



SAPIENZA
UNIVERSITÀ DI ROMA

Progettazione e sviluppo di un controllore PID per ottimizzare l'approccio alla palla nel tiro del robot calciatore

Facoltà di Ingegneria dell'informazione, informatica e statistica
Corso di Laurea in Ingegneria Informatica ed Automatica

Candidato
Samuele Civale
Matricola 1938135

Relatore
Prof. Thomas Alessandro Ciarfuglia

Anno Accademico 2023/2024

Tesi non ancora discussa

Progettazione e sviluppo di un controllore PID per ottimizzare l'approccio alla palla nel tiro del robot calciatore

Tesi di Laurea. Sapienza – Università di Roma

© 2024 Samuele Civale. Tutti i diritti riservati

Questa tesi è stata composta con L^AT_EX e la classe Sapthesis.

Email dell'autore: samuele.civale@gmail.com

Sommario

La progettazione di un controllore rappresenta un elemento cruciale nel processo di ottimizzazione di un sistema e della sua interazione con l'ambiente circostante. L'obiettivo primario di questa ricerca è affrontare l'instabilità riscontrata nel comportamento del robot calciatore durante l'approccio al pallone, mediante l'implementazione di un controllore Proporzionale-Integrale-Derivativo (da ora in avanti PID). La fase di progettazione è stata condotta sfruttando il software SimRobot, che consente la simulazione dettagliata delle dinamiche del robot in campo. Questa ricerca non si limita a risolvere il problema specifico dell'instabilità, ma ambisce anche a contribuire alla comprensione più ampia delle dinamiche di controllo nei sistemi robotici operanti in ambienti dinamici e competitivi, come quello caratteristico della RoboCup. La tesi è organizzata in cinque capitoli, ognuno dedicato a un aspetto specifico della ricerca. Nel primo capitolo, si delinea il problema in analisi e si fornisce un quadro contestuale per comprenderne l'importanza. Il secondo capitolo si concentra sugli strumenti utilizzati, offrendo una descrizione approfondita del simulatore e del robot coinvolti nel lavoro. Nel terzo capitolo, la Metodologia di ricerca viene esaminata in dettaglio, delineando le procedure adottate per la progettazione del controllore e la loro applicazione pratica e un confronto tra la vecchia e la nuova implementazione. Nel quarto capitolo, vengono presentati e analizzati i risultati ottenuti durante la ricerca. Infine, nel quinto capitolo, vengono tratte le conclusioni, evidenziando le implicazioni dei risultati e proponendo possibili sviluppi futuri.

Indice

1	Introduzione	1
1.1	Contestualizzazione della RoboCup	1
1.2	Motivazione, Rilevanza del Problema, Scopo e Obiettivi della Tesi	2
2	Strumenti utilizzati	4
2.1	Descrizione del Simulatore	4
2.2	Caratteristiche del Robot	6
3	Metodologia	7
3.1	Architettura del controllore PID	7
3.2	Contesto di chiamata della funzione	9
3.3	Implementazione precedente del Controllore PID	10
3.4	Nuova implementazione del Controllore PID	14
3.5	Confronto tra le due implementazioni	18
4	Risultati e analisi	20
4.1	Procedura Sperimentale	20
4.2	Metriche di Valutazione	21
4.3	Risultati numerici	22
4.4	Analisi dei risultati	25
5	Conclusioni	30
	Bibliografia	32

Capitolo 1

Introduzione

1.1 Contestualizzazione della RoboCup

La RoboCup, conosciuta anche come Robot World Cup, è stata ideata nel 1997 con l'audace obiettivo di sviluppare entro il 2050 una squadra di robot umanoidi in grado di competere con la squadra di calcio campione del mondo[4].

Negli anni '90, l'idea di far giocare i robot a calcio è emersa inizialmente grazie al Professor Alan Mackworth dell'Università della Columbia Britannica [5]. Nel 1992, un gruppo di ricercatori giapponesi ha discusso delle grandi sfide nell'Intelligenza Artificiale e ha proposto l'uso del calcio come vettore di diffusione dello stato attuale della scienza e promozione dell'innovazione tecnologica. Dopo studi di fattibilità, nel giugno 1993, un gruppo di ricercatori, tra cui Minoru Asada, Yasuo Kuniyoshi e Hiroaki Kitano, ha lanciato la competizione di robot chiamata inizialmente Robot J-League. Il termine "J-League" fa riferimento alla massima serie di calcio giapponese. Successivamente, la competizione è stata ribattezzata "RoboCup" in risposta alle reazioni positive della comunità internazionale di ricercatori. Indipendentemente nello stesso periodo, il laboratorio del Professor Minoru Asada presso l'Università di Osaka e la Prof.ssa Manuela Veloso insieme al suo studente Peter Stone presso la Carnegie Mellon University stavano lavorando su robot capaci di giocare a calcio. Senza la partecipazione di questi precursori del settore, RoboCup non avrebbe visto mai la luce. Nel settembre 1993, sono state stabilite regolamentazioni specifiche, affrontando poi problemi organizzativi e tecnici in varie conferenze e workshop.

Contemporaneamente il team di Noda di ETL ha annunciato il Soccer Server versione 0, un simulatore di sistema aperto per il calcio multi-agente, seguito dalla versione 1.0 distribuita via web. La prima dimostrazione pubblica è avvenuta all'IJCAI-95. Durante questa conferenza, si è annunciata l'organizzazione della prima Coppa del Mondo di Calcio per Robot in concomitanza con l'IJCAI-97.

Si decise anche di organizzare la Pre-RoboCup-96 in concomitanza con la Conferenza Internazionale sulla Robotica e i Sistemi di Intelligenza (IROS-96) a Osaka. Questa competizione comprendeva otto squadre impegnate in un campionato di simulazione e dimostrazioni di robot reali. Nonostante le dimensioni limitate, questo evento ha contribuito significativamente a promuovere la ricerca e l'istruzione nel settore.

La prima gara e conferenza ufficiale di RoboCup si tenne nel 1997, con la partecipazione di oltre 40 squadre, sia reali che virtuali. Da allora, RoboCup ha continuato a crescere, diventando un evento internazionale di rilievo che coinvolge squadre di robot nel contesto del calcio, incentivando lo sviluppo e la ricerca in questo ambito.

La RoboCup nel corso degli anni si è evoluta, non rimanendo ancorata alle sole simulazioni calcistiche, ma abbracciando anche nuovi tipi di sfide. Attualmente, la competizione si articola in cinque leghe principali, ognuna con un focus distintivo:

- **RoboCup Soccer:** Questa lega si concentra sulla competizione calcistica tra squadre di robot e presenta diverse categorie, tra cui Small Size, Middle Size, Humanoid e 2D Simulation
- **RoboCup Rescue:** Affrontando simulazioni di operazioni di soccorso in scenari di disastro, questa lega mira a sviluppare robot in grado di operare in ambienti pericolosi e assistere nelle operazioni di salvataggio.
- **RoboCup@Home:** Rivolta alla robotica domestica e all'assistenza, questa lega sfida i partecipanti a sviluppare robot capaci di svolgere compiti utili in ambienti domestici o di assistere persone con esigenze specifiche.
- **RoboCup Industrial:** Con un focus sull'applicazione di tecnologie robotiche avanzate in contesti industriali, questa lega mira a migliorare l'automazione e l'efficienza in ambienti di produzione.
- **RoboCupJunior:** Rivolta agli studenti più giovani, questa lega offre una piattaforma per coinvolgere i futuri talenti nella robotica attraverso discipline come il calcio robotico, la danza e altre sfide creative.

I team SPQR, rappresentanti del Dipartimento di Ingegneria Informatica, del Controllo e Gestionale Antonio Ruberti della Sapienza Università di Roma, partecipano attivamente alle competizioni RoboCup dal 1998. Il mio coinvolgimento si è concentrato sulla distribuzione del software utilizzato nei robot NAO, che competono nella Standard Platform League[2].

1.2 Motivazione, Rilevanza del Problema, Scopo e Obiettivi della Tesi

La progettazione di un controllore PID mirato alla gestione dell'approccio alla palla nella RoboCup si origina dalla necessità di affrontare una delle sfide cruciali nella robotica calcistica. Il controllo preciso di come il robot si avvicina al pallone riveste un ruolo fondamentale per garantire il successo delle squadre di robot calciatori, impattando direttamente sulla precisione e la velocità delle azioni di gioco. La mancanza di un controllo ottimale può compromettere le performance globali del robot durante le competizioni.

La tesi mira a contribuire in modo significativo alla comprensione e all'ottimizzazione del controllo dell'approccio alla palla, affrontando una sfida chiave nel campo della robotica applicata al calcio. L'ambiente della RoboCup fornisce un eccellente



Figura 1.1. RoboCup 2023

terreno di prova per lo sviluppo e il test di soluzioni innovative, che possono essere applicate al di fuori del contesto calcistico. La specifica applicazione riguardante la regolazione della velocità di avvicinamento alla palla può essere generalizzata per qualsiasi obiettivo coinvolgente un robot in movimento nello spazio tridimensionale. Le soluzioni ottimali sviluppate per la RoboCup possono quindi essere adattate con successo a veicoli con guida autonoma, nanorobot impiegati in ambito biomedico e altre applicazioni.

L’obiettivo centrale della tesi è la progettazione, implementazione e valutazione di un controllore Proporzionale-Integrale-Derivativo (PID) specificamente concepito per gestire l’approccio alla palla da parte di un robot calciatore. La sfida consiste nell’ottimizzare la regolazione della velocità di avvicinamento del robot alla palla, migliorando così le sue performance durante le partite.

Al fine di completare l’obiettivo centrale, si rende necessario affrontare alcuni obiettivi specifici della tesi. Questi includono un’analisi approfondita del contesto RoboCup e dei requisiti associati all’approccio alla palla. Inoltre, è previsto uno studio dettagliato dei controllori PID, seguito dalla progettazione di un controllore appositamente adattato alle esigenze specifiche della competizione. Successivamente, si procederà con l’implementazione e l’integrazione di tale controllore nel sistema robotico. Una fase successiva comprenderà la sperimentazione e la valutazione delle performance attraverso test mirati, con l’obiettivo di analizzare i risultati ottenuti e apportare eventuali ottimizzazioni al controllore.

Capitolo 2

Strumenti utilizzati

2.1 Descrizione del Simulatore

Il software utilizzato per l'analisi dei comportamenti del robot e per la progettazione del controllore è stato SimRobot, incluso nel pacchetto B-Human. Questo pacchetto fa uso del simulatore fisico di robot SimRobot12 come interfaccia principale per lo sviluppo del software. Il simulatore non è soltanto impiegato per lavorare con robot simulati, ma funge anche da interfaccia utente grafica per riprodurre file di log e collegarsi a robot fisici tramite Ethernet o Wi-Fi.

All'interno di SimRobot, è possibile lavorare su alcuni scenari già predisposti. Personalmente, ho concentrato il mio lavoro su `bh.ros2`, apportandovi modifiche per meglio focalizzarmi sugli aspetti di mio interesse (le modifiche specifiche verranno discusse nel capitolo). Ho sfruttato principalmente la Scene view, le Plot views e le Data views di SimRobot: la prima consente di osservare la simulazione attraverso una rappresentazione dinamica del campo di gioco, dei robot e della palla.

Le Plot views permettono la visualizzazione dei dati inviati dal programma di controllo del robot tramite l'utilizzo della macro PLOT. Le Data views, invece, forniscono una visualizzazione strutturata dei dati, agevolando gli sviluppatori nel monitorare e modificare facilmente variabili o campi complessi nel tempo. Questo approccio semplifica il processo di analisi dei dati e di apportare modifiche, migliorando l'efficienza nello sviluppo e nel debug del software del robot. [1].

Analizziamo ora i principali vantaggi e svantaggi dell' utilizzo del simulatore:

Vantaggi:

1. **Costi ridotti:** L'utilizzo dei simulatori può ridurre significativamente i costi associati allo sviluppo e al testing di nuovi algoritmi e strategie. Eliminando la necessità di hardware fisico, i team possono concentrarsi sulla parte software senza preoccuparsi di acquistare componenti costose.
2. **Ambiente controllato:** I simulatori offrono un ambiente virtuale controllato, consentendo ai partecipanti di riprodurre scenari specifici, regolare variabili e testare le prestazioni dei loro robot in condizioni controllate e ripetibili.
3. **Velocità di sviluppo:** La simulazione consente agli sviluppatori di iterare rapidamente e testare diverse strategie senza dover attendere la costruzione

fisica del robot o la partecipazione a eventi dal vivo. Ciò accelera il processo di sviluppo.

4. **Accesso a risorse condivise:** La condivisione di ambienti di simulazione standard può facilitare la collaborazione tra team e consentire loro di confrontarsi su un terreno neutro.

Limitazioni:

1. **Mancanza di realismo:** Anche se il simulatore cerca di replicare il più possibile l'ambiente del mondo reale, possono mancare alcuni aspetti seppur minimi di realismo. Questo può portare a comportamenti e prestazioni dei robot che differiscono da quelli che si verificherebbero in un ambiente fisico.
2. **Complessità del modello:** Creare un modello accurato e complesso dell'ambiente e dei robot può richiedere molto tempo e risorse. Inoltre, la complessità del modello potrebbe aumentare la richiesta di potenza di calcolo.
3. **Overfitting:** Gli algoritmi e le strategie sviluppate in un simulatore potrebbero essere troppo adattate alle condizioni specifiche della simulazione e non generalizzarsi bene nel mondo reale.

La scene view del simulatore è composta da una shell dalla quale è possibile impartire i comandi di inizio e fine simulazione, una finestra nella quale è possibile visualizzare l'andamento dei parametri di interesse attraverso una funzione che mostra la variazione di questi ultimi e infine una sezione in cui è possibile vedere altri dati e info generali del robot, questi ultimi però non sono di nostro interesse.

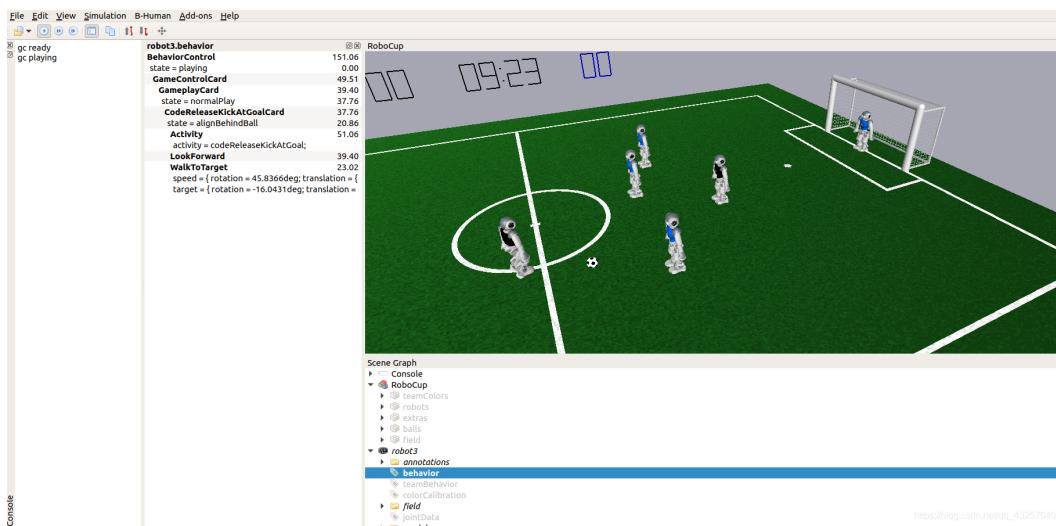


Figura 2.1. SimRobot. Scene View

2.2 Caratteristiche del Robot

Il robot umanoide NAO si distingue per la sua mobilità e innovativa progettazione dei sensori. Con 25 gradi di libertà, offre notevole flessibilità grazie ai motori coreless brush DC Maxon. La sua cinematica del bacino brevettata e l'esclusivo sistema di attuazione contribuiscono alla sua agilità.

Nel contesto della RoboCup, NAO rivela la sua potenza sensoriale. La fotocamera CMOS da 30 FPS, i giroscopi, gli accelerometri e i sensori di forza sotto i piedi forniscono dati in tempo reale, essenziali per le sfide della competizione. Gli encoder rotativi magnetici e i bumper anteriore sui piedi offrono rilevamento di collisioni, mentre i sensori capacitivi sulla fronte consentono l'input tattile.

Il robot è dotato di software integrato, come Choregraphe, che supporta funzioni come la sintesi vocale e il riconoscimento di forme colorate. La comunicazione avviene tramite Ethernet o wireless 802.11g. NAO è programmabile in diversi linguaggi, tra cui C, C++, Python e Urbi, offrendo versatilità per gli sviluppatori.

Grazie alla produzione su larga scala e alla riduzione dei costi, NAO è ora accessibile per le istituzioni accademiche, risolvendo le sfide finanziarie e aprendo nuove possibilità di ricerca nell'ambito della RoboCup. [7]

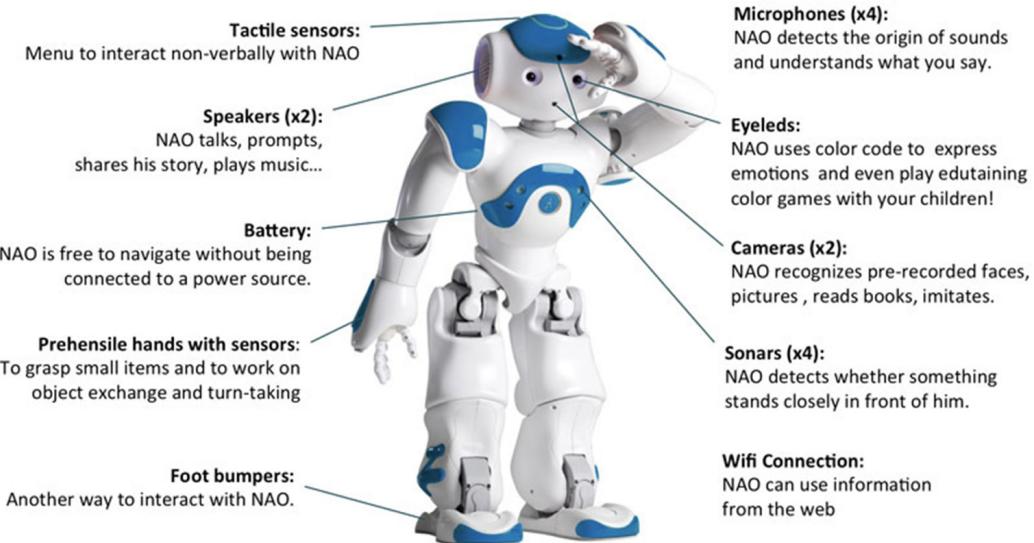


Figura 2.2. Nao Robot.

Capitolo 3

Metodologia

3.1 Architettura del controllore PID

Il controllore PID è un dispositivo di controllo ampiamente utilizzato in ingegneria e automazione per regolare sistemi dinamici. La sua denominazione deriva dalle tre componenti fondamentali che lo compongono: Proporzionale (P), Integrativa (I), e Derivativa (D). Questo tipo di controllore è progettato per mantenere la variabile controllata (output) il più possibile vicina al valore di riferimento desiderato.

La somma pesata di questi tre termini costituisce l'uscita complessiva del controllore PID. In termini algebrici:

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau + k_d \frac{de(t)}{dt}$$

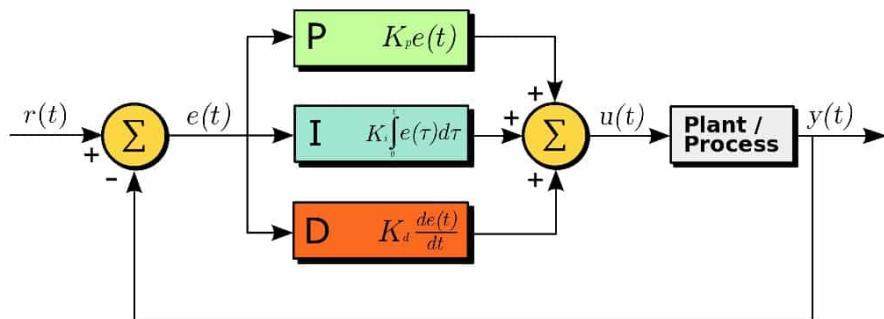


Figura 3.1. Schema di un controllore PID

Osserviamo in dettaglio le tre componenti:

- **Proporzionale (P):**

la componente proporzionale risponde proporzionalmente all'errore attuale, che è la differenza tra il valore desiderato e il valore effettivo della variabile controllata. Questo significa che maggiore è l'errore, maggiore sarà l'uscita del termine proporzionale. La sua funzione principale è quella di ridurre l'errore attuale e portare la variabile controllata più vicina al valore di riferimento.

- **Integrativa (I):**

La componente integrativa è proporzionata all'integrale dell'errore nel tempo. Misura quindi la somma cumulativa degli errori passati. Il termine integrativo contribuisce a eliminare gli errori a regime, riducendo eventuali discrepanze persistenti tra la variabile controllata e il suo valore desiderato. Aiuta a compensare l'errore accumulato nel tempo e a stabilizzare il sistema.

- **Derivativa (D):** la componente derivativa risponde al tasso di variazione dell'errore nel tempo. Il termine derivativo anticipa il comportamento futuro del sistema, riducendo il rischio di oscillazioni e migliorando la stabilità. Contrasta i cambiamenti rapidi nell'errore, contribuendo a prevenire il sovraffaticamento del sistema.

Vantaggi dei regolatori PID:

1. **Semplicità e Facilità di Implementazione:** I regolatori PID sono semplici da implementare, richiedendo solo la misura dell'errore e la messa a punto dei coefficienti, che può essere fatta manualmente o automaticamente.
2. **Versatilità:** I regolatori PID possono gestire una vasta gamma di dinamiche di sistema e disturbi, adattandosi a cambiamenti nei parametri del sistema o nell'ambiente.
3. **Risposte Veloci e Stabili:** I regolatori PID possono fornire risposte rapide e stabili riducendo l'errore, eliminando l'errore a regime e minimizzando sovraelongazioni ed oscillazioni.

Svantaggi dei regolatori PID:

1. **Sensibilità a Rumori ed Errori di Misurazione:** I regolatori PID possono essere sensibili a rumori ed errori di misurazione, amplificando eventuali fluttuazioni nel segnale in ingresso e causando instabilità o oscillazioni.
2. **Sfide con Sistemi Non Lineari:** I regolatori PID possono incontrare difficoltà con sistemi non lineari, manifestando degrado delle prestazioni o instabilità quando le dinamiche del sistema cambiano in modo significativo o imprevedibile.
3. **Integral Windup:** I regolatori PID possono soffrire di integral windup, dove il termine integrale accumula un grande errore nel tempo, causando una risposta considerevole e ritardata. Questo si verifica tipicamente durante la saturazione del sistema o in presenza di disturbi rilevanti.

Inizieremo la nostra analisi considerando il contesto in cui la funzione che regola la velocità di avvicinamento al pallone viene chiamata. Successivamente, esamineremo l'implementazione precedente e la nuova proposta.

3.2 Contesto di chiamata della funzione

La funzione su cui andiamo a lavorare è "getApproachSpeed", che viene chiamata all'interno della libreria "LibStriker.h". Il contesto in cui viene chiamato è quello dell'implementazione di un comportamento base per un attaccante chiamata "card".

Card

La card è basata su un framework di comportamento e dichiara i requisiti e le chiamate di diverse funzionalità utilizzate all'interno del comportamento, come la posizione del pallone, la posizione degli avversari, le dimensioni del campo, la posizione del robot, ecc.

Parametri della Card

Vengono definiti alcuni parametri configurabili della card come la velocità di camminata (`walkSpeed`), il tempo iniziale di attesa (`initialWaitTime`), la soglia di intervallo per il calcio (`kickIntervalThreshold`), alcuni parametri per la regolazione PID (`kp`, `kd`, `minSpeed`, `ki`), ecc.

Implementazione della Card

La classe `BasicStrikerCard` eredita dalla classe `BasicStrikerCardBase` e implementa i metodi `preconditions` e `postconditions`. I preconditions e postconditions sono condizioni che devono essere verificate prima e dopo l'esecuzione di ogni stato nella macchina a stati.

Macchina a Stati

La card implementa una macchina a stati che gestisce il comportamento dello striker. Gli stati includono "start", "goToBallAndDribble", "goToBallAndKick" e "searchForBall". Ogni stato ha azioni associate e transizioni basate su condizioni.

- **Start:**

- *Condizioni di transizione:* Dopo un periodo iniziale di attesa (definito da `initialWaitTime`), la transizione avviene verso lo stato di "goToBallAndDribble".
- *Azioni:*
 1. Attiva la skill per guardare in avanti (`theLookForwardSkill()`).
 2. Attiva la skill per rimanere in piedi (`theStandSkill()`).

- **GoToBallAndDribble:**

- *Condizioni di transizione:* Si sposta a "searchForBall" se il pallone non è visto entro un certo timeout, altrimenti controlla se è possibile eseguire un calcio e transita a "goToBallAndKick" in tal caso.
- *Azioni:*

1. Attiva la skill per muoversi verso il pallone con dribbling (`theGoToBallAndDribbleSkill()`).

- **GoToBallAndKick:**

- *Condizioni di transizione:* Si sposta a "searchForBall" se il pallone non è visto entro un certo timeout, altrimenti ritorna a "goToBallAndDribble" se non è possibile eseguire un calcio.
 - *Azioni:*
 1. Determina se eseguire il calcio immediatamente o meno (`doItAsap`).
 2. Calcola informazioni per il calcio utilizzando la libreria dello striker (`theLibStriker`).
 3. Attiva la skill per muoversi verso il pallone e cercare il tiro (`theGoToBallAndKickSkill()`).

- **SearchForBall:**

- *Condizioni di transizione:* Si sposta a "goToBallAndDribble" se il pallone è visto.
 - *Azioni:*
 1. Attiva la skill per guardare in avanti (`theLookForwardSkill()`).
 2. Attiva la skill per camminare a una velocità relativa (`theWalkAtRelativeSpeedSkill()`).

Funzione `theGoToBallAndKickSkill()`

Questa funzione viene chiamata nello stato "goToBallAndKick" e viene utilizzata per dirigere il robot verso il pallone con l'intenzione di effettuare un calcio. Alcuni parametri passati a questa funzione includono l'angolo di direzione verso il pallone (`angle`), il tipo di calcio (`typeKick`), se deve essere eseguito il prima possibile (`!doItAsap`), la lunghezza massima del calcio (`std::numeric_limits<float>::max()`), alcune opzioni aggiuntive come allineamento preciso e giri durante il calcio, e infine la velocità di approccio (`theLibStriker.getApproachSpeed(range, kp, kd, minSpeed)`) e la precisione di direzione (`Rangea(0_deg, 0_deg)`).

In sostanza, la funzione `theGoToBallAndKickSkill()` gestisce il movimento del robot verso il pallone e la pianificazione del calcio, utilizzando una serie di parametri per controllare il comportamento del robot durante l'approccio. La velocità di approccio è determinata dalla chiamata a `theLibStriker.getApproachSpeed()`, che utilizza un controllore PID con i parametri forniti (`range, kp, kd, minSpeed`).

Andremo quindi a concentrarci sulla funzione "getApproachSpeed" e di riflesso ci concentriamo sullo stato "goToBallAndKick".

3.3 Implementazione precedente del Controllore PID

```

1 Pose2f LibStrikerProvider::getApproachSpeed(Rangef range, float kp,
2   float kd, float minSpeed) const{
3   if(theFieldBall.recentBallPositionRelative().norm() > range.max)
4     return Pose2f(1,1,1);
5
6   if(kd>kp)
7     OUTPUT_WARNING("Derivative gain is greater than proportional gain
8     , this may lead to an extremely slow convergence");
9
10  if(kd == 0)
11    OUTPUT_WARNING("Derivative gain is set to 0, this may lead to
12      lack of convergence due to rippling");
13
14  Vector2f target = goalTarget(true, true);
15
16  float e_angle = abs((theRobotPose.inversePose * target).angle()),
17    e_p = mapToRange(theFieldBall.recentBallPositionRelative().
18    norm()+e_angle , range.min, range.max, minSpeed, 1.f),
19    e_d = theMotionInfo.speed.translation.norm()/1e3,
20    out = clip(kp*e_p - kd*e_d, minSpeed, 1.f);
21
22  return Pose2f(out,out,out);
23 }
```

La funzione `getApproachSpeed` prende in input `range` che è il range di distanze entro la quale chiamare la funzione, `kp` ossia il guadagno porporzionale, `kd` ossia il guadagno derivativo e `minSpeed` ossia la minima velocità ammessa; viene restituito in output una `Pose2f` dove `Pose2f` è una struttura dati che rappresenta una posizione nello spazio bidimensionale con coordinate (x, y) e un angolo di orientamento θ .

Se la norma della posizione relativa della palla rispetto al campo è maggiore del valore massimo specificato da `range.max`, il metodo restituisce una `Pose2f` con valori tutti impostati a 1.

Se il guadagno derivativo è maggiore di quello proporzionale o se il guadagno derivativo dovesse essere nullo stampa in output un avvertimento di una possibile difficoltà di arrivare alla convergenza.

Viene calcolato un vettore di destinazione (`target`) utilizzando una funzione chiamata `goalTarget`.

Vengono calcolati l'errore di angolo (`e_angle`) tra la posizione attuale del robot e il target, l'errore proporzionale (`e_p`) sulla base della norma della posizione relativa della palla e dell'errore di angolo, l'errore derivativo (`e_d`) basato sulla velocità attuale del robot, l'output finale (`out`) usando il controllo proporzionale e derivativo e viene clipato tra i valori di velocità minimi e massimi consentiti.

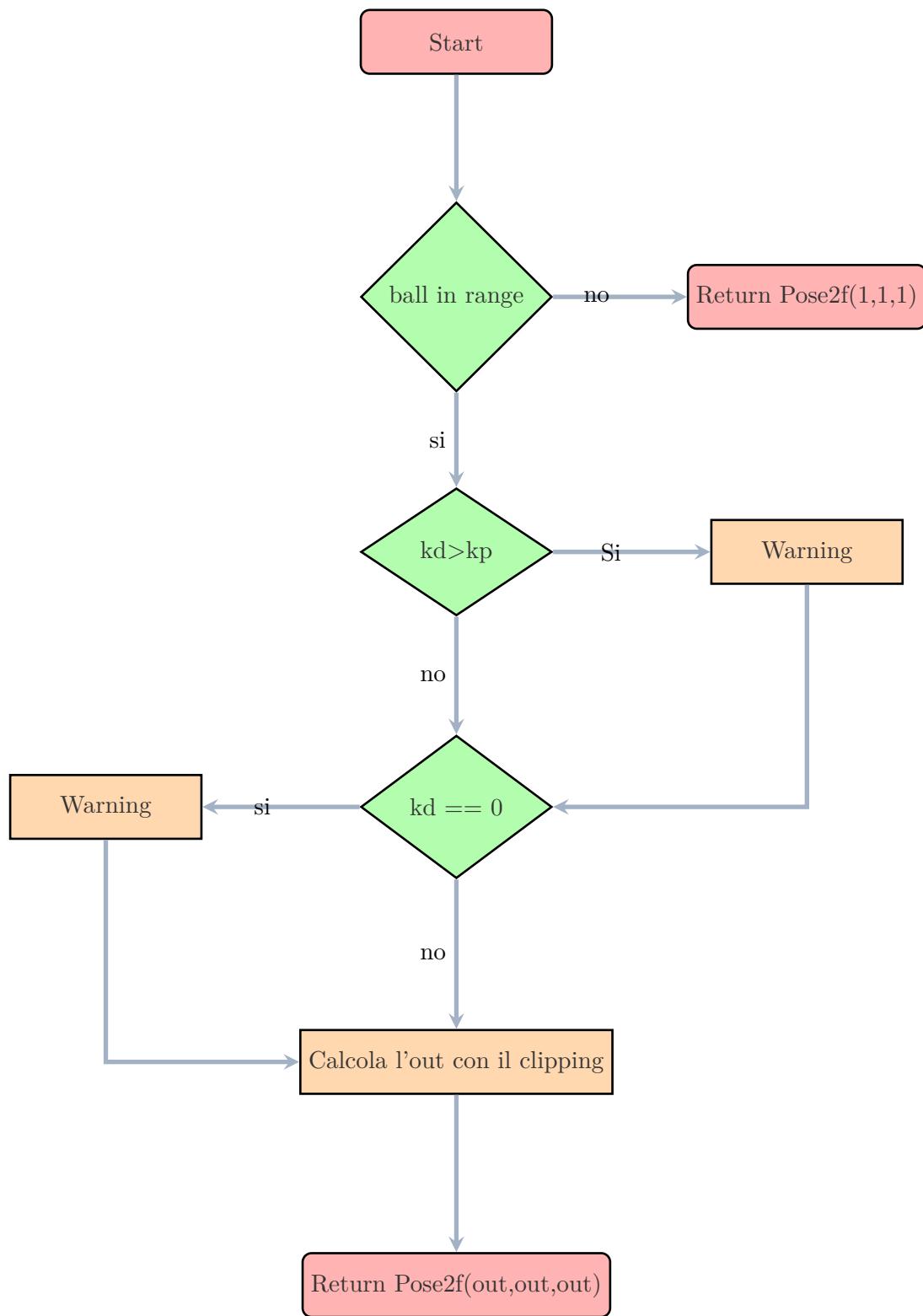
La funzione `mapToRange` è progettata per mappare un valore `val` da un certo intervallo di input `[minInput, maxInput]` a un intervallo di output `[minOutput, maxOutput]`. La funzione utilizza l'operatore di "clipping" `clip` per assicurarsi che il valore di input `val` sia all'interno dell'intervallo di input specificato.

La formula principale per il mapping è la seguente:

$$result = minOutput + \frac{(v - minInput) \cdot (maxOutput - minOutput)}{sizeOfInputRange}$$

Dove:

- **v**: Il valore di input "clippato" nell'intervallo $[minInput, maxInput]$.
- **sizeOfInputRange**: La lunghezza dell'intervallo di input, calcolata come $(maxInput - minInput)$.
- **sizeOfOutputRange**: La lunghezza dell'intervallo di output, calcolata come $(maxOutput - minOutput)$.
- **result**: Il valore mappato nell'intervallo di output.



3.4 Nuova implementazione del Controllore PID

```

1  /**
2  * PID controller that returns the approaching speed.
3  * @param range The distance range taken to pass from speed 1 to the
4  * specified minimum speed
5  * @param kp The controller proportional gain
6  * @param kd The controller derivative gain
7  * @param ki The controller integral gain
8  * @param minSpeed The minimum speed allowed
9  * @return speed in Pose2f in a range from minSpeed to 1
10 */
11 static float integral = 0.0f;
12 float previousError = 0.0f;
13 const float MAX_INTEGRAL = 50.0f;
14 std::chrono::high_resolution_clock::time_point lastCallTime = std::
15     chrono::high_resolution_clock::time_point::min();
16 Pose2f LibStrikerProvider::myGetApproachSpeed(Rangef range, float kp,
17     float kd, float ki, float minSpeed) const{
18     Vector2f position = theFieldBall.recentBallPositionRelative();
19
20     Angle rot = (theRobotPose.inversePose * position).angle();
21
22     std::cout << "l'angolo di: " << rot.toDegrees() << "\n";
23     if(position.norm() > range.max){
24         std::cout << "position.norm > range.max:" << position.norm() << "
25         > " << range.max << "\n";
26         return Pose2f(1,1,1);
27     }
28     if(kd>kp){
29         std::cout << "kd > kp\n";
30         OUTPUT_WARNING("Derivative gain is greater than proportional gain
31             , this may lead to an extremely slow convergence");
32     }
33     if(kd == 0){
34         std::cout << "kd == 0\n";
35         OUTPUT_WARNING("Derivative gain is set to 0, this may lead a to
36             lack of convergence due to rippling");
37     }
38
39     Vector2f target = goalTarget(true, true);
40
41     auto currentTime = std::chrono::high_resolution_clock::now();
42     float dt = 0.0f;
43
44     if (lastCallTime.time_since_epoch().count() != 0){
45         std::chrono::duration<float> duration = currentTime -
46         lastCallTime;
47         dt = duration.count();
48         std::cout << "dt: " << dt << "\n";
49     }
50
51     lastCallTime = currentTime;
52
53     float e_angle = abs((theRobotPose.inversePose * target).angle());
54     std::cout << "e_angle calcolato: " << e_angle << "\n";
55
56 }
```

```

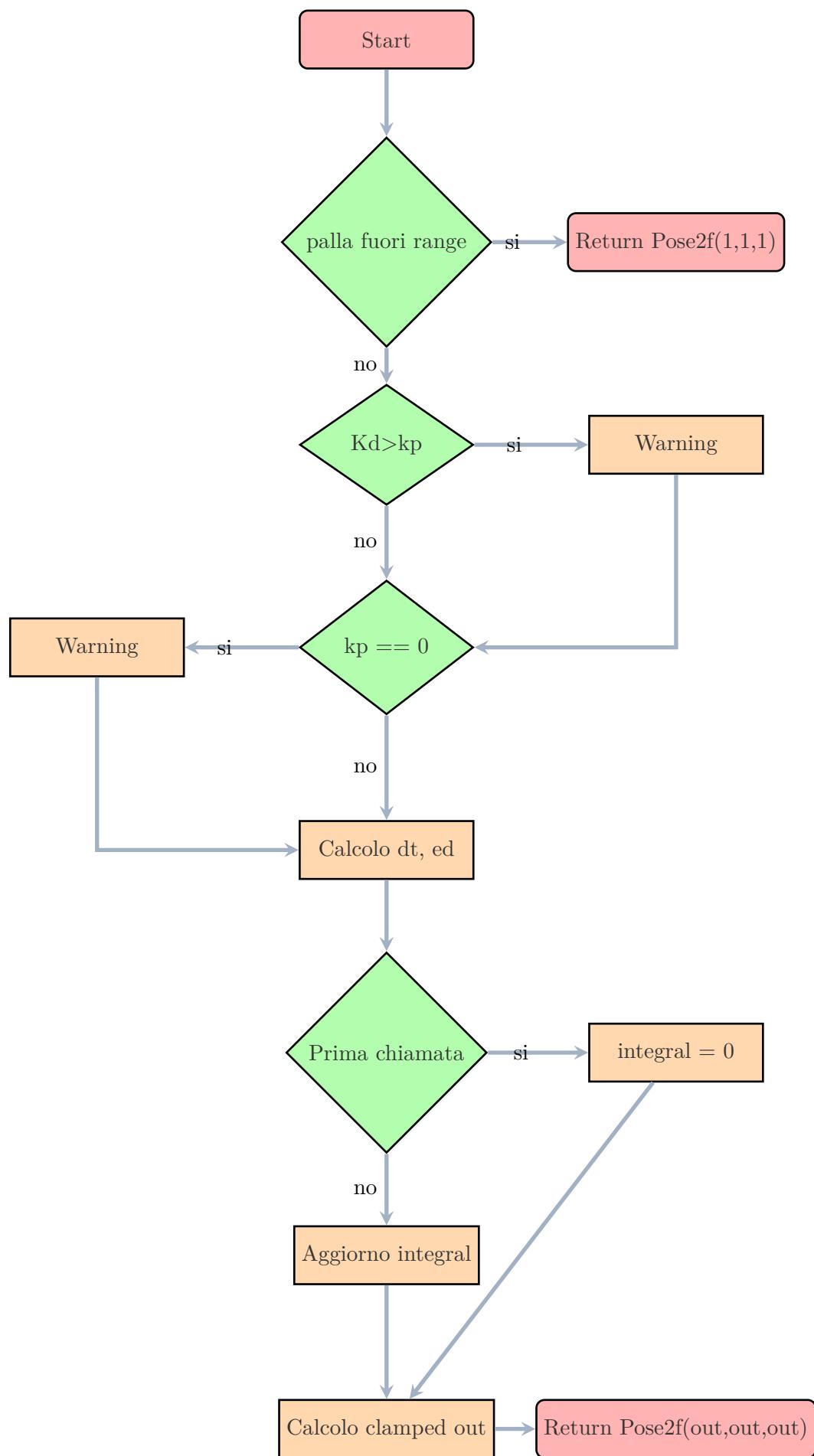
49
50     float e_p = mapToRange(theFieldBall.recentBallPositionRelative().
51         norm() + e_angle, range.min, range.max, minSpeed, 1.f);
52     std::cout << "e_p calcolato: " << e_p << "\n";
53
54     float e_d = calcDerivative(e_p, dt);
55     std::cout << "e_d calcolato: " << e_d << "\n";
56
57     if (dt > 0){
58         integral += e_p*dt;
59         if (integral > MAX_INTEGRAL){
60             integral = MAX_INTEGRAL;
61         }
62         if (integral < -MAX_INTEGRAL){
63             integral = -MAX_INTEGRAL;
64         }
65     }
66     else{
67         integral = 0;
68         std::cout << "integral pre dovrebbe 0: " << integral << "\n";
69     }
70
71     std::cout << "integral " << integral << "\n";
72
73     float output = kp*e_p + ki*integral + kd*e_d;
74     std::cout << "output: " << output << "\n";
75
76     float approachSpeed = customClamp(output, minSpeed, 1.0f);
77     std::cout << "approachSpeed: " << approachSpeed << "\n";
78
79     return Pose2f(approachSpeed, approachSpeed, approachSpeed);
80 }
81 /**
82 * @param value the value to be clamped
83 * @param low the inferior limit
84 * @param high the superior limit
85 * @return the value clamped in range (low, high)
86 */
87 float LibStrikerProvider::customClamp(float value, float low, float
88                                         high) const{
89     return low + std::max(0.0f, std::min(value, high-low));
90 }
91 float LibStrikerProvider::calcDerivative(float currentError, float dt
92                                         ) const{
93     if (dt == 0){
94         return 0.0f;
95     }
96     float derivativeError = (currentError - previousError)/dt;
97     previousError = currentError;
98     return derivativeError;
99 }
```

Analizziamo il codice:

- 10-14: Definiamo le variabili che ci serviranno durante l'esecuzione della funzione. `integral` viene definita come static e rappresenta l'accumulo degli errori poichè vogliamo che il suo valore non si azzeri tra una chia-

mata e l'altra, `previousError` è una variabile che useremo per il calcolo derivativo nel controllore, `MAX_INTEGRAL` è una costante che rappresenta il valore massimo che può essere assunto integral per evitare il windup, `lastCallTime` è una variabile che verrà inizializzato al valore minimo possibile per `std::chrono::high_resolution_clock::time_point`, indicando che nessuna chiamata precedente è stata registrata.

- 15-33: Vengono definite le variabili `position` e `rot` che rappresentano rispettivamente la posizione della palla relativa al campo e l'angolo calcolato dalla posizione inversa del robot e dalla posizione della sfera. Vengono effettuate alcune stampe di debug, altre di warning e viene definito come agire nel caso in cui la palla fosse ad una distanza maggiore rispetto a quella massima definita in `range`. Infine viene salvato nella variabile `target` la posizione espressa in coordinate cartesiane del pallone.
- 35-44: viene calcolato il Delta Tempo tra una chiamata e l'altra.
- 46-55: Vengono effettuati calcoli relativi al controllore PID: `e_angle` rappresenta l'angolo assoluto tra la posizione inversa del robot e il target desiderato, `e_p` è il termine porporzionale e tiene conto della distanza della palla dalla posizione corrente del robot e dell'angolo tra il robot e il target. La funzione `mapToRange` normalizza questo valore nel range specificato. `e_d` rappresenta il termine derivativo, calcolato utilizzando la funzione `calcDerivative`. Questa funzione calcola la derivata dell'errore rispetto al tempo e la sua implementazione si può osservare da riga 89 a riga 96.
- 56-69: In sintesi, questo passaggio si occupa di calcolare e aggiornare il termine integrale del controllore PID, prendendo in considerazione la variazione di tempo tra le chiamate successive alla funzione. L'aggiornamento del termine integrale è vincolato per evitare fenomeni indesiderati come il windup dell'integrale.
- 70-79: In `output` viene calcolato la computazione che segue le regole dell'architettura del controllore PID, questo valore viene poi "clampato" tra la velocità minima ammessa e 1. Vengono infine prodotte altre stampe di debug e ritornata la velocità di approccio con una struttura `Pose2f`
- 80-96: Vengono implementante le funzioni ausiliare `calcDerivative` e `customClamp`



3.5 Confronto tra le due implementazioni

Esaminando i due codici, è evidente che la nuova implementazione del controllore PID include un termine integrale (`ki*integral`) nella formula di calcolo dell'output. Ciò rende il controller PID più complesso rispetto alla vecchia implementazione, che conteneva solo i termini proporzionali e derivativi. È importante notare che l'assenza del termine integrale nella prima implementazione lo identifica come un controllore PD (proporzionale derivativo).

I vantaggi di un controllore PD rispetto a uno PID sono la sua semplicità di implementazione e taratura. Tuttavia, presenta uno svantaggio significativo: è meno efficace nel gestire errori a regime. Quando il sistema si avvicina al valore desiderato, il termine integrale in uno PID contribuisce ad accumulare e correggere gradualmente l'errore residuo, stabilizzando il sistema e rendendolo più preciso.

Inoltre, la mancanza del termine integrale nel controllore PD può comportare una maggiore sensibilità ai disturbi. Poiché non vi è compensazione per gli errori cumulativi nel tempo, il controllore PD potrebbe essere più suscettibile ai disturbi a lungo termine. In sintesi, sebbene il controllore PD sia più semplice da implementare e tarare, lo PID offre una maggiore precisione e stabilità nel gestire situazioni a regime.

Nella nuova implementazione c'è un diverso tipo di approccio alla gestione dell'intervallo di tempo (`dt`): viene utilizzata `std::chrono`[3] per calcolare la durata tra le chiamate alla funzione e di conseguenza modificare il calcolo del termine derivativo. Nella prima implementazione, il calcolo di `e_d` avviene utilizzando la norma della velocità del robot (`theMotionInfo.speed.translation.norm()`) e dividendo il risultato per 1000 (1e3). Questo fornisce un'approssimazione della derivata dell'errore rispetto al tempo (`e_d`), `dt` viene quindi fissato manualmente. Nella seconda implementazione, invece, il calcolo di `dt` è effettuato utilizzando la libreria `<chrono>`. Viene misurato il tempo trascorso tra le chiamate alla funzione mediante `std::chrono::duration`; `dt` viene quindi calcolato in modo dinamico basato sul tempo di sistema. Successivamente, `dt` viene utilizzato nel calcolo dell'errore derivativo `e_d` nella funzione `calcDerivative`.

In sintesi, la nuova implementazione offre l'aggiunta del termine integrale, una gestione più accurata dell'intervallo di tempo e un calcolo più preciso della derivata rispetto alla vecchia implementazione. Questi vantaggi possono tradursi in una maggiore precisione e stabilità del sistema di controllo, specialmente in situazioni in cui è fondamentale catturare variazioni temporali più dettagliate. L'approccio bilanciato tra i componenti PID contribuisce a un comportamento più prevedibile e reattivo.

Pur essendo dotata di nuove funzionalità che portano a diversi benefici, la nuova implementazione presenta anche alcune potenziali sfide e svantaggi che devono essere considerati:

- **Complessità Aggiuntiva:** L'aggiunta del termine integrale e la gestione più dettagliata dell'intervallo di tempo aumentano la complessità della logica di controllo. Una maggiore complessità può rendere il codice più difficile da comprendere, mantenere e debuggare.
- **Sensibilità ai Parametri PID:** Un controllore PID è sensibile alla sintonizzazione dei suoi parametri (k_p , k_i , k_d). La presenza del termine integrale può portare a instabilità se il parametro k_i è troppo elevato, mentre un k_d eccessivamente alto può causare oscillazioni indesiderate. La necessità di sintonizzare accuratamente i parametri può richiedere tempo e sforzi.
- **Possibilità di Integrazione di Rumore:** L'aggiunta del termine integrale può introdurre la possibilità di integrare il rumore nel sistema. Se il rumore è presente nell'errore, il termine integrale può accumularlo nel tempo, portando a una risposta instabile. Un'eccessiva integrazione del rumore può rendere il sistema sensibile a perturbazioni esterne.
- **Overfitting ai Dati di Test:** Una sintonizzazione accurata dei parametri PID potrebbe basarsi su specifici scenari di test, rischiando di non generalizzare bene a condizioni reali. Ciò potrebbe comportare problemi se il sistema viene utilizzato in ambienti operativi diversi da quelli di test.
- **Ulteriore utilizzo di Risorse Computazionali:** L'aggiunta di nuove funzionalità e la gestione più dettagliata dell'intervallo di tempo possono richiedere una maggiore potenza di calcolo, specialmente in sistemi embedded o con risorse limitate.

In conclusione, l'ottimizzazione del controller PID presenta notevoli vantaggi in termini di precisione e stabilità, ma richiede un approccio bilanciato considerando anche la complessità aggiuntiva e le sfide di sintonizzazione.

Capitolo 4

Risultati e analisi

4.1 Procedura Sperimentale

Dal momento che il codice non è deterministico, poiché la chiamata alla funzione `getApproachSpeed` avviene in base ad altre computazioni che esulano dallo sviluppo del controllore PID, portando il robot nello stato di "goToBall&kick" in diverse posizioni e circostanze, per valutare l'efficacia della mia implementazione ho deciso di procedere nel seguente modo: ho fornito a entrambe le funzioni gli stessi valori di range di attivazione, guadagno proporzionale e guadagno derivativo.

La scelta di questi parametri è stata effettuata attraverso un'esplorazione di "grid search" su un set di valori. Questa operazione coinvolge la definizione di un insieme di valori candidati per ciascun parametro del modello e l'esecuzione del modello su tutte le combinazioni possibili di questi valori. Ho selezionato un insieme di tre valori per ciascun parametro e ho valutato la miglior combinazione per entrambi i modelli. Dopo aver scelto i tre valori che hanno fornito i risultati migliori per entrambi i modelli, ho eseguito 25 simulazioni per ciascuna implementazione.

Al fine di valutare al meglio l'approccio al pallone, ho modificato lo scenario `bh.ros2`, eliminando i robot avversari (che comunque erano immobili durante la simulazione) per analizzare più accuratamente i tempi di approccio. Nella configurazione iniziale, come si può osservare dalla 4.1, è presente un orologio che parte da 10:00 (mm:ss) e un robot avversario che funge da portiere, rimanendo immobile. Il portiere è stato mantenuto perché, nella fase di tiro, la sua posizione immobile serve da punto di riferimento cruciale. Questo ci consente di osservare se il robot riesce a calciare il pallone evitando l'ostacolo del portiere e centrando la porta con precisione, mette infatti alla prova la capacità del robot di calcolare la traiettoria e la forza necessarie per superare l'ostacolo e raggiungere l'obiettivo desiderato.

Le condizioni di partenza di ogni simulazione sono quindi identiche per valutare al meglio le peculiarità di entrambe le implementazioni. Ho scelto di eseguire 25 simulazioni per garantire una validità statistica ai risultati ottenuti. Andiamo ora a definire le metriche di valutazione usate.

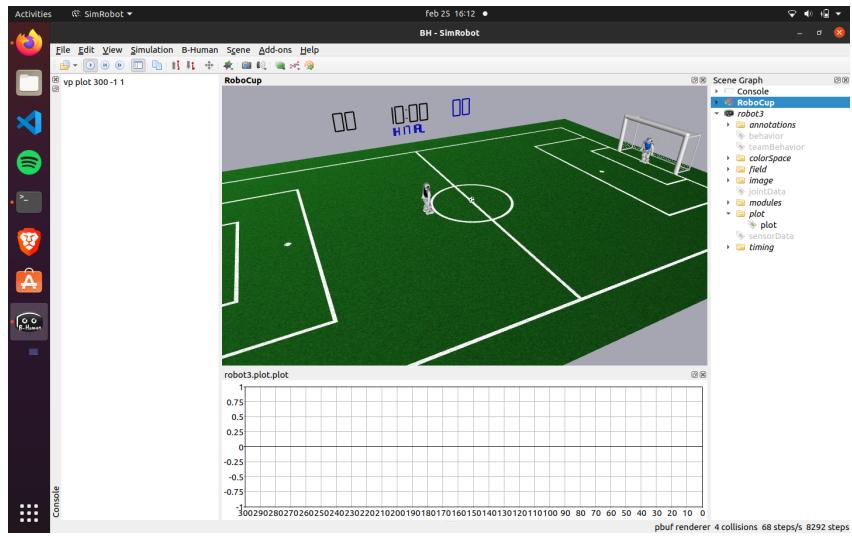


Figura 4.1. Configurazione iniziale

4.2 Metriche di Valutazione

Le metriche di valutazione usate nella nostra analisi sono tre: sovraelongazione, tempo di approccio, conversione dei tiri in goal.

La sovraelongazione, o overshooting in inglese, è una misura che indica quanto un sistema va oltre il valore desiderato prima di stabilizzarsi o raggiungere il punto di set, (la posizione target che il sistema cerca di raggiungere e mantenere) ed è una metrica critica per diverse ragioni. Innanzitutto, essa influisce direttamente sulla precisione del posizionamento del robot rispetto al pallone. Un'eccessiva sovraelongazione potrebbe significare che il robot si avvicina al pallone con una velocità troppo elevata, superando la posizione desiderata e richiedendo tempo aggiuntivo per correggere l'errore. Inoltre, una sovraelongazione significativa può portare a oscillazioni indesiderate o instabilità nel sistema, compromettendo l'efficacia dell'appuccio al pallone. Un controllo troppo aggressivo potrebbe causare movimenti oscillatori o comportamenti erratici, influenzando negativamente la precisione del tiro e la capacità del robot di mantenere una traiettoria controllata; se dovesse essere troppo eccessiva potrebbe indicare la necessità di ottimizzare i parametri del PID per garantire un comportamento più preciso e stabile durante la fase di tiro. Monitorare attentamente questa metrica può fornire indicazioni cruciali per il perfezionamento delle implementazioni e migliorare l'efficacia globale del sistema di controllo del robot.

Per quanto detto sopra, in questo contesto, una sovraelongazione più bassa è preferibile rispetto ad una maggiore.

Il tempo di approccio è una metrica che indica il tempo che intercorre dall'inizio della simulazione al momento di tiro del robot. Esso riflette la tempestività, l'efficienza e la precisione del robot durante la fase di tiro. Un tempo di approccio ottimizzato contribuisce significativamente al successo delle azioni del robot sul campo e alla sua capacità di affrontare situazioni dinamiche in modo efficace. Intuibilmente

un tempo di approccio minore è preferibile ad uno maggiore.

Infine l'ultima metrica che teniamo in considerazione è il tasso di conversione in goal dei tiri effettuati. Un tasso di conversione più alto suggerisce una maggiore precisione e successo nell'implementazione delle azioni offensive, mentre un tasso più basso potrebbe indicare la necessità di ottimizzare l'approccio al tiro o la precisione nel controllo del pallone.

Per ottenere una comprensione più dettagliata della procedura di raccolta dei dati, facciamo riferimento alla figura 4.2. La simulazione è stata progressivamente avanzata step by step, considerando una frequenza di rendering delle immagini di 10 FPS (frames per secondo). Nella parte superiore al centro della figura, si osserva che l'orologio di countdown indica le 9:37. Ciò implica che il tempo di approccio è di 23 secondi ($10 : 00 - 9 : 37 = 00 : 23$ mm:ss).

Nel riquadro in basso della figura è presentato il grafico della velocità di approccio. Per ottenere valori precisi della velocità massima e della velocità a regime permanente, questi sono stati stampati su un terminale durante l'esecuzione della simulazione.

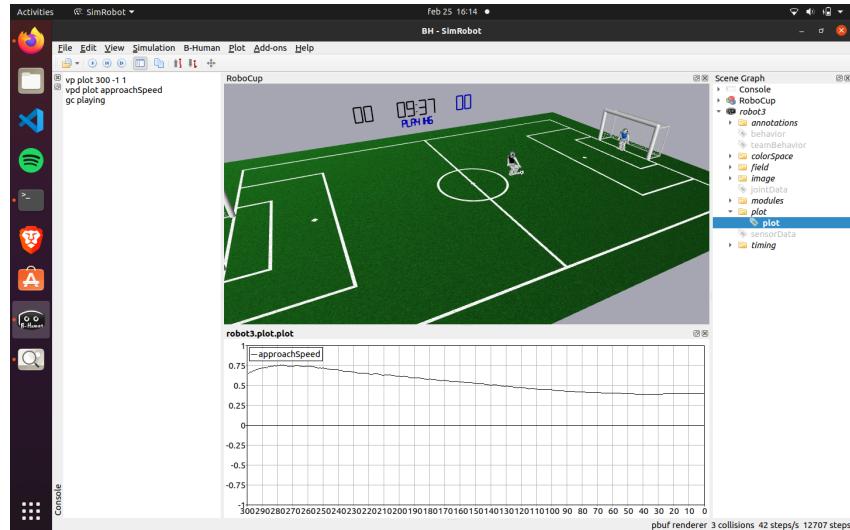


Figura 4.2. Istante di tiro di una simulazione

4.3 Risultati numerici

Nel contesto della presentazione dei risultati numerici, ho optato per l'utilizzo di tabelle al fine di fornire una rappresentazione chiara e organizzata delle misurazioni ottenute. Questa scelta mira a garantire una presentazione efficiente e accessibile dei dati numerici raccolti nel corso dell'esperimento, consentendo una valutazione agevole delle prestazioni delle diverse implementazioni e sottolineando i principali risultati ottenuti.

Di seguito sono fornite le principali componenti strutturali delle tabelle:

- colonne:

- Y_Max: rappresenta il massimo valore raggiunto dalla velocità d'approccio del robot.
- Y_out: rappresenta il valore di output a regime permanente raggiunto dalla velocità d'approccio del robot.
- Sovraelongazione: rappresenta la sovraelongazione ed è calcolata con la seguente formula:

$$\frac{Y_{Max} - Y_{out}}{Y_{out}}$$

- Countdown(mm:ss): Il tempo rimanente in formato mm:ss al momento del tiro.
- Tempo (ss): Il tempo totale trascorso tra lo start e il momento del tiro
- Goal: Indica se la simulazione ha portato a un gol (1) o meno (0).
- righe dei dati: Ciascuna riga rappresenta una singola simulazione.
- righe aggiuntive: Le righe "Min", "Max", "Media", "Mediana", e "Deviazione Standard" forniscono statistiche riassuntive dei risultati ottenuti, offrendo una panoramica complessiva delle prestazioni del sistema in termini di precisione, tempestività e successo nei tiri.

New	Y_Max	Y_out	Sovraelongazione	Countdown(mm:ss)	Tempo(ss)	Goal
1	0,49	0,44	0,1136	9,45	0,15	1
2	0,49	0,44	0,1136	9,44	0,16	1
3	0,45	0,44	0,0227	9,44	0,16	1
4	0,90	0,41	1,1951	9,44	0,16	1
5	0,60	0,44	0,3636	9,44	0,16	1
6	0,59	0,44	0,3409	9,43	0,17	1
7	0,48	0,45	0,0667	9,43	0,17	1
8	0,51	0,44	0,1591	9,42	0,18	1
9	0,67	0,44	0,5227	9,42	0,18	1
10	0,57	0,44	0,2955	9,41	0,19	1
11	0,48	0,44	0,0909	9,41	0,19	0
12	0,62	0,45	0,3778	9,41	0,19	1
13	0,65	0,45	0,4444	9,41	0,19	0
14	0,65	0,45	0,4444	9,41	0,19	1
15	0,63	0,44	0,4318	9,41	0,19	0
16	0,72	0,42	0,7143	9,40	0,20	1
17	0,85	0,45	0,8889	9,40	0,20	0
18	0,70	0,45	0,5556	9,40	0,20	1
19	0,60	0,44	0,3636	9,40	0,20	1
20	0,46	0,44	0,0455	9,40	0,20	1
21	0,76	0,44	0,7273	9,39	0,21	1
22	0,70	0,45	0,5556	9,38	0,22	1
23	0,82	0,40	1,0500	9,37	0,23	1
24	0,78	0,38	1,0526	9,36	0,24	1
25	1	0,40	1,5000	9,30	0,30	1
Min	0,45	0,38	0,0227	9,30	0,15	0
Max	1	0,45	1,50	9,45	0,30	1
Media	0,6468	0,4352	0,4975	9,4068	0,1932	0,84
Mediana	0,63	0,44	0,4318	9,4100	0,19	1
Dev Standard	0,1328	0,0103	0,3453	0,0096	0,0096	0,0000

Tabella 4.1. Tabella dati nuova implementazione

OLD	Y_Max	Y_out	Sovraelongazione	Countdown(mm:ss)	Tempo(ss)	Goal
1	0,29	0,27	0,0741	9,45	0,15	0
2	0,30	0,29	0,0345	9,45	0,15	1
3	0,37	0,29	0,2759	9,43	0,17	0
4	0,30	0,29	0,0345	9,43	0,17	1
5	0,35	0,29	0,2069	9,42	0,18	1
6	0,35	0,29	0,2069	9,42	0,18	0
7	0,41	0,29	0,4138	9,41	0,19	1
8	0,41	0,29	0,4138	9,41	0,19	1
9	0,53	0,31	0,7097	9,41	0,19	1
10	0,50	0,31	0,6129	9,40	0,20	0
11	0,35	0,29	0,2069	9,40	0,20	1
12	0,47	0,29	0,6207	9,39	0,21	0
13	0,55	0,29	0,8966	9,38	0,22	1
14	0,54	0,29	0,8621	9,37	0,23	1
15	0,51	0,29	0,7586	9,37	0,23	1
16	0,44	0,30	0,4667	9,37	0,23	1
17	0,55	0,29	0,8966	9,36	0,24	1
18	0,42	0,29	0,4483	9,36	0,24	0
19	0,35	0,29	0,2069	9,35	0,25	1
20	0,61	0,30	1,0333	9,35	0,25	1
21	0,70	0,27	1,5926	9,34	0,26	1
22	0,78	0,26	2,0000	9,32	0,28	1
23	0,97	0,26	2,7308	9,32	0,28	1
24	1,00	0,26	2,8462	9,26	0,34	1
25	1,00	0,25	3,0000	9,25	0,35	1
Min	0,29	0,25	0,0345	9,25	0,15	0
Max	1	0,31	3	9,45	0,35	1
Media	0,5220	0,2856	0,8620	9,3768	0,2232	0,76
Mediana	0,47	0,29	0,6129	9,38	0,22	1
Dev Standard	0,0784	0,0108	0,2240	0,0162	0,0162	0,4899

Tabella 4.2. Tabella dati vecchia implementazione

4.4 Analisi dei risultati

L'analisi dei risultati procederà nel seguente modo: inizieremo esaminando i dati per un'implementazione e successivamente per l'altra, procedendo colonna per colonna. Infine, verrà effettuato un confronto tra i risultati ottenuti dalle due implementazioni.

Iniziamo analizzando i dati relativi alla vecchia implementazione 4.3.

- **Y_max:** La distribuzione indica una variabilità significativa nella velocità massima raggiunta durante le simulazioni, con una deviazione standard del

7.84%. La media e la mediana si trovano a circa metà strada tra i valori minimo e massimo, indicando una distribuzione bilanciata.

- **Y_out:** La distribuzione mostra meno variabilità rispetto a Y_Max con una deviazione standard abbastanza bassa del 1.08%. La maggior parte delle simulazioni sembra convergere verso un valore di output simile.
- **Sovraelongazione:** La sovraelongazione mostra una varietà di comportamenti durante le simulazioni. La presenza di un valore massimo molto alto e la differenza tra media e mediana suggeriscono che alcune simulazioni possono essere meno stabili e influenzare significativamente la variabilità complessiva.
- **Tempo di approccio:** Il tempo totale trascorso varia tra 0.15 e 0.35 secondi, media e mediana risultano a circa metà strada e combaciano, indice quindi di una distribuzione bilanciata.
- **Goal:** La media realizzativa è del 76%.

Per quanto riguarda i dati relativi alla nuova implementazione riportati nella 4.2:

- **Y_Max:** I dati indicano una notevole variabilità, con valori compresi tra 0,45 e 1. La deviazione standard suggerisce una significativa dispersione rispetto ai valori medi, con alcune simulazioni che raggiungono velocità molto elevate.
- **Y_out:** La variabilità è più contenuta rispetto a Y_Max, indicando una maggiore stabilità nei valori di output a regime permanente e indicando un valore di convergenza più preciso.
- **Sovraelongazione:** La sovraelongazione mostra una variabilità considerevole, con alcuni valori estremamente alti. La deviazione standard elevata indica una distribuzione più ampia, con alcune simulazioni che presentano sovraelongazione notevolmente superiore alla media.
- **Tempo di approccio:** Il tempo totale trascorso varia tra 0,15 e 0,30 secondi, indicando una buona coerenza nei tempi di approccio.
- **Goal:** L'indicatore di successo mostra che, in media, l'84% delle simulazioni ha portato a un gol.

Per quanto riguarda entrambe le implementazioni l'elevata deviazione standard è dovuta al fatto che la chiamata alla funzione non avviene sempre nelle stesse condizioni, dipende soprattutto da come viene calciata la palla durante lo stato GoToBall&Dribble e di conseguenza dalla posizione e dalla velocità della palla nel momento in cui si passa allo stato di GoToBall&Kick.

Osserviamo che entrambe le implementazioni convergono ad una velocità di regime ben definita entrambe con una deviazione standard molto bassa sull'ordine del 1%, ciò implica quindi una buona stabilità nelle prestazioni del sistema. La presenza di una variazione limitata nella velocità a regime permanente suggerisce coerenza e affidabilità nelle risposte del sistema. Per quanto riguarda il confronto

diretto tra i due valori di convergenza osserviamo come nella nuova implementazione il valore di convergenza medio (0,4352) sia maggiore rispetto a quello della vecchia implementazione (0,2856) il che come vedremo si riflette sulle statistiche della sovraelongazione e di conseguenza dei tempi di approccio al pallone.

È nel confronto della sovraelongazione che riusciamo ad apprezzare al meglio le differenze e le migliorie apportate dalla mia soluzione. La massima sovraelongazione raggiunta dalla nuova soluzione (1,5) risulta essere la metà della massima raggiunta dalla precedente (3) e che il valore minimo (0,0227) è a sua volta minore del valore minimo della vecchia (0,0345). Entrambe le misure soffrono di una grande deviazione standard dovuta dal contributo di Y_{\max} che porta grande deviazione per i motivi discussi sopra. Ciò che maggiormente enfatizza la supremazia della mia implementazione è l'analisi della sovraelongazione media: la mia infatti risulta essere poco più della metà (precisamente il 56%) del valore medio della vecchia realizzazione, ciò è indice di una migliore capacità del sistema di controllare e rispondere in modo più preciso e rapido alle variazioni delle condizioni operative. Più nello specifico:

- Una sovraelongazione più bassa indica una maggiore precisione nel controllo delle variabili del sistema. Il sistema raggiunge e mantiene l'obiettivo con un minor "sorpasso" rispetto alla configurazione precedente
- Una sovraelongazione inferiore suggerisce una risposta più rapida alle perturbazioni. Il sistema si adatta più rapidamente e stabilizza la sua uscita senza eccessive oscillazioni.
- La riduzione della sovraelongazione contribuisce alla stabilità del sistema. Un controllo più preciso riduce il rischio di instabilità e comportamenti indesiderati, migliorando la robustezza complessiva del sistema.
- Il sistema raggiunge l'obiettivo desiderato in modo più diretto e con meno spreco di risorse dovute ad eventuali correzioni di traiettoria.
- Il tempo richiesto per raggiungere la stabilità dopo una perturbazione può essere ridotto con una sovraelongazione più bassa. Ciò contribuisce a tempi di stabilizzazione più brevi, migliorando la reattività del sistema.

Ritengo importante inoltre sottolineare una certa consistenza e uno scarto esiguo tra il valore medio (0,4975) e mediano (0,4318) della mia implementazione rispetto alla differenza tra il valore medio (0,8620) e mediano (0,6129) della vecchia implementazione.

Di riflesso avendo avuto un netto miglioramento nella sovraelongazione troviamo le stesse migliorie anche nei tempi di approccio, seppur il valore minimo assunto da entrambe le implementazioni sia lo stesso (0,15) il valore massimo risulta essere minore nella nuova implementazione di ben 5 secondi, valore medio e mediano di ciascuna implementazione è simile, con uno scarto di circa tre secondi in favore della mia.

Di seguito sono presentati due grafici 4.3 e 4.4 in cui sull'asse delle ascisse (x) saranno rappresentati gli intervalli di tempo di approccio, variando da 0.15 a 0.40 secondi. Le barre sul grafico mostreranno il numero di simulazioni che rientrano in ciascun intervallo di tempo di approccio, con altezze variabili a seconda della distribuzione delle simulazioni in quegli intervalli.

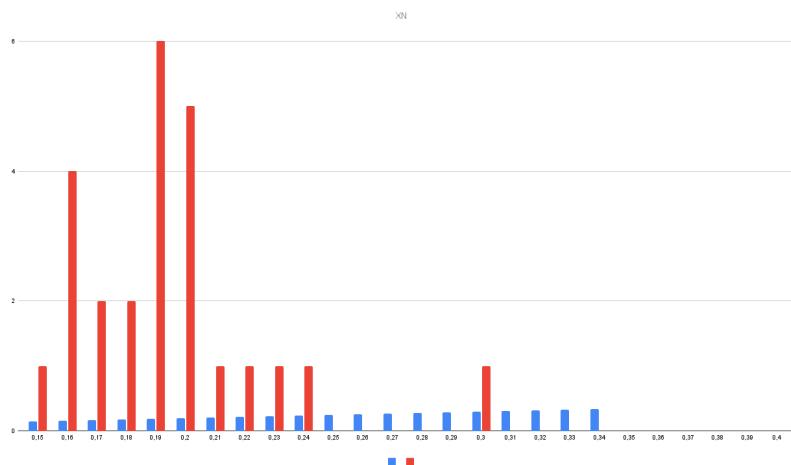


Figura 4.3. Distribuzione della nuova implementazione

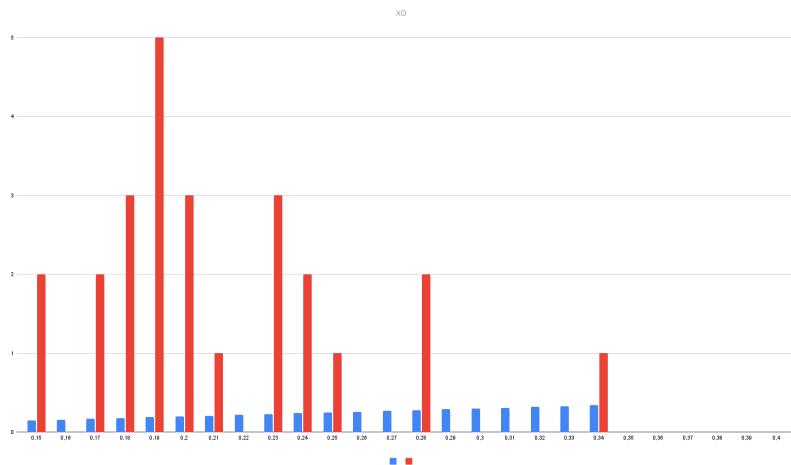


Figura 4.4. Distribuzione della vecchia

Analizziamo ora due campioni di grafico particolarmente significativi relativi alla velocità di avvicinamento ("approach speed") appartenenti uno alla vecchia implementazione e uno alla nuova.

Come evidenziato nelle figure 4.5 e 4.6, entrambe le funzioni sono prevalentemente decrescenti, presentando una forma a "dente di sega" più evidente nella figura 4.5 rispetto alla 4.6. Questa caratteristica indica l'intervento delle componenti propor-

zionale e derivativa del controllore. Nel grafico della nuova implementazione, l'effetto a "dente di sega" è più attenuato grazie all'intervento della componente integrale, che contribuisce a ridurre le oscillazioni e a migliorare la stabilità complessiva del sistema.

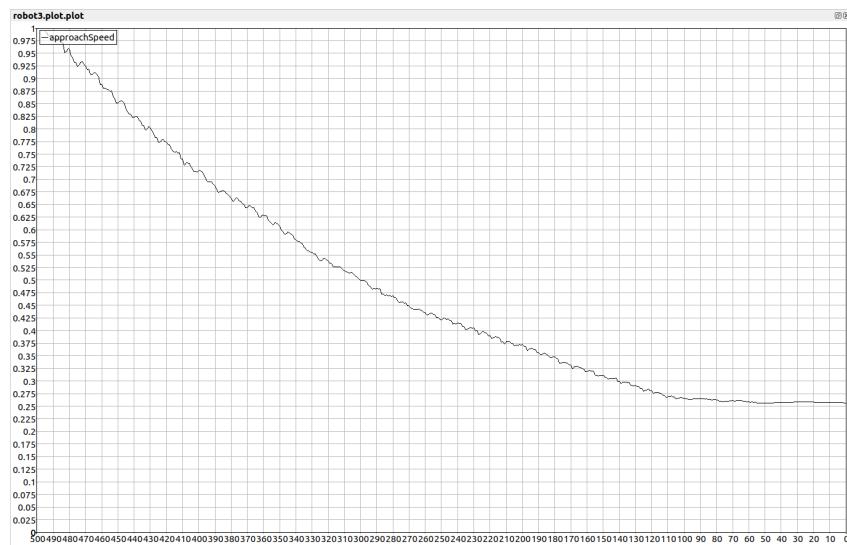


Figura 4.5. Grafico di approachSpeed della vecchia implementazione

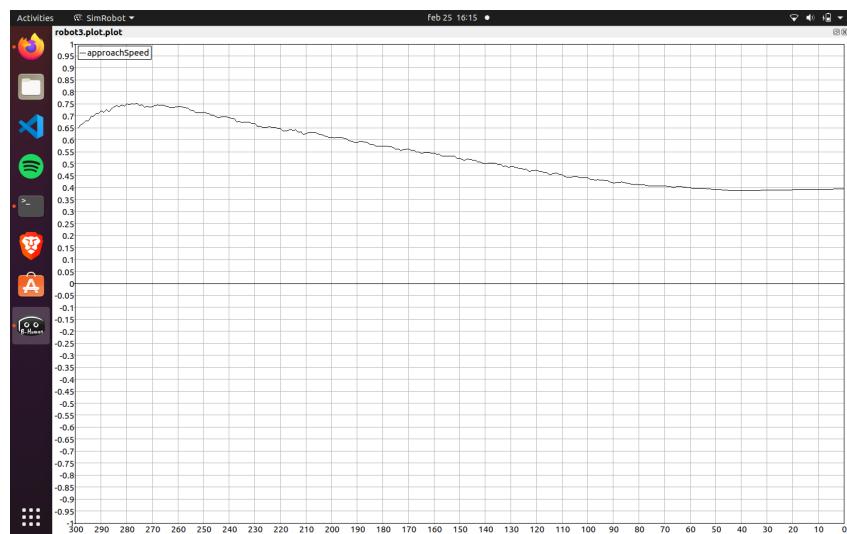


Figura 4.6. Grafico di approachSpeed della nuova implementazione

Capitolo 5

Conclusioni

In conclusione, il settore della robocup e lo sviluppo di controllori robotici rappresentano un ambito di ricerca in continua evoluzione. Nonostante gli attuali progressi di punta, siamo ancora alla ricerca di nuove opportunità e innovazioni per migliorare ulteriormente le prestazioni e l'efficienza di tali sistemi.

Dall'analisi dei dati emerge chiaramente che la nuova implementazione è generalmente più performante della precedente. La sovraelongazione media è inferiore, l'uscita a regime permanente è mediamente superiore e viene raggiunta in un tempo minore. Tuttavia, è importante considerare alcuni aspetti aggiuntivi.

La mia soluzione è più complessa: il controllore nella vecchia implementazione manca di una componente integrale, trasformando di fatto il controllore da PID a PD. Inoltre, la mia implementazione prevede un calcolo dinamico del delta tempo, a differenza della vecchia implementazione che utilizza un delta tempo statico. Entrambi questi elementi potrebbero rappresentare un problema in un sistema embedded come il robot NAO. I sistemi embedded operano spesso con risorse hardware limitate, come memoria e potenza di calcolo. Queste limitazioni possono influenzare le prestazioni e la complessità delle operazioni gestite dal sistema. Pertanto, è fondamentale valutare attentamente l'impatto di tali considerazioni sulla praticità e sull'efficacia dell'implementazione.

Tuttavia, per garantire una valutazione più completa delle prestazioni della mia implementazione, è necessario condurre ulteriori simulazioni sia al simulatore che sul robot in modo tale da ottenere maggiori conferme riguardo a quanto è emerso dall'analisi dei dati.

Possibili ulteriori sviluppi del lavoro potrebbero includere l'utilizzo di una grid search più estesa per esplorare i parametri e rendere la sintonizzazione più accurata. Si potrebbero anche testare diverse tecniche di ottimizzazione, come la Ziegler-Nichols [6], una tecnica classica per ottenere una stima iniziale dei parametri di un controllore PID. Questo metodo si basa sull'osservazione della risposta al gradino di un sistema dinamico e prevede la determinazione di tre parametri chiave: la banda proporzionale, il periodo integrale e il tempo derivativo.

Un passo ulteriore potrebbe consistere nel confrontare i controllori attuali con controllori con architetture diverse come il Model Predictive Controller (MPC) oppure un controllore basato sulla Logica Fuzzy. Il controllore in logica fuzzy si basa sulla teoria dei sistemi fuzzy, un approccio che consente di gestire l'incertezza

e la vaghezza nei dati di input. Questo tipo di controllore utilizza insiemi fuzzy, regole linguistiche e inferenza fuzzy per tradurre input imprecisi in azioni di controllo definite. Gli insiemi fuzzy permettono di rappresentare concetti come "piccolo", "medio" e "grande" attraverso funzioni di appartenenza, mentre le regole fuzzy collegano gli insiemi di input a quelli di output. L'inferenza fuzzy combina queste regole, producendo un output fuzzy che viene infine defuzzificato per ottenere il valore di controllo.

Dall'altra parte, il controllore Model Predictive Control (MPC) adotta un approccio più avanzato basato sulla previsione del comportamento futuro del sistema. Con un modello matematico del sistema espresso attraverso equazioni differenziali o differenze, il MPC opera in modo predittivo. Inizialmente, il modello viene discretizzato e si definisce un orizzonte di previsione. Il controllore calcola quindi una sequenza di azioni di controllo ottimali, minimizzando una funzione di costo che tiene conto di vincoli sulle variabili di stato e di controllo. Solo la prima azione di controllo viene implementata, e questo processo iterativo si ripete ad ogni passo, consentendo al controllore di adattarsi alle dinamiche del sistema nel tempo.

Bibliografia

- [1] Simrobot - b-human, 2022. <https://wiki.b-human.de/master/simrobot/>.
- [2] Spqr team | official website of spqr team spl, 2022. <https://spqr.diag.uniroma1.it/>.
- [3] cppreference.com, 2023. https://en.cppreference.com/w/cpp/chrono/time_point.
- [4] H. Kitano, M. Asada, I. Noda, and H. Matsubara. Robocup: Robot world cup. *XRDS: Crossroads, The ACM Magazine for Students*, 4(3):8–10, 1998.
- [5] A. K. Mackworth. On seeing robots. In *Computer Vision: Systems, Theory and Applications*, pages 1–13. World Scientific, 1993.
- [6] P. Shahverdi, M. J. Ansari, and M. T. Masouleh. Balance strategy for human imitation by a nao humanoid robot. In *2017 5th RSI International Conference on Robotics and Mechatronics (ICRoM)*, pages 138–143. IEEE, 2017.
- [7] S. Shamsuddin, L. I. Ismail, H. Yussof, N. Ismarrubie Zahari, S. Bahari, H. Hashim, and A. Jaffar. Humanoid robot nao: Review of control and motion exploration. In *2011 IEEE International Conference on Control System, Computing and Engineering*, pages 511–516, 2011.