Università degli Studi di Pisa
Informatica e Sistemi in Tempo Reale

Corso di Laurea Magistrale in Ing. Robotica e dell'Automazione

Documento di Progetto

Ping Pong Robot

Aprile 2021

Gruppo di lavoro:
Pietro Gori, p.gori3@studenti.unipi.it
Matricola 564872

Vincenzo Degiacomo, v.degiacomo@studenti.unipi.it
Matricola 564713

Docente:
Prof. Giorgio C. Buttazzo

Anno Accademico 2020-2021

Indice

1 Descrizione Generale

Il progetto si propone la realizzazione del software di un robot cartesiano a due gradi di libertà (direzione longitudinale e di spinta), capace di individuare la posizione attuale e futura di una pallina in un piano, per mezzo di una telecamera, al fine di colpirla e invertirne il verso del moto. È stato realizzato, inoltre, un ambiente di simulazione ispirato al gioco del Ping-Pong, così da testare il comportamento del robot in un contesto dinamico e il quanto più possibile simile alla realtà.

2 Modellazione Fisica

Il sistema si compone di una pallina, due robot, giocatore e avversario, e l'ambiente di gioco: ciascuno modellato fisicamente secondo le equazioni che ne descrivono il comportamento in un sistema cartesiano a tre dimensioni. In alcuni casi sono stati introdotti fattori, moltiplicativi o additivi, con lo scopo di rendere maggiormente comprensibile quello che sta accadendo all'interno del sistema.

2.1 Pallina

La pallina è stata considerata come un punto materiale privo di massa, dotato di velocità iniziale propria, il cui comportamento cinetico è regolato dalle seguenti relazioni:

$$\begin{cases} x = x_0 + v_{Px} * \Delta t * (1 - \beta) \\ y = y_0 \\ z = z_0 + v_{Pz} * \Delta t * (1 - \beta) \end{cases}$$

dove $[x_0, y_0, z_0]$ rappresentano la posizione iniziale dell'oggetto, $[v_{Px}, v_{Py}, v_{Pz}]$ le velocità istantanee della pallina e β un coefficiente di rallentamento del moto. Sono state invece trascurate le accelerazioni poiché avrebbero aumentato notevolmente la complessità del modello senza apportare benefici ai fini della simulazione.

Gli urti contro le racchette dei robot sono stati considerati completamente anelastici e modellati secondo le relazioni:

$$v_{Px} = -v_{Px_0} + v_{robot_x} * damp \tag{1}$$

$$v_{Pz} = -v_{Pz_0} + v_{robot_z} * damp \tag{2}$$

dopo l'urto quindi, la pallina inverte il verso della propria velocità iniziale a cui si somma quella della racchetta, già opposta in verso, soggetta ad un coefficiente di smorzamento del rimbalzo ("damp").

2.2 Robot

I robot si muovono sfruttando dei motori DC controllati attraverso un regolatore PID ed inseguono il riferimento dato dalla posizione della pallina individuata dalla camera. Anche la camera ha due motori, uno per il tilt e l'altro per il pan, che però non sono stati implementati direttamente nel programma.

2.2.1 Motore

Il motore è stato modellato a partire da parametri reali, di un vero motore DC, di cui è stata ricavata la funzione di trasferimento tramite la Trasformata di Laplace e poi quest'ultima è stata discretizzata per poterla riportare in codice C. Chiaramente, anche il controllore è stato progettato già discretizzato così da facilitarne l'implementazione nel programma.

$$G(s) = \frac{\Theta(s)}{V(s)} = \frac{K}{s(\tau s + 1)}$$

Figura 1: Funzione di Trasferimento del motore DC in Laplace

$$G(z) = \frac{\Theta(z)}{V(z)} = \frac{Az^{-1} + Bz^{-2}}{1 - (1+p)z^{-1} + pz^{-2}}$$

Figura 2: Funzione di Trasferimento del motore DC discretizzata

Antitrasformando l'ultima equazione otteniamo un'espressione tempo discreta che restituisce il valore dell'angolo di rotazione dell'albero motore in funzione delle uscite e degli ingressi del regolatore agli istanti precedenti.

$$\Theta(k) = Av(k-1) + Bv(k-2) + (1+p)\Theta(k-1) - p\Theta(k-2)$$

Figura 3: Espressione che rappresenta il funzionamento del motore nel dominio tempo discreto

I parametri che si vedono nelle equazioni (K, A, B, p, τ) non sono altro che delle costanti legate a parametri fisici del motore come il tempo di risposta, la coppia nominale, velocità di rotazione, tensione di alimentazione, corrente nominale, resistenza elettrica del motore ma anche al tempo di campionamento della funzione continua ($T = 5ms$).

$$\begin{cases} p = e^{-T/\tau} = 0.769 \\ A = K(T - \tau + p\tau) = 0.116 \\ B = K(\tau - pT - p\tau) = 0.106 \end{cases}$$

2.2.2 Controllore PID

Per quanto riguarda il controllore è stato implementato, come da richiesta, un regolatore PID che esegue un controllo Proporzionale, Integrativo e Derivativo sulla posizione del robot. In particolare il controllo PI è effettuato direttamente sulla posizione del robot per avere un errore nullo a regime ma anche una risposta piuttosto rapida. La parte derivativa è introdotta prendendo direttamente l'uscita del sistema e applicando il controllo di tipo D. Questo di fatto è equivalente ad effettuare una regolazione proporzionale sulla velocità del robot ma ha come vantaggio quello di stabilizzare il sistema che con un PID ideale sarebbe instabile.

In figura è riportato lo schema su cui si è basato il controllo dei motori.

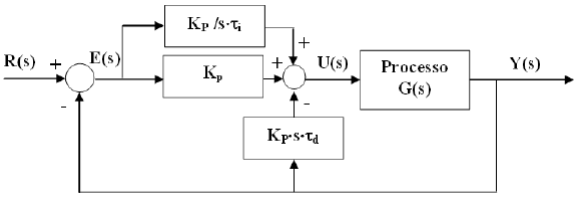

Figura 4: Sistema in reazione con controllo PID sul processo

Tramite l'utilizzo di Matlab/Simulink, con cui è stato anche rappresentato il sistema, sono stati tarati i vari parametri del regolatore.

$$\begin{cases} K_P = 0.5 \\ K_I = 0.01 \\ K_D = 0.008 \end{cases}$$

2.3 Ambiente

Oggetto principale dell'ambiente di simulazione è certamente il tavolo da gioco; si tratta di un rettangolo di dimensioni come specificato in **Figura 6** e appartenente al piano XZ, con la normale Y uscente da esso.

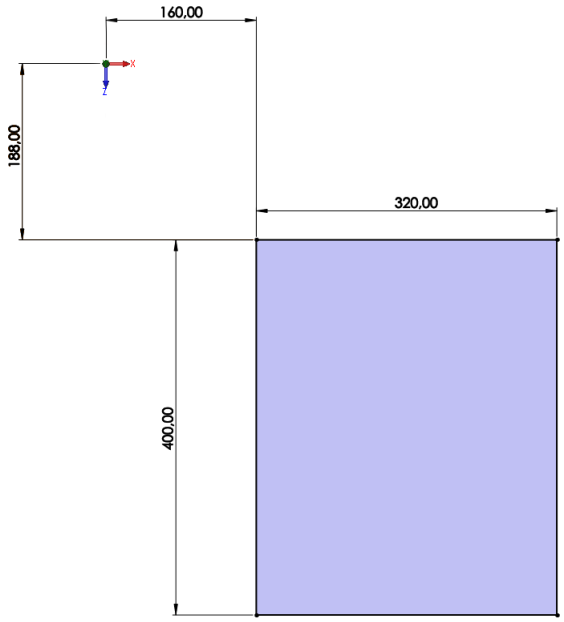

Figura 5: Dimensioni e orientamento rispetto all'origine del tavolo di gioco (in pixel)

3 Scelte di Progetto

3.1 Interfaccia utente

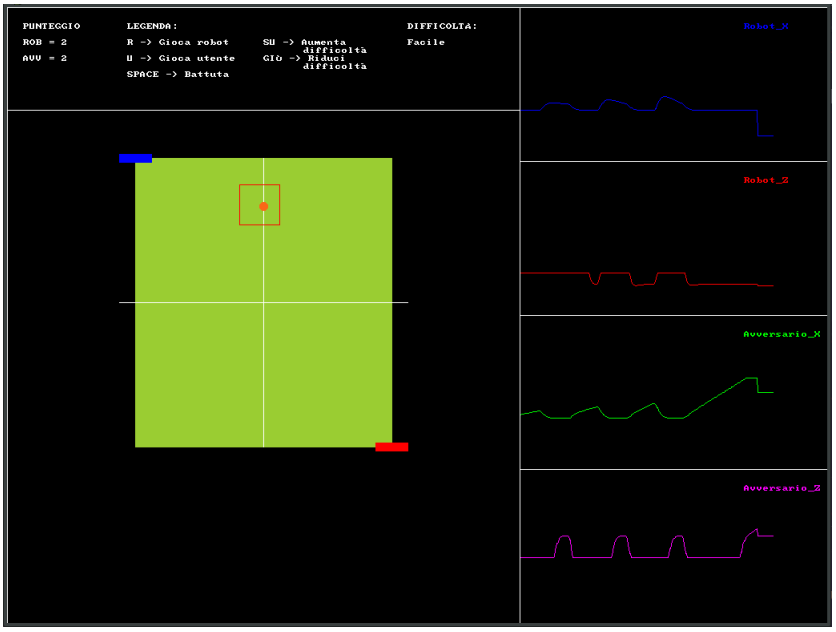

Figura 6: Interfaccia utente

Per lo sviluppo dell'interfaccia utente è stata utilizzata la libreria Allegro (versione 4.4.2), la quale consente la creazione di finestre, la rappresentazione di testo e figure 2D e inoltre gestisce perfettamente le periferiche di input, quali mouse e tastiera.

La finestra, di dimensioni 1024x768 px, si compone di tre parti principali come in **Figura 7**.

La prima, in alto a sinistra, contiene diverse informazioni quali il punteggio di gioco, la difficoltà impostate e la legenda dei tasti utilizzabili e delle loro funzioni. Prendendo in analisi quest'ultima, notiamo che è possibile effettuare la "battuta", cioè inizializzare la velocità della pallina ad un valore prefissato, per mezzo della *Spacebar*; attraverso i tasti *U* e *R* è possibile scegliere, rispettivamente, se a giocare come avversario sia l'utente, utilizzando il mouse, oppure un altro robot. Infine con i tasti direzionali *Su* e *Giù* è possibile variare la difficoltà del gioco, ossia l'incremento di velocità della pallina.

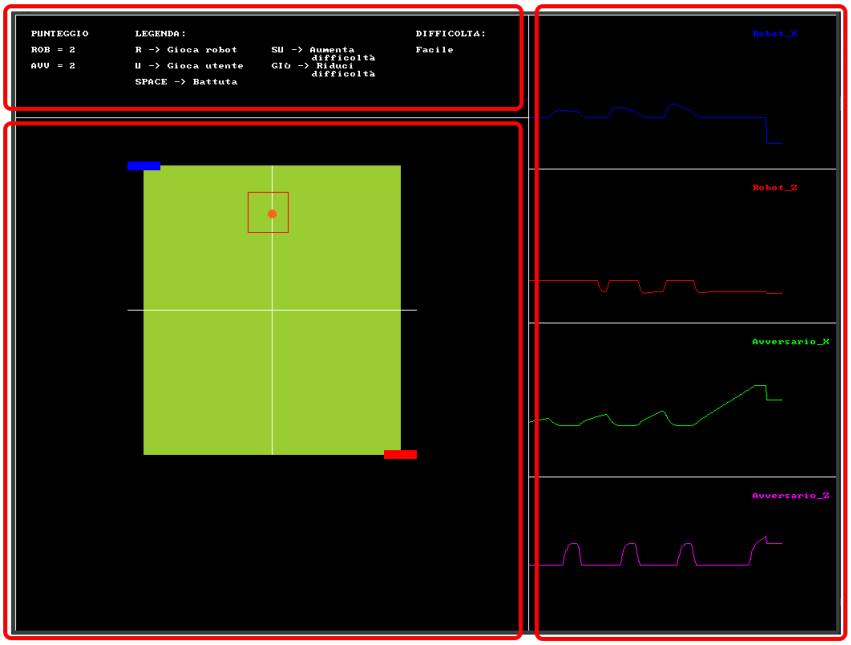

Figura 7: Divisione finestra principale

La seconda, in basso a sinistra, raffigura l'ambiente di simulazione vero e proprio in cui i giocatori possono muoversi; i due rettangoli pieni, blu e rosso, rappresentano rispettivamente la posizione nel piano della racchetta del robot e dell'avversario. Il quadrato rappresenta la sezione di immagine che la camera sta scansionando in quell'istante alla ricerca del cerchio arancione che invece rappresenta la posizione della pallina nel piano.

La terza, nella parte destra della finestra, contiene quattro grafici, uno per ciascun motore presente nel sistema (due per robot). Sulle ascisse e le ordinate sono riportati rispettivamente tempo e posizione del motore in questione, così da evidenziarne la risposta, ossia il transitorio, rispetto ad un riferimento dato.

3.2 Strutture Dati Condivise

Il programma è stato diviso in più file così da permettere a più persone contemporaneamente di lavorare ad elementi diversi del progetto oltre al fatto che in questo modo si riescono a riconoscere molto facilmente le varie parti del programma e questo è utile soprattutto in fase di debugging. Per quanto riguarda le strutture condivise dai vari task esse sono:

- *win*: definisce la finestra di ricerca dell'oggetto.
- *coord*: contiene le coordinate di un punto.
- mt_func: contiene le variabili sulla base delle quali viene calcolato $\Theta(k)$.
- *state*: permette di definire le variabili di stato (posizione e velocità).

- *status*: descrive tutti i parametri della pallina (colore, raggio, coordinate, velocità).

In aggiunta, come verrà spiegato nel paragrafo successivo, le variabili condivise appartengono a varie tipologie di risorse a seconda del task che le sovrascrive. Di seguito è inserito un elenco delle variabili che appartengono a ciascuna risorsa, per una migliore comprensione del programma:

- $Camera\ buffer$: $buffer$, $window$.
- $Motor\_X\ buffer$: $robot\_x$, $home$.
- $Motor\_Z\ buffer$: $robot\_z$.
- $Motor\_X\_Adversary\ buffer$: $adversary\_x$.
- $Motor\_Z\_Adversary\ buffer$: $adversary\_z$.
- $Ball\ buffer$: $ball$, p_rob, p_avv.
- $User\ buffer$: $player$, $start$.

Tutte le variabili sopra citate sono ovviamente protette a semafori per evitare problemi di inconsistenza dei dati. In particolare si è fatto ricorso a semafori di tipo $Mutex$, adottando il protocollo $HighestLockerPriority$ (anche conosciuto come $ImmediatePriorityCeilingprotocol$), eliminando così la presenza delle code e il rischio di deadlock.

3.3 Tasks

Nel programma sono presenti 9 tasks, la cui gestione è affidata alla libreria *pthread.h* e all'algoritmo di scheduling $Round - Robin$. Di questi, otto sono quelli fondamentali per il corretto funzionamento del sistema e uno è per il debugging (stampa a terminale le deadline miss degli altri tasks). Quest'ultimo aspetto è molto importante soprattutto in fase di realizzazione o di eventuale modifica del software pertanto non è stato eliminato una volta ultimata la prima versione.

Dal diagramma Task-Risorse di **Figura 8** si capisce come i task comunicano tra loro. In ciascuna risorsa sono presenti le variabili su cui scrivono (o che vengono lette) dai vari task. Di seguito è riportato un elenco dei vari task con la loro descrizione.

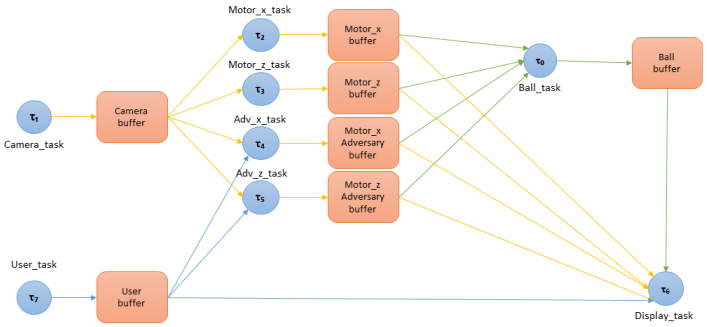

Figura 8: Diagramma Task-Risorse

3.3.1 Pallina

Esso si occupa della gestione della pallina aggiornando le sue coordinate nel piano e gestisce anche i rimbalzi con le racchette, sempre seguendo la fisica modellata in precedenza.
La funzione $frand(ERR\_MIN, ERR\_MAX)$ aggiunge una certa casualità al rimbalzo lungo X della pallina così che le racchette non entrino in un loop infinito in cui si scambiano la pallina senza mai sbagliare. Sempre questo task controlla se la pallina esce dall'area di gioco ed eventualmente assegna un punto a favore dell'ultima racchetta che ha toccato la pallina e poi reinizializza quest'ultima nella posizione iniziale.

3.3.2 Robot Camera

Il task legato alla camera ha come scopo quello della ricerca della pallina, in base al suo colore, all'interno dell'area di gioco. La funzione *centroid* si occupa della ricerca della pallina in una finestra specificata da una struttura di tipo *win*. La ricerca consiste in una scansione dei pixel per trovare quelli del colore esatto della pallina. Dopo la scansione viene calcolato il centro di massa della sfera. Un'altra funzione, *prediction*, sfruttando i risultati della funzione precedente esegue una predizione lineare sulla posizione della pallina. In questo modo la finestra di ricerca è in grado di inseguire la pallina senza essere sempre in ritardo.

$$\begin{cases} x(k+1) = 2x(k) - x(k-1) \\ z(k+1) = 2z(k) - z(k-1) \end{cases}$$

Figura 9: Equazioni di predizione della posizione della pallina

I dati relativi alla posizione attuale, precedente e predetta sono raccolti nel vettore di tipo *coord* chiamato $buffer$ a cui accedono altri task. Un'altra caratteristica del task è che restringe o allarga la finestra di ricerca se trova o meno la pallina.

3.3.3 Motor_X

Implementa la fisica del motore DC e del suo controllo spiegati in precedenza per quanto riguarda l'asse X del robot autonomo.

3.3.4 Motor_Z

Implementa la fisica del motore DC e del suo controllo spiegati in precedenza per l'asse Z del robot autonomo.
Chiaramente sia in $Motor\_X$ che in $Motor\_Z$ esiste una apposita funzione *update_state* che converte il movimento rotativo dell'albero motore in un movimento lineare (ad esempio tramite una cinghia) e successivamente definisce i nuovi valori di posizione e velocità del robot; contemporaneamente viene fatto un controllo sulla posizione per evitare che la racchetta invada il campo avversario.

3.3.5 Adversary_X

Implementa la fisica del motore DC e del suo controllo spiegati in precedenza per quanto riguarda l'asse X del robot avversario.

3.3.6 Adversary_Z

Implementa la fisica del motore DC e del suo controllo spiegati in precedenza per l'asse Z del robot avversario.
Così come per il robot autonomo, anche per quello avversario esiste la funzione *update_state* che esegue la conversione del moto, aggiorna velocità e posizione e allo stesso tempo evita che la racchetta invada il campo avversario. Una aggiunta è la possibilità di far giocare l'utente tramite il mouse. Questa è una vera e propria modalità attivabile tramite tastiera in cui è possibile cambiare anche la difficoltà di gioco.

3.3.7 Display

Questo è il task addetto alla rappresentazione grafica.Tramite l'ausilio di apposite funzioni disegna due macro-aree: l'area di gioco e quella dei grafici come visibile in **Figura 6**.
Nell'area di gioco è compresa una legenda che aiuta l'utente nel capire come interfacciarsi con il programma oltre al tabellone del punteggio, come già spiegato nel paragrafo 3.1.

3.3.8 Command

Il task *Command* è quello che interpreta la tastiera e, insieme al mouse, costituisce l'unico modo per interagire con il software in esecuzione. Come già anticipato nel paragrafo 3.1 premendo particolari tasti si ottengono azioni diverse. Nello specifico questo task si occupa di individuare quale tasto è stato premuto e di conseguenza esegue la parte di codice legata a tale evento.

3.3.9 Deadline Stamp

In base a quanto accennato in precedenza è stato inserito un ulteriore task che non conferisce al software particolari caratteristiche poiché è stato e potrà essere utilizzato (cambiando leggermente il software) per la fase di debugging; infatti si occupa di stampare su terminale le deadline miss degli altri task così da dare ulteriori informazioni legate a possibili malfunzionamenti.

4 Analisi e Comportamento del sistema

Il programma è stato eseguito più volte e per finestre temporali differenti, e in ciascuna di esse la deadline ottenute sono state nulle; alla luce di ciò si è ritenuta vincente la scelta dei periodi dei task e delle priorità assegnate, anche se sarebbero necessari tempi maggiori e condizioni di esecuzione eterogenee per poter confermare tale affermazione.

Eseguendo il programma si nota subito la reattività del sistema: la pallina si muove senza problemi all'interno dello spazio di gioco e viene immediatamente riconosciuta dalla camera, che restringe su di essa la finestra di scansione. Anche in caso di fuoriuscita da quest'ultima, la pallina viene ritrovata immediatamente allargando l'area di scansione a tutto il campo da gioco. Premendo *space* viene eseguita la battuta e le racchette, ossia i robot, iniziano immediatamente a inseguire la palla, le cui coordinate sono ottenute dalla camera. Lo spazio di libertà di movimento dei robot è limitato a circa metà campo, essi inoltre dopo aver colpito la palla tornano in una posizione centrale così da preparasi alla prossima ricezione. Il gioco continua finché la pallina non fuoriesce dal campo: a questo punto viene assegnato un punto al giocatore che ha effettuato l'ultimo tocco utile e il sistema resta in attesa di una nuova battuta.

5 Conclusioni

Dopo una prima implementazione di visuale prospettica, si é preferito inserire dei grafici che chiariscano maggiormente ciò che accade all'interno del sistema, in particolare le risposte dei motori dei robot, soggetti principali del progetto stesso. Nello specifico, la tecnica di realizzazione utilizzata per la prospettiva non aggiungeva nessuna informazione utile alla comprensione della dinamica del sistema poiché realizzata attraverso la rotazione e la proiezione su un piano ausiliario del campo di gioco; ciò restituiva un'immagine priva di profondità e senza punto di fuga, fondamentale per una comprensione corretta della spazialità. Per il futuro sarebbe interessante la realizzazione di una visualizzazione tridimensionale, ad esempio con Unreal Engine, poiché tutti gli oggetti del programma hanno buffer di coordinate tridimensionali, correttamente gestite e aggiornate. Sarebbe inoltre interessante l'implementazione ad hardware (per cui il progetto è stato di fatto pensato) e il confronto con i risultati ottenuti in simulazione.