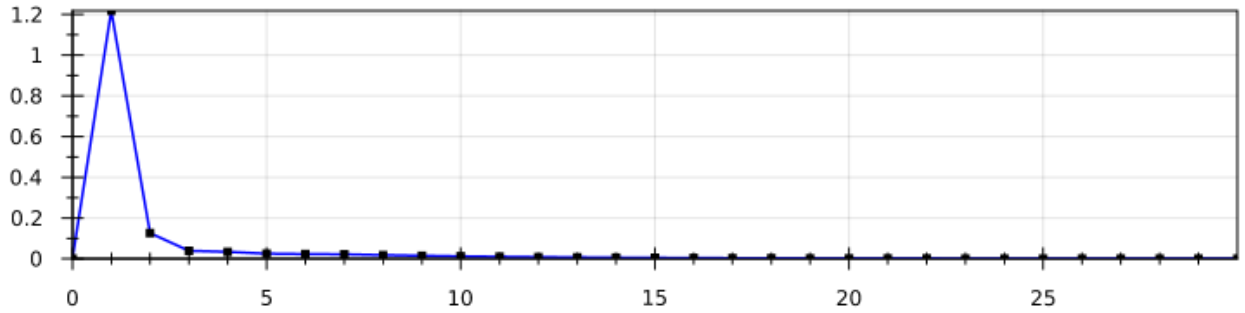


(b)

Grafico 4.7. – Guadagni dei regolatori – (a) regolatore 1 e (b) regolatore 2.



(a)



(b)

Grafico 4.8. – Errori angolari – (a) sul θ_1 e (b) sul θ_2

Si noti come sia migliorata di tanto la velocità di risposta del sistema, con un tempo di assestamento circa 5 secondi, prima che ci basassimo su queste regole.

Oltretutto, guardando i grafici dei guadagni, si riscontrano dei risultati non aspettati. Dal grafico del guadagno del secondo regolatore si vede che il guadagno non rimane piccolo nel range tra 1 e 2. Questo è dovuto al tempo di campionamento dell'errore definito prima come 5 t.u., perché in questo modo nei primi 5 t.u. il sistema non vede ancora che l'errore è piccolo, quindi sceglie un generico valore tra 1 e 10.

Se spostiamo l'attenzione sul grafico del guadagno di 1 vediamo che non si ottiene esattamente quello che ci si aspettava. Ad esempio, nell'istante 6 il guadagno è ancora troppo grande e l'errore è già piccolissimo.

A seguito di questo si può affermare come la preoccupazione iniziale riguardo l'effettuare troppi cambiamenti considerati inutili non sia necessaria. Possiamo così cambiare le regole logiche (1 – 4) togliendo l'operatore *Lasted:*

$Alw(err > 90 \rightarrow K_{gde} < g < K_{max})$ (1c)

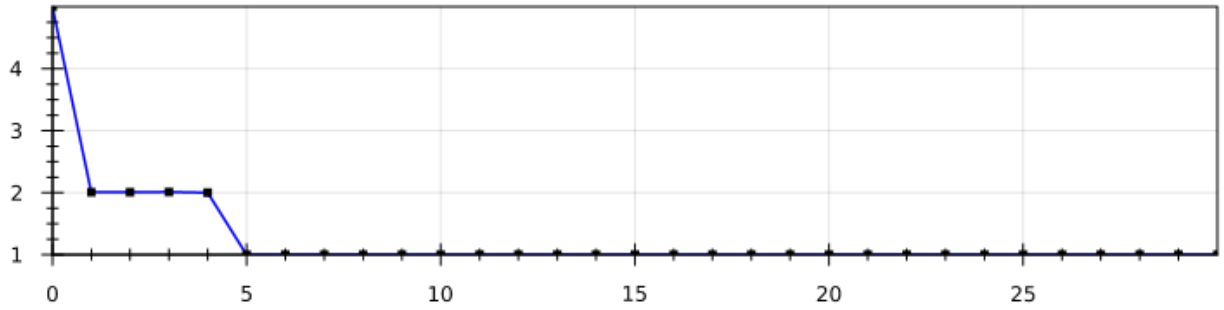
$Alw(60 < err < 90 \rightarrow K_{med} < g < K_{gde})$ (2c)

$Alw(10 < err < 60 \rightarrow K_{picc} < g < K_{med})$ (3c)

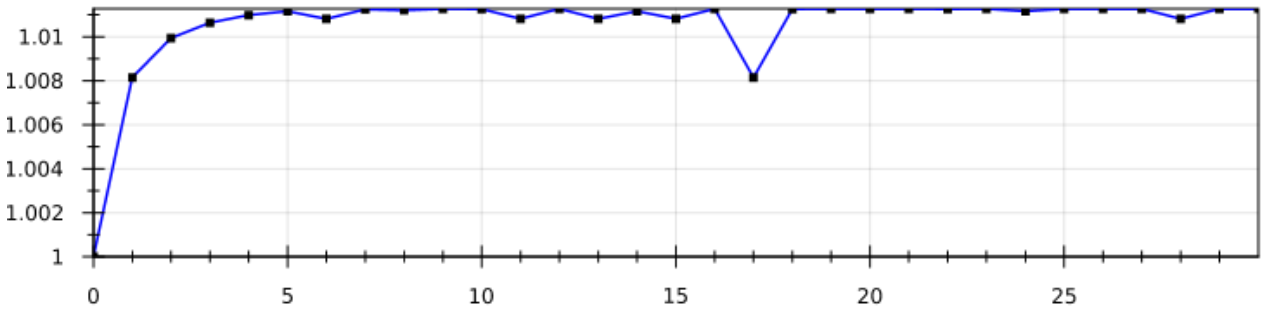
$Alw(err < 10 \rightarrow K_{min} < g < K_{picc})$ (4c)

Con questo accorgimento si risolvono i due problemi trovati, perché in questo modo il sistema si aggiusta velocemente per cambiare i guadagni.

Usando le nuove regole otteniamo le seguenti risposte, ancora con delta = 1s e 30 t.u.

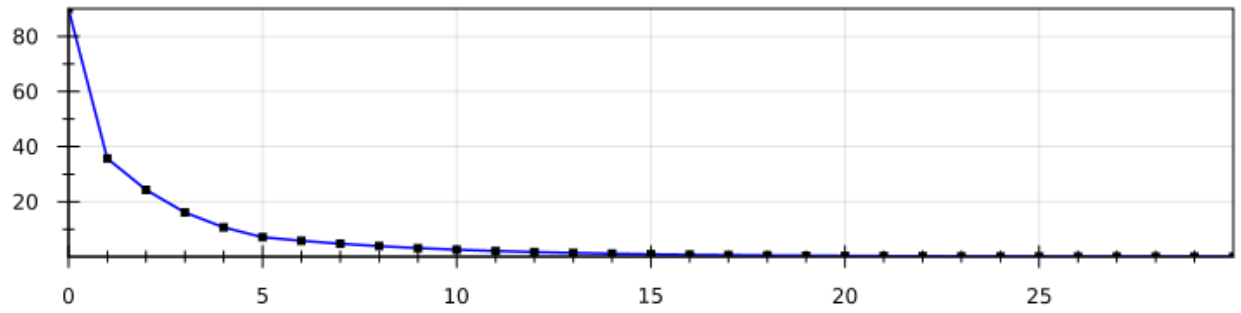


(a)

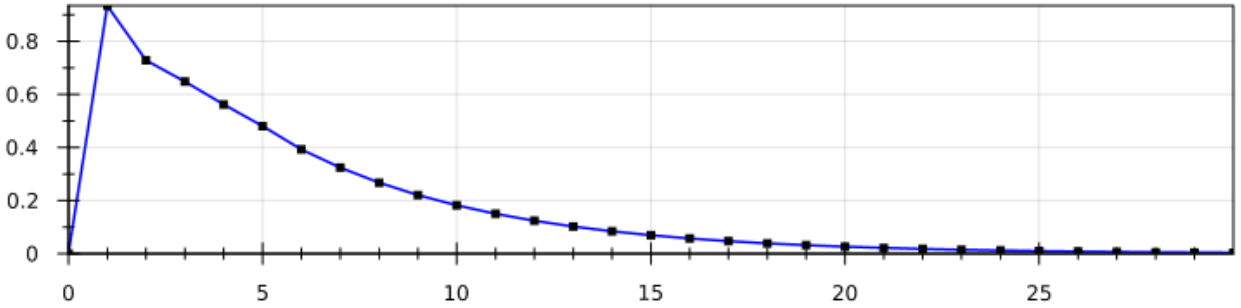


(b)

Grafico 4.9. – Guadagni dei regolatori – (a) regolatore 1 e (b) regolatore 2.



(a)

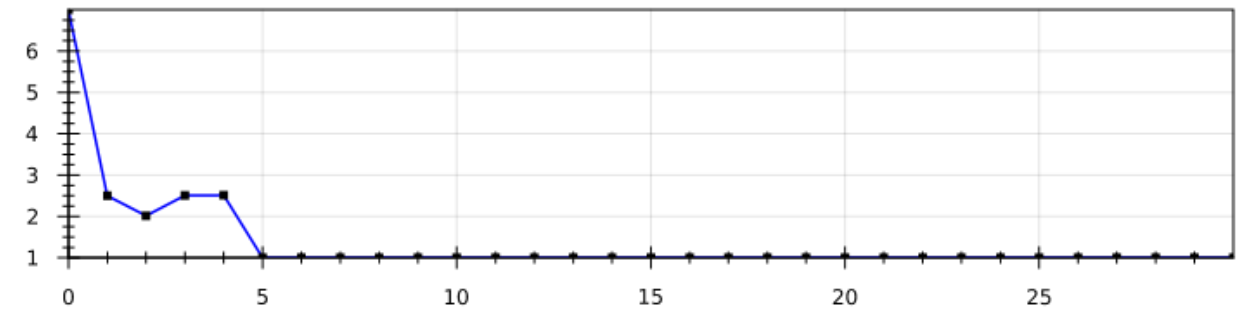


(b)

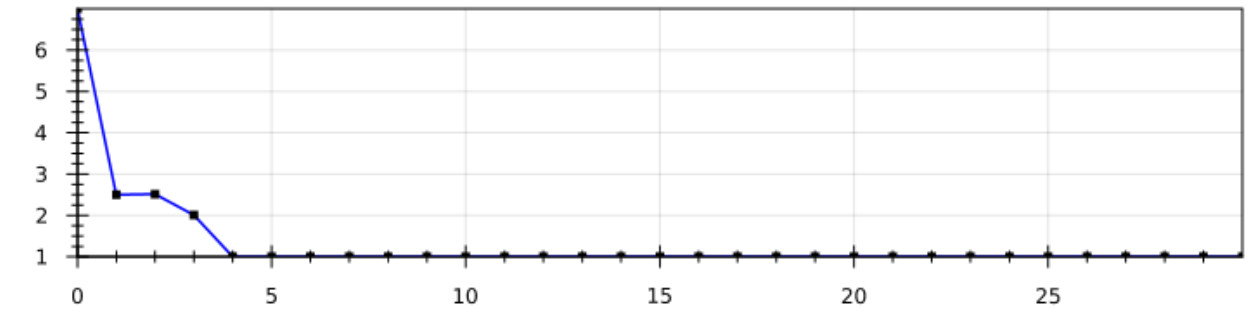
Grafico 4.10. – Errori angolari – (a) sul θ_1 e (b) sul θ_2

Con queste nuove regole non è cambiato di molto il tempo di assestamento ottenuto, ma ora si rispettano gli obiettivi della logica sui guadagni.

Un ultimo aspetto importante è verificare che i controllori riescano a lavorare disacoppiati, e per questo si procede ad una simulazione con il Target =(1,0).

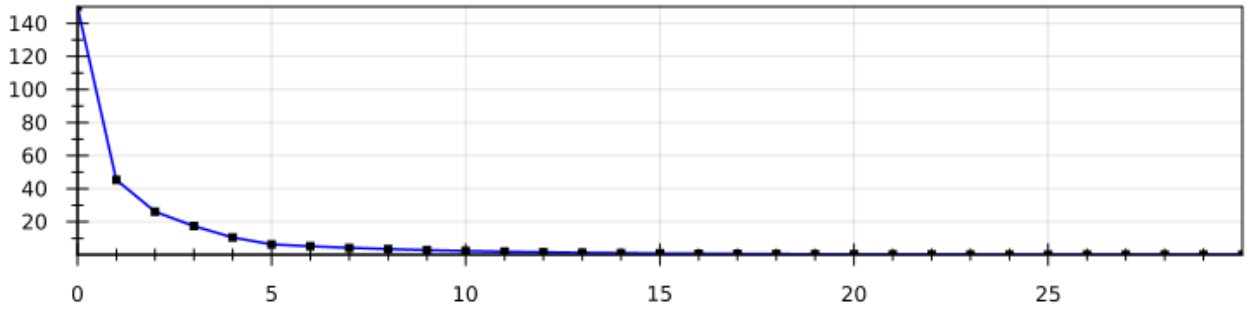


(a)

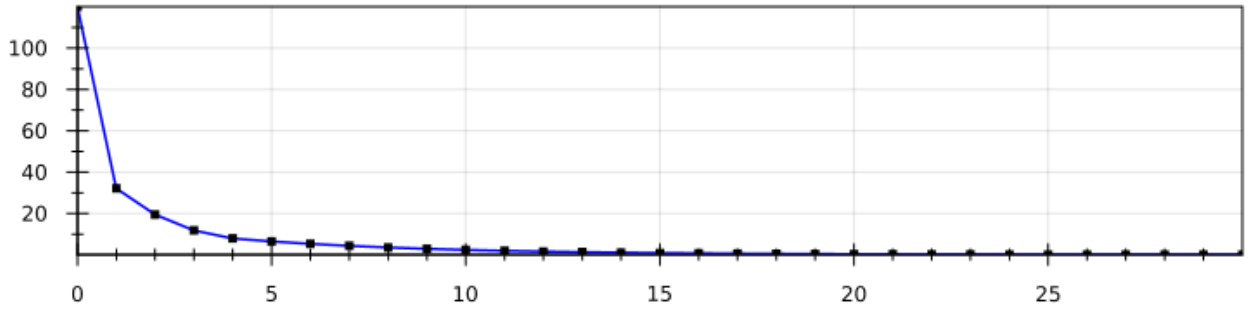


(b)

Grafico 4.11. – Guadagni dei regolatori – (a) regolatore 1 e (b) regolatore 2.



(a)



(b)

Grafico 4.12. – Errori angolari – (a) sul θ_1 e (b) sul θ_2

Questa simulazione conferma che i regolatori riescono a lavorare bene indipendentemente l'uno dall'altro e quindi non bisogna più eseguire prove per più di un Target separatamente.

4.1.3. Seconda simulazione: aggiungendo le regole per PersonDetected

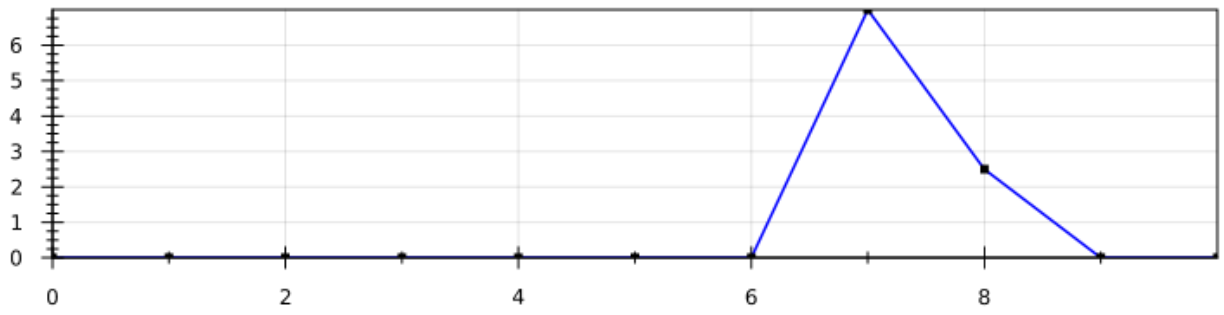
Per effettuare questa simulazione si sono scelte le seguenti regole “guide”:

- Target costante per tutto il periodo di simulazione : $Alw(SP_x = 0 \wedge SP_y = 1)$.
- il sensore rileva la presenza di una persona nell’istante 2 t.u. e il robot resta fermo per 5 t.u.

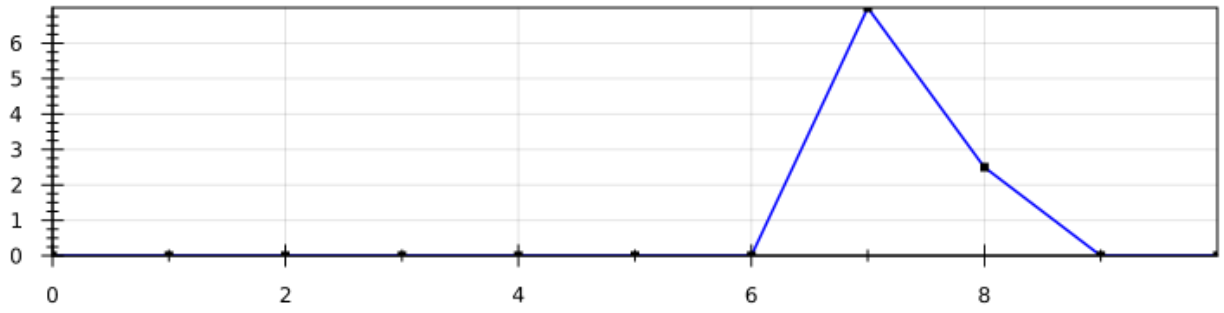
Usando questa regola, sommata alle regole (1c-4c) e (5), otteniamo le risposte riportate sotto, avendo impostato ancora delta = 1s ma eseguendo la simulazione solo fino ad 10 t.u. (dato che il sistema è già a regime a 5 s).

Si deve decidere a questo punto quale regola *DontMove* è meglio usare, la (10) o la (11-12). Per compiere la scelta migliore, si è deciso di eseguire le simulazioni con entrambe le regole e confrontare i risultati. Nella prima simulazione riportata si è scelta la regola (10), che è la più facile da applicare, con l’aggiunzione delle modifiche fatte (1b-4b), e successivamente si confronterà con le regole (11 e 12).

Inizialmente si riproducono i grafici dei guadagni per guardare se le regole funzionano come ci si aspetta.



(a)



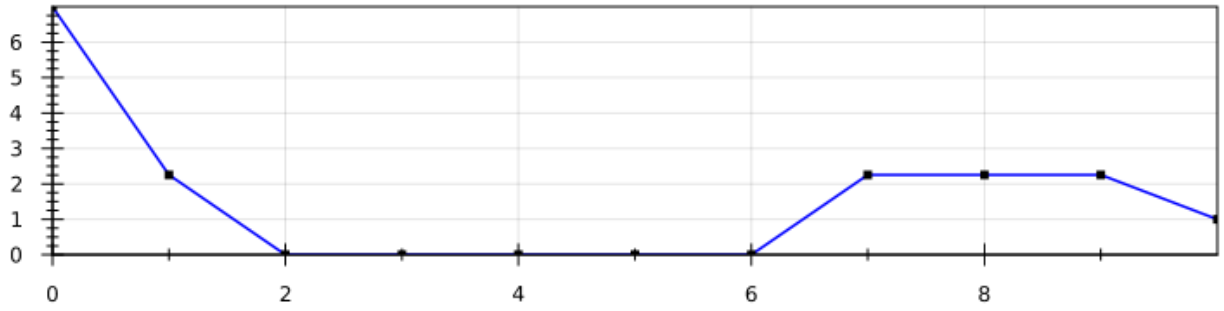
(b)

Grafico 4.13. – Guadagni dei regolatori – (a) regolatore 1 e (b) regolatore 2.

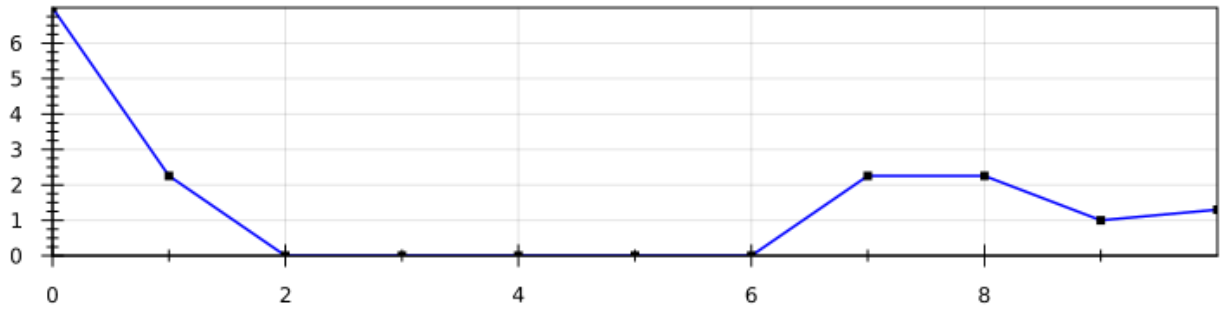
Guardando il grafico 4.13 si riscontrano dei risultati inaspettati. Si nota che il segnale logico *DontMove* è presente negli intervalli da 0 t.u. a 6 t.u. e da 9 t.u. a 10 t.u., quando doveva essere rilevato solo nell'intervallo da 2 t.u. a 6 t.u. secondo la regola logica (5). Questo è dovuto al fatto che la regola non impone che il segnale *DontMove* accada solo dopo una rilevazione di "persona vicina". Per una condizione più esauriente dobbiamo aggiungere la seguente regola:

$$Alw\left(DontMove \rightarrow WithinPast\left(PersonDetected, T_{person}\right)\right) \qquad (18)$$

Essa impone che il segnale *DontMove* può occorrere solo se in alcuni degli ultimi T_{person} istanti è avvenuto *PersonDetected*. Con l'aggiunta di questa regola, otteniamo i seguenti risultati:

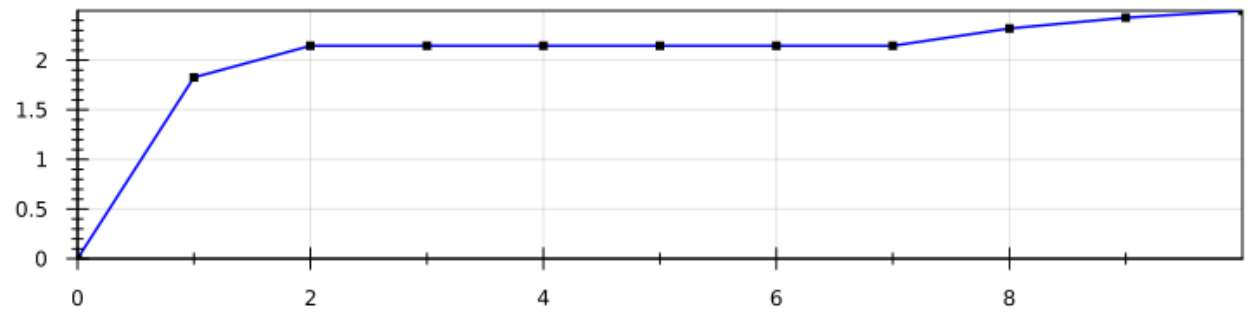


(a)

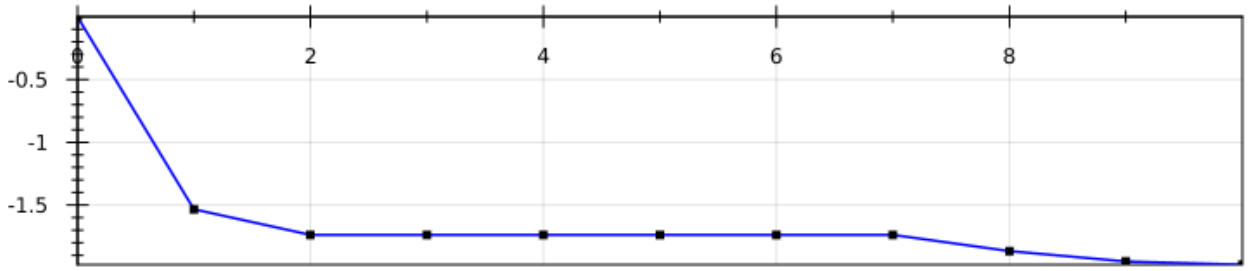


(b)

Grafico 4.14. – Guadagni dei regolatori – (a) regolatore 1 e (b) regolatore 2.



(a)



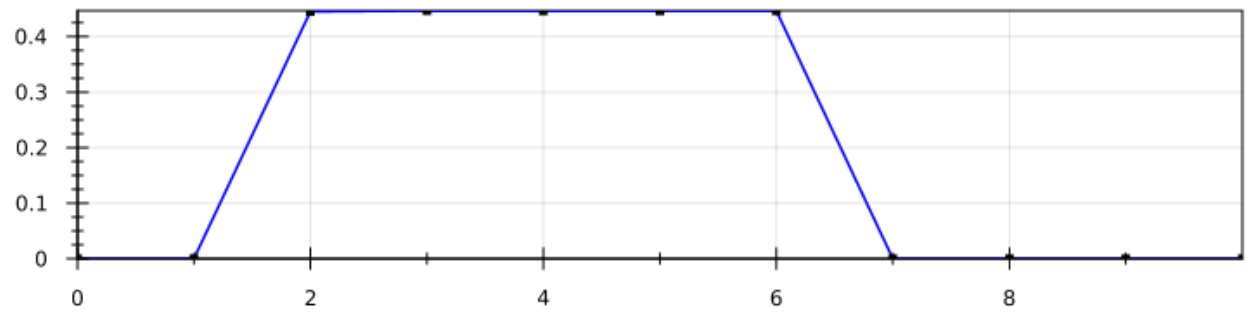
(b)

Grafico 4.15. – Posizione angolari – (a) sul θ_1 e (b) sul θ_2

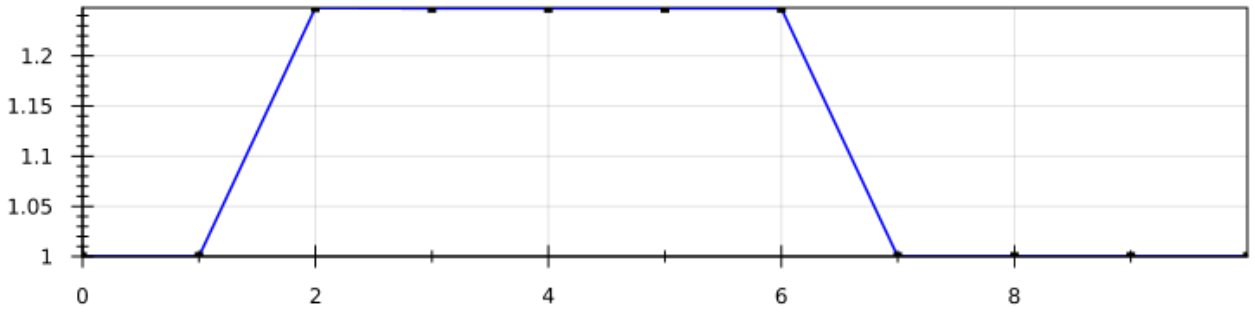
Dai grafici vediamo che adesso il risultato è congruente a ciò che ci si aspettava, in quanto i guadagni si annullano tra 2 t.u. e 6 t.u., e il sistema rimane fermo per 5 t.u. istanti. Come detto in precedenza, occorre ora confrontare questi ultimi risultati con una prova relativa alla regola che definisce un modo alternativo per fermare il braccio, secondo le formule (11) e (12).

A questo punto per sapere se la regola sta funzionando non serve più guardare i grafici dei guadagni ma quelli dei setpoint.

Otteniamo quindi le seguenti risposte, ancora con delta = 1s e periodo 10 t.u.

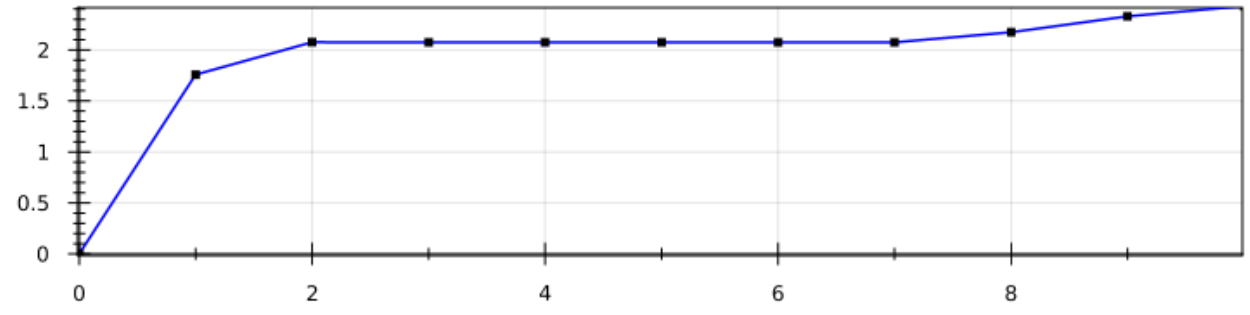


(a)

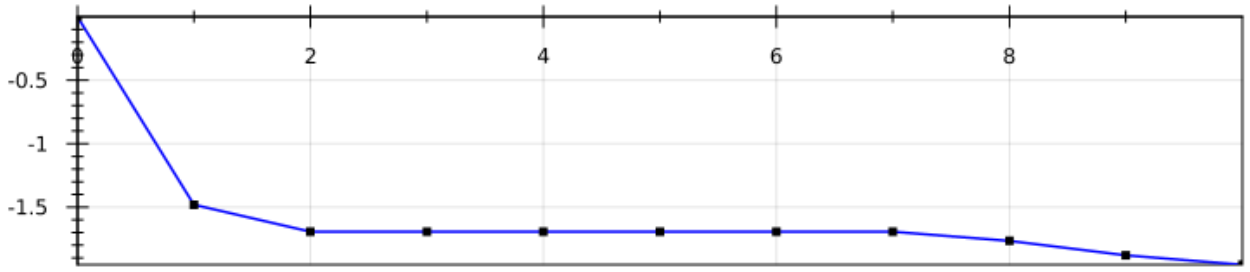


(b)

Grafico 4.16. – Setpoint del sistema - (a) coordinata x e (b) coordinata y.



(a)



(b)

Grafico 4.17. – Posizioni angolari – (a) sul θ_1 e (b) sul θ_2

Vediamo dal grafico 4.15 che la regola funziona bene, e confrontando i grafici 4.14 e 4.16 si nota che sono proprio uguali, quindi le regole riportano gli stessi risultati.

A parità di risultati ottenuti, si riscontra uno svantaggio nell'usare la prima regola, che è quella più facile da applicare. Annullare il guadagno di un regolatore, infatti, è generalmente una pratica sconsigliata, dato che è difficile da applicare a sistemi reali.

D'altra parte la seconda regola comporta un vantaggio che rimanda a situazioni ancora più interessanti. Ad esempio, invece di bloccare il braccio quando viene rilevata una persona, il braccio può effettuare piccoli spostamenti, oppure spostarsi nell'altro senso. Questo è un punto interessante, perché servirà in seguito per controllare sistemi costituiti da più bracci robotici.

Come conseguenza a quanto appena affermato, d'ora in poi si userà sempre la seconda regola.

4.1.4. Terza simulazione: aggiungendo le regole per fermarsi nel target

Da questa simulazione in poi si è notato come sia meglio modificare la posizione iniziale del sistema e di conseguenza i target definiti.

Questa idea è sorta in seguito al seguente fatto:

- Supponiamo che il target sia un oggetto che il robot deve prendere
- La cinematica inversa comporta l'esistenza di due configurazioni possibili per arrivare al target
- La configurazione scelta come standard è quella dove θ_2 è negativo, cioè che l'angolo assoluto del giunto 2 è sempre minore di quello del giunto 1.
- Come conseguenza di questo fatto, se un oggetto si trova nel punto (1,0) e il robot parte dal punto (0,2), il robot collide sempre con l'oggetto.

Supponendo quindi che questa restrizione su $\theta_2$sia dovuta alla struttura fisica del robot e che lavori sempre nel primo quadrante, si sono modificati la condizione iniziale e il target desiderato. La nuova condizione iniziale è con il braccio sull'asse y, cioè, la posizione iniziale dell'end effector è (0,2).

Per effettuare la simulazione si è scelta la seguente regola sul setpoint per “guidare” la simulazione:

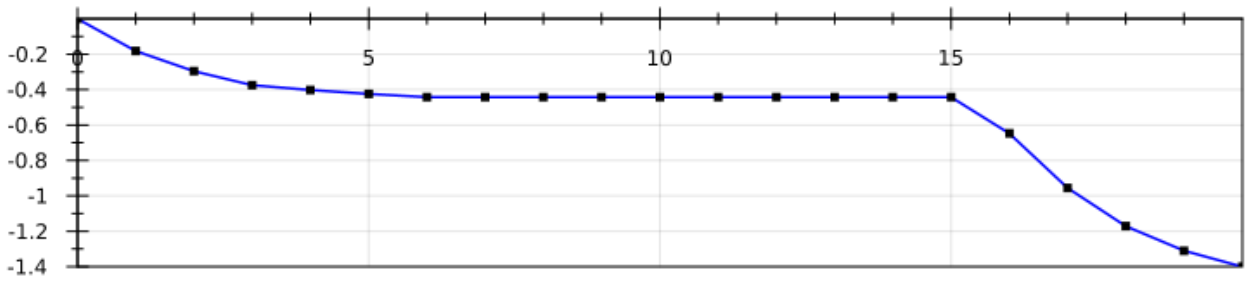
$$Lasts\left(\left(SP_x = 1 \wedge SP_y = 0\right), T_{set}\right) \wedge Futr\left(Lasts\left(\left(SP_x = 2 \wedge SP_y = 0\right), 20\right), T_{set}\right) \quad (19)$$

Il tempo T_{set} è stato scelto in maniera da poter vedere che, con la regola del target, il sistema non risponde istantaneamente ma solo dopo di 10 t.u.

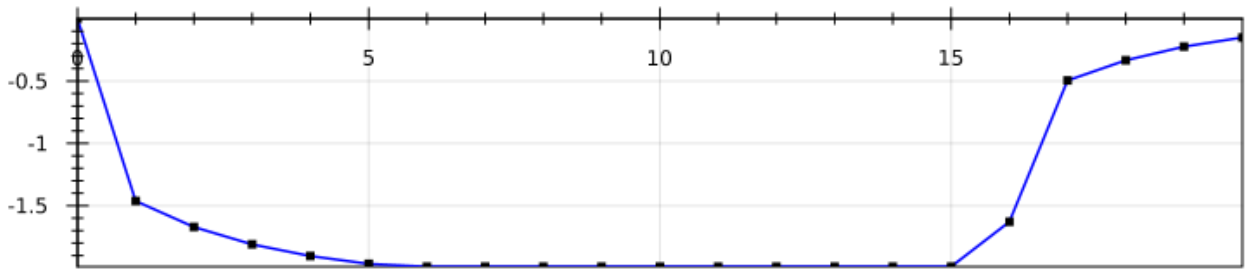
Si aggiunta pure una regola analoga a (18) del seguente modo:

$$Alw\left(DontMove \rightarrow WithinPast\left(OnTarget, T_{ontarget}\right)\right) \quad (18)$$

La regola di *DontMove* scelta è la seconda (11-12) come giustificato prima e la simulazione è eseguita con un delta=1s e tempo di 25 t.u.



(a)



(b)

Grafico 4.18. – Posizione angolari – (a) sul θ_1 e (b) sul θ_2

La regola funziona bene, però non rispetta esattamente la prestazione che si vuole. Quello che si vuole è che se il braccio è arrivato al Target, rimanga T_{Target} istanti fermi, per poter

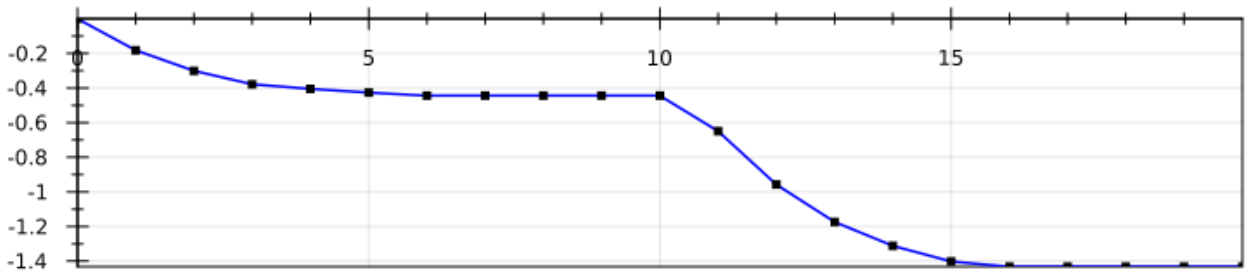
per esempio chiudere il braccio su un oggeto, ma dopo di questo che possa iniziare subito il movimento ad un nuovo Target necessario.

Nel caso simulato il sistema è arrivato al Target nell'istante 5 t.u. e riceve dopo un nuovo Target nell'istante 10 t.u., allora secondo la prestazione deve rimanersi fermo fino a 14 t.u., e dopo di questo può muoversi. Però quello che succede è che rimane fermo fino a 20 t.u., perchè la regola alla fine define che il sistema rimanga fermo 10 t.u. dopo di un cambiamento di Target, e questo non è quello che si vuole.

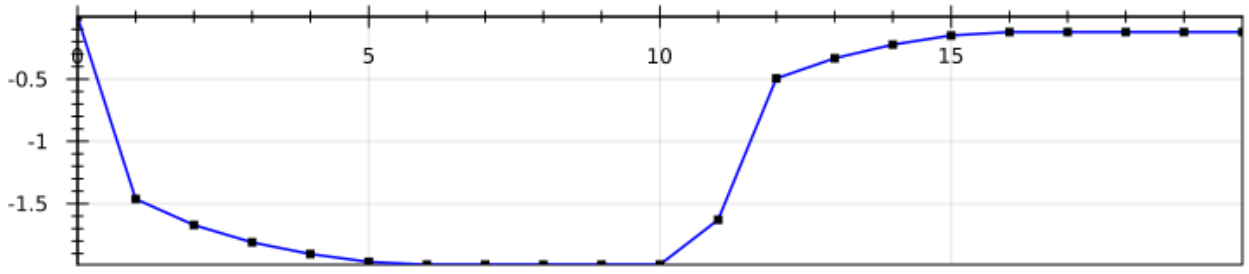
Il problema è che il parametro *OnTarget* rimane attivo fino al cambiamento di Target, cioè, dal 5 t.u. al 10 t.u. e quindi sodisfacendo la regola (6) fino a questo istanti, quando doveva soddisfarla solo nell'istante 5 t.u. Per risolvere questo si sono messe le seguente altre regole per far che il segnale *OnTarget* genere il predicato solo nell'istante subito dopo l'arrivo al Target.

$$Alw\big(Past(\neg OnTarget, 1) \wedge OnTarget \rightarrow Lasts(DontMove, 5)\big) \qquad (6b)$$

Con queste nuove regole, la nuova simulazione risulta:



(a)



(b)

Grafico 4.19. – Posizione angolari – (a) sul θ_1 e (b) sul θ_2

Il sistema funziona come ci si aspettava e a questo punto si procede affrontando un problema più difficile, che consta nel mettere più bracci robotici a lavorare insieme nello stesso spazio operativo.

4.2. Sistema con 2 bracci che lavorano insieme

Per i sistemi con più bracci robotici posizionati nello stesso spazio di lavoro, i problemi sono legati al trovare un buon metodo per il calcolo della distanza tra di essi e successivamente al definire bene quali passi compiere, relativi agli stessi bracci.

Si sono provati alcuni metodi per calcolare la distanza tramite simulazioni in situazioni critiche, dove i bracci collliderebbero sicuramente senza nessuna regola logica che glielo impedisca.

Le prove saranno fate per 2 bracci, ma possono facilmente essere espanse per più bracci. Il primo braccio ovviamente avrà la sua origine locale nella origine dello spazio di lavoro (0,0) e il secondo braccio sarà nella posizione (1.5,0). Entrambi bracci iniziano nella posizione verticale, cioé, Pw1 (Posizione del end effector del primo braccio) è inizialmente (0,2) e Pw2 (Analogamente, posizione dell'end effector del secondo braccio) è (1.5,2)

4.2.1. Primo tentativo

In questo primo tentativo, il calcolo dalla distanza tra i bracci è stato semplicemente implementato attraverso il calcolo della distanza tra le coordinate degli end effector di ogni singolo braccio.

Per provare la funzionalità di questo metodo per il calcolo della distanza si deve imporre Targets che sappiamo che succede una collizione.

Una prima possibilità è fare che il primo braccio abbia come Target (2,0) e il secondo un Target di (-0.5,0).

Nella figura 4.1. ricostruiamo le traiettorie che sarebbero compiute dai bracci se non fossero stati messi nello stesso spazio di lavoro. Il primo braccio e la sua traiettoria sono rappresentati in azzurro e il secondo braccio in rosso.

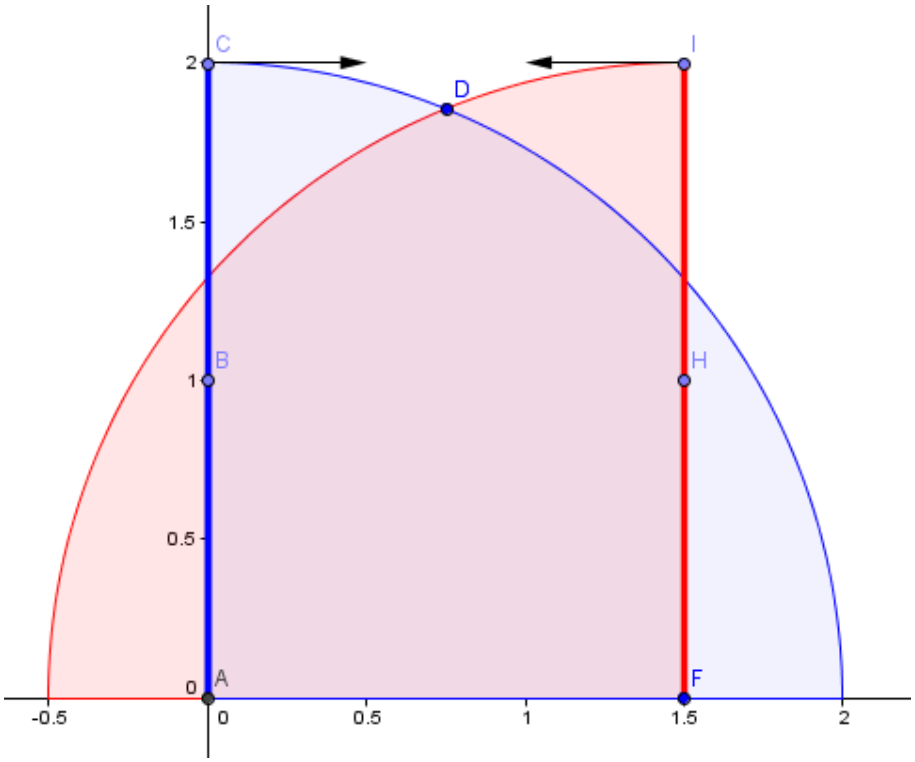
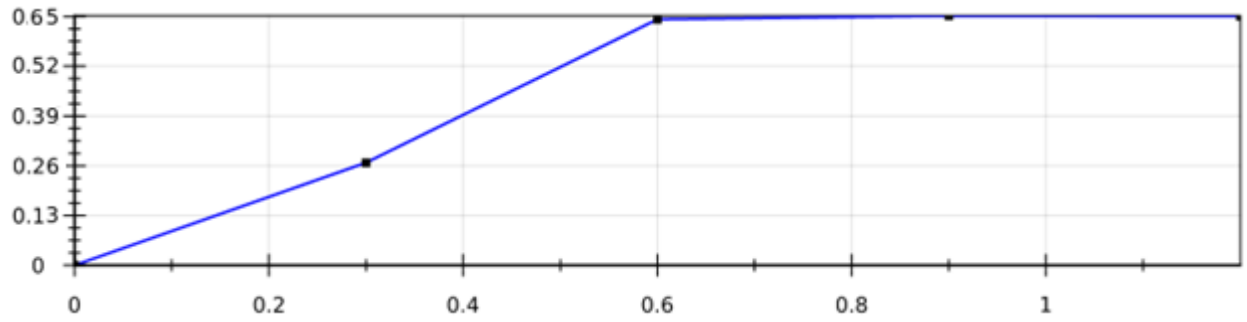


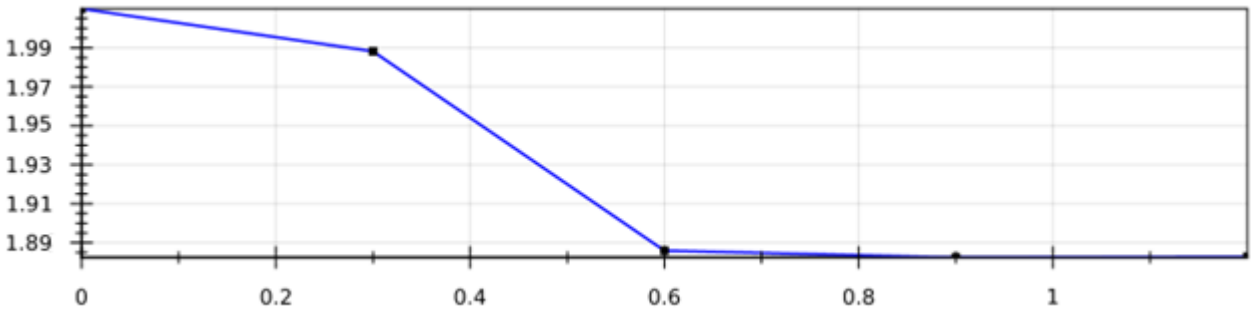
Figura 4.1. – Traiettoria dei bracci e punto di collisione D

Dalla intersessione delle traiettorie abbiamo che il punto di collisione sarebbe D, con coordinate di circa (0.75,1.85). Quindi si aspetta che dalla simulazione con la regola per evitare collisioni, che le posizioni rimangono vicine a questi valori.

Facendo la simulazione con un delta di 0.3s (più piccolo perchè in 1 secondo il sistema sarebbe già dopo della posizione di collisione) e periodo di 1.2s, si ottengono i risultati mostrati sotto.



(a)



(b)

Grafico 4.20. – Posizione dell'end effector (Pw1) del primo braccio – "x" nella figura (a) e "y" nella figura (b)

Da questi grafici vediamo che da 0.6s in poi il braccio 1 resta fermo nella posizione (0.65,1.89), quindi circa 0.1 di distanza dal possibile punto di collisione.

Il seguente grafico mostra come varia la distanza calcolata tra i bracci e si vede che dopo 0.6s rimane ferma in circa 0.2.

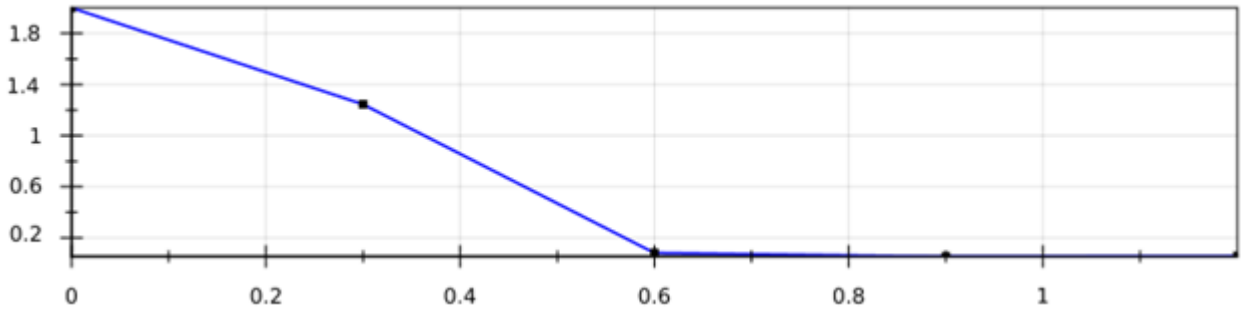


Grafico 4.21. – Valore della distanza calcolata

In questa prima prova il calcolo della distanzia con questo metodo va benissimo, però se proviamo una nuova situazione , muovendo solo il primo braccio e lasciando l'altro fermo, avremo pure un punto di collisione, però sarà nel mezzo del link, come si vede nella figura 4.2.

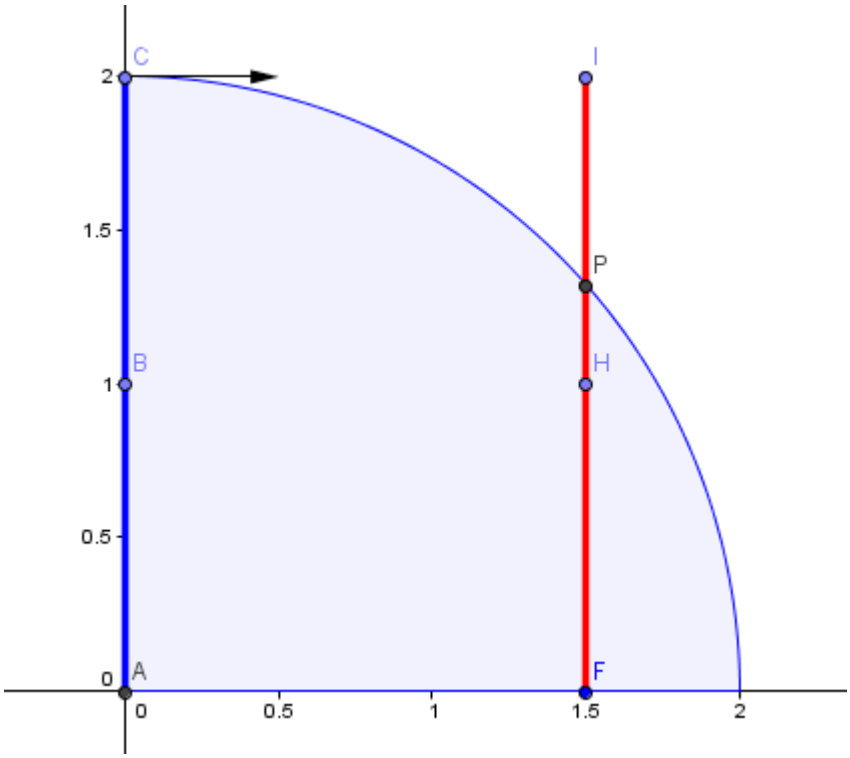
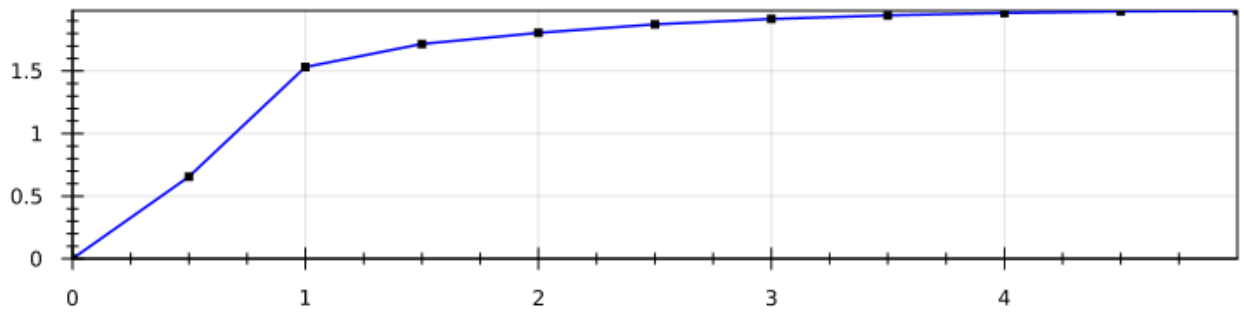


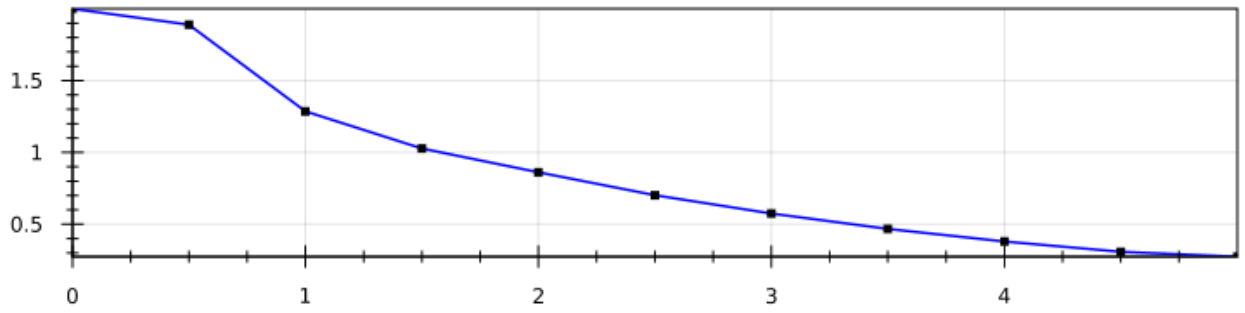
Figura 4.2. – Nuova triettoria dei bracci e punto di collisione P =(1.5,1.56)

In questa nuova situazione la regola logica (16) non riuscirà a fermare il braccio, perchè la distanza calcolata non sarà mai sufficientemente piccola.

Si vede nei grafici sotto che il braccio 1 non “sente” la presenza del secondo braccio e arriva al suo Target come se la simulazione fosse con solo il primo braccio, cioè, non ci blocca prima del punto P come doveva fare.



(a)



(b)

Grafico 4.22. – Posizione dell'end effector (Pw1) del primo braccio – "x" nella figura (a) e "y" nella figura (b)

Infatti la distanza calcolata tra i bracci rimane sempre sopra 0.4 perchè i punti Pw1 e Pw2 non si tocano, neanche si avicinano, allora il sistema non blocca il braccio.

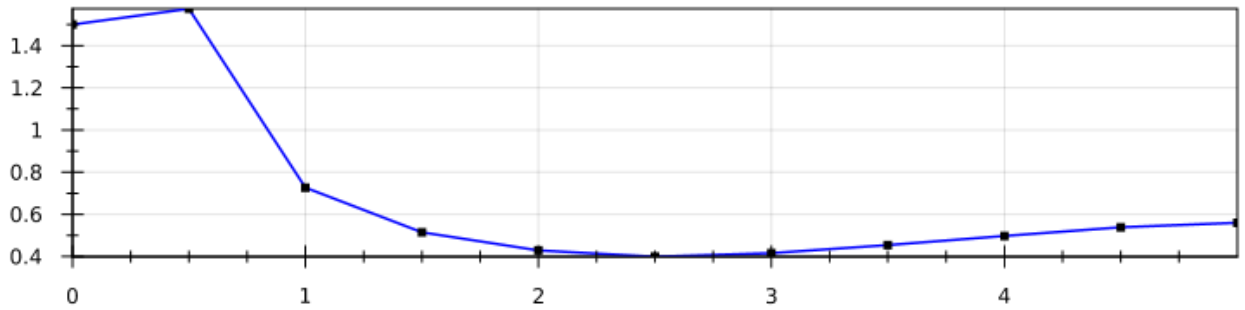


Grafico 4.23. – Valore della distanza al punto di collisione conosciuto

Per risolvere questo problema dobbiamo cambiare il modo per calcolare la distanza tra i bracci. Nella prossima simulazione proveremo un'altro modo, più ragionevole.

4.2.2. Distanza calcolata col metodo distanza punto-retta

Adesso cambiamo il modo per il calcolo dalla distanza tra i bracci, usando la nozione di distanza tra punto e retta. Il link 2 di ogni braccio costituisce un segmento di retta e quindi dalle coordinate di Pj2 e Pw possiamo calcolare la equazione di questa retta.

Quindi la distanza tra il braccio 1 e il braccio 2 sarà la distanza tra l'end effector di 1 e la retta definita per il link 2 del braccio 2.

Usiamo le seguente equazioni, dove Pw1 è la posizione dell'end effector del braccio 1, Pw2 è la posizione dell'end effector del braccio 2, Pj2 è la posizione della giunta del braccio 2.

$$d_{12} = \frac{abs\ (a.Pw1_x + b.Pw1_y + c)}{\sqrt{a^2 + b^2}}$$

$$a = Pw2_y - Pj2_y$$

$$b = -Pw2_x + Pj2_x$$

$$c = Pw2_x.Pj2_y - Pw2_y.Pj2_x$$

Analogamente si può ottenere queste formule per la distanza dall'end effector del braccio 2 al link 2 del braccio 1. Si definisce come la distanza il minimo valore tra questi due.

Seguendo questo stesso ragionamento appare la necessità di calcolare non sono queste 2 distanze, ma pure altre per evitare la collisione non solo tra l'end effector e il link 2 ma in tutto il braccio. La tabella 4.3 mostra tutte le distanze che saranno calcolate.

Tabella 4.3 – Distanze importanti tra i due bracci

Distanza	Punto	Segmento di reta
$\boldsymbol{d12_{PwL2}}$	Pw1	Link 2 del braccio 2
$\boldsymbol{d12_{PwL1}}$	Pw1	Link 1 del braccio 2
$\boldsymbol{d12_{PjL2}}$	Pj1	Link 2 del braccio 2
$\boldsymbol{d12_{PjL1}}$	Pj1	Link 1 del braccio 2
$\boldsymbol{d21_{PwL2}}$	Pw2	Link 2 del braccio 1
$\boldsymbol{d21_{PwL1}}$	Pw2	Link 1 del braccio 1
$\boldsymbol{d21_{PjL2}}$	Pj2	Link 2 del braccio 1
$\boldsymbol{d21_{PjL1}}$	Pj2	Link 1 del braccio 1

Alla fine si fa il minimo valore tra questi e si ottiene la distanza tra i bracci che sarà usata nelle regole logiche.

Come si era aspettato con queste regole il braccio 1 si blocca (la distanza rimane costante) quando soggetto alla situazione come nella figura 4.2., come si vede nei grafici sotto.

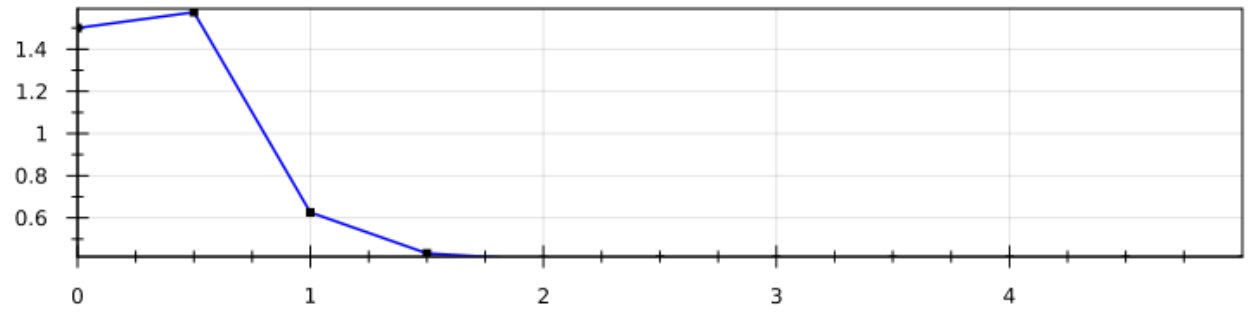


Grafico 4.24. – Distanza dal punto di collisione

È importante notare che esiste una situazione dove il sistema si blocca quando non doveva bloccarsi. Questa situazione è rappresentata nella figura 4.3.

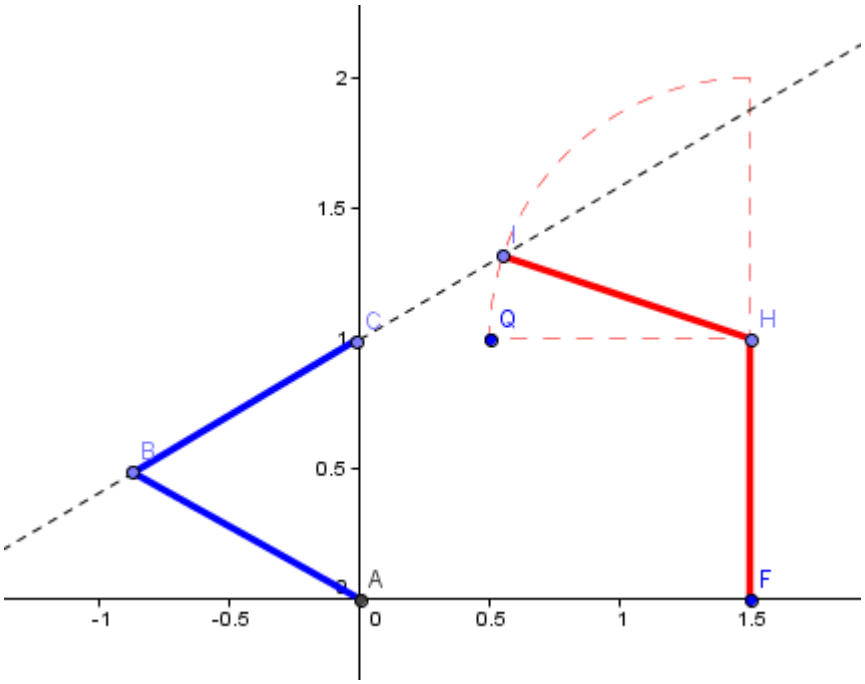


Figura 4.3. – Nuova triettoria dei bracci e punto sbagliato di collisione I

Quindi se imponiamo queste condizione (Target1 = (0,1) e Target2=(0.5,1)) , il sistema si bloccherà all'arrivare al punto I, perché secondo i metodi di calcolo di distanza che abbiamo utilizzato, la distanza tra Pw2 e il Link 2 del braccio 1 sarà nulla, quando davvero non lo è.

Questo è dovuto al fatto che abbiamo utilizzato la equazione di retta per il Link 2, ma il Link 2 è un segmento di retta. Allora sono necessarie modifiche nei calcoli per portare un risultato più ragionevole. Nella figura 4.4 si vede la rappresentazione di questa situazione.

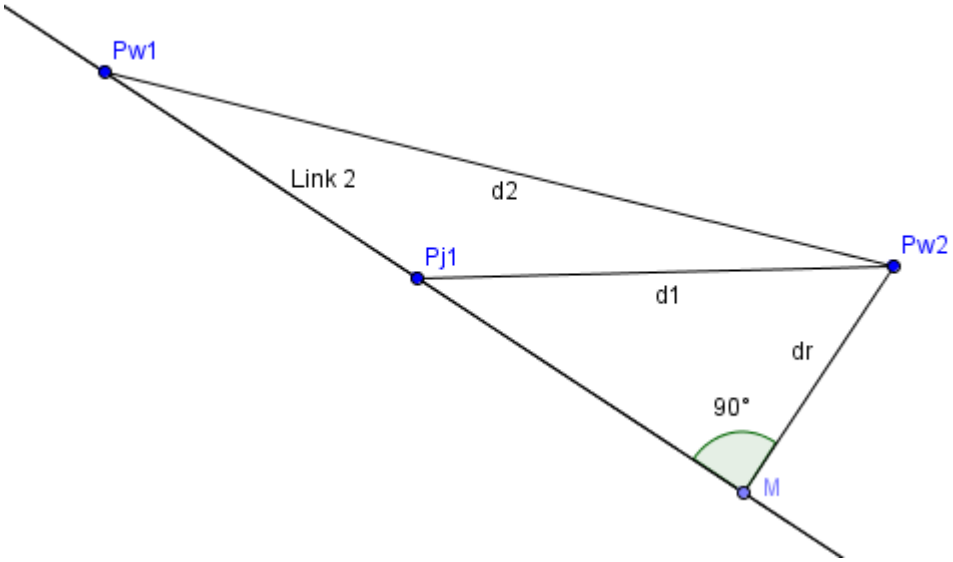


Figura 4.4. – Situazione dove l'uso di dr come distanza è sbagliato

Allora si deve creare un algoritmo che vedi se il punto (Pw2 nel caso) è "sopra" il segmento di retta, cioè, se la proiezione del punto sulla retta (punto M) è nel segmento di retta. E se non lo è, la distanza tra Pw2 e il link deve essere calcolata come distanza semplice tra i punti. Questo è l'algoritmo usato, dove dm2 è la distanza tra M e Pw1, e dm1 è la distanza tra M e Pj1

If $dm_2 > dm_1$

If $dm_2 > L_2$ *then* $d_{21} = d_1$

Else $d_{21} = d_R$

Else

If $dm_1 > L_2$ *then* $d_{21} = d_2$

Else $d_{21} = d_R$

Con l'inserimento di queste modifiche riusciamo ad avere i risultati aspettati col sistema arrivando alle posizione desiderate.

Questi ultimi comenti servono per mostrare che l'uso del software una migliora dinamica nella progettazione delle regole, poiché se troviamo una situazione che non è da noi desiderata, possiamo modificare le regole, senza ancora implementare nei sistemi reali.

5. Conclusioni

L'uso della co-simulazione (attraverso lo MCA) per la modellazione di sistemi meccatronici si è rilevata di non grande dificoltà per un utente non esperto in linguaggi logici, come era il mio caso.

Inoltre si è dimostrata di grande utilità per sviluppare le regole logiche, comportando un metodo "dinamico" per sviluparle, in modo che le regole pensate possono essere facilmente testate, confrontate con l'aspettato e corretti se necessario.

Dall'altra parte, l'apprendimento del linguaggio, a livello base, è stato non molto espendioso, dato che, nel mio caso, mi sono messo per meno di due mesi a fare tutte le manipolazioni necessarie per possibili richieste dal programma.

L'unica barriera per un uso più diffuso è la necessità di utilizzarlo in ambiente Linux, che richiede ad un utente di Windows di usarlo attraverso una macchina virtuale, che porta una grande simulazione rallentamento.

In ultima analisi, l'uso del co-simulatore porta anche il vantaggio di testare le regole prima di provarle in pratica, evitando che succedano errori come collisioni o pure movimenti che non sono desiderati dal progetto.

6. Bibliografia

[1] L. Baresi, G.Ferretti, A. Leva, and M. Rossi, “Flexible Logic-based Co-simulation of Modelica Models”, Dipartimento di Elettronica e Informazione - Politecnico di Milano

[2] L.Sciavicco, B.Siciliano, “Robotica industriale – Modellistica e controllo di robot manipolatori”, (2a ed.) Mc Graw-Hill, 2000

[3] P. A. Fritzson, Principles of object-oriented modeling and simulation with Modelica 2.1. John Wiley and Sons, 2004.

[4] M. Pradella, A. Morzenti, and P. San Pietro, “The symmetry of the past and of the future: bi-infinite time in the verification of temporal properties,” in Proceedings of ESEC/SIGSOFT FSE, 2007, pp. 312–320.

[5] E. Ciapessoni, P. Mirandola, A. Coen-Porisini, D. Mandrioli, and A. Morzenti, “From formal models to formally based methods: An industrial experience,” ACM TOSEM, vol. 8, no. 1, pp. 79–113, 1999.

[6] – C.A. Furia, D. Mandrioli, A. Morzenti, M. Rossi, "Modeling Time in Computing", XVI, 423 p., Cap 9 , 2012

[7] – G. Ferretti, M. Gritti, G. Magnani, and P. Rocco, "A Remote User Interface to Modelica Robot Models", Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, pp. 231-240, 2003

[8] L. Baresi, A. Morzenti, A. Motta, and M. Rossi, “Towards the uml-based formal verification of timed systems,” in Proc. of FMCO, ser. LNCS, vol. 6957, 2010, pp. 267–286.

[9] M. Pradella, “A User's Guide to Zot,” in CNR IEIIT, 2009.