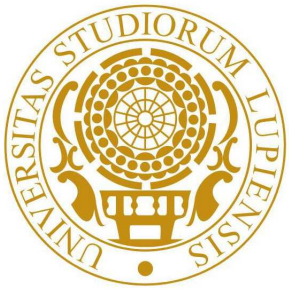
UNIVERSITÀ DEL SALENTO



Facoltà di Ingegneria

Corso di Laurea Triennale in Ingegneria dell'Informazione

Tesi di Laurea in

Teoria dei Sistemi

Sviluppo di un'architettura software per il controllo di robot cooperativi mobili

Relatore: **Ch.mo Prof. Giuseppe Notarstefano**

Correlatori: **Dott. Andrea Testa**

Dott.ssa Sara Spedicato

Studente: **Federico Vergallo**

Anno accademico 2016-2017

Abstract

I team di robot mobili cooperativi avranno un largo utilizzo pratico nel prossimo futuro. La ricerca su questo tema è florida e molti sono i contributi rigurardo la progettazione di strategie vincenti per il controllo dei robot.
Questa tesi affronta il problema del controllo di un team di robot mobili cooperativi.
Si vuole presentare un'architettura sviluppata per il controllo di un team multi robot, in cui l'obiettivo di controllo dell'architettura è far convergere lo stato del robot verso una posizione desiderata.
Si sono presi in considerazione dei robot di tipo uniciclo in uno spazio bidimensionale, modellati attraverso un simulatore, un controllore e un comunicatore della posizione desiderata. Nella prima parte della tesi viene presentato il framework ROS per la realizzazione di applicativi software per la robotica, utilizzato nell'implementazione dell'architettura. Successivamente, l'attenzione viene posta sulla modellazione della dinamica di un robot di tipo uniciclo e sulla modellazione dell'architettura proposta discutendo della strategia di controllo applicata.
Per lo sviluppo dell'architettura si è preso in esame uno scenario in cui si considerano dei robot in un rapporto leader-follower, in cui la traiettoria del robot leader converge in un punto desiderato mentre quella del follower insegue la traiettoria del leader.
La seconda parte della tesi riguarda l'implementazione software del modello proposto offrendo delle simulazioni nello scenario preso in esame.

Indice

Elenco delle figure

Introduzione

Motivazioni

Il campo della robotica cooperativa pone le sue radici durante la fine degli anni '80, quando molti ricercatori cominciarono ad investigare sugli aspetti dei sistemi di robot mobili multipli. Precedentemente a questo periodo, gli sforzi dei ricercatori erano concentrati verso i sistemi di robot singoli o i sistemi di problem-solving distribuiti, i quali non comprendevano componenti robotici.
Tuttora si possono trovare diverse aree in cui vi sono applicazioni di tipo robotico, come: esplorazione sottomarina e spaziale, operazioni di ricerca e salvataggio in ambienti pericolosi, servizi robotici nel privato e nel pubblico, compiti militari, trasporto cooperativo e molti altri. In questi ambienti, i sistemi multi robot spesso si trovano a svolgere dei compiti difficili da affrontare, se non impossibili, per un singolo robot.

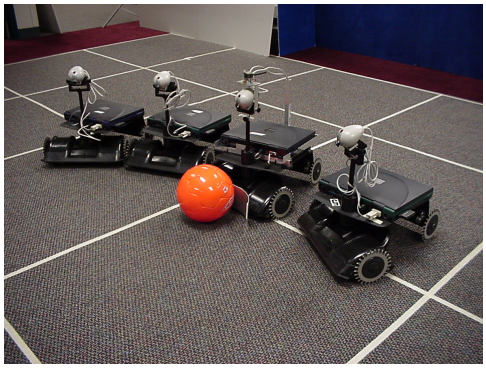


Figura 1: Robot che cooperano per spostare una palla

La gestione dei sistemi di robot multipli trova ispirazione nella biologia, in particolare nello studio dei comportamenti sociali di alcuni animali come api, formiche e uccelli. La conoscenza di queste semplici regole di controllo di questi animali ispira lo sviluppo di simili comportamenti nei sistemi di robot cooperativi. In particolare, si è mostrata la tendenza dei team multi

robot a radunarsi, disperdersi, aggregarsi, ricercare e seguire una traccia.
È logico pensare che nell'agire in modo cooperativo nell'esecuzione di un compito dato,un sistema di robot mobili possa garantire un'esecuzione più efficiente rispetto a quanto potrebbe fare un sistema a singolo robot.
In generale, i vantaggi di un sistema di robot multipli rispetto ad uno singolo sono:

- **Convenienza economica**: realizzare un robot multi obiettivo è meno vantaggioso da un punto di vista temporale ed economico rispetto alla realizzazione di molti a singolo obiettivo
- **Robustezza e affidabilità**: un sistema multi robot è più robusto e affidabile rispetto ad un sistema a singolo robot dato che non soffre del problema del 'single point of failure'
- **Azione distribuita**: robot multipli possono trovarsi in diverse aree e lavorare in sinergia su compiti differenti
- **Complessità**: progettare un singolo robot che svolga tutto il lavoro è spesso complesso e sconveniente

Stato dell'arte

Da qualche decennio sta prendendo forma un'ampia letteratura riguardante il campo dei sistemi di robot autonomi cooperativi, [1]. La modernità di tale campo si presta ad un'intensiva ricerca con l'obiettivo di esplorare le potenzialità e le possibili applicazioni che un sistema di robot mobili cooperativi può fornire.
L'articolo [1] affronta il tema dell'impostazione di una strategia di controllo di un sistema di robot mobili utilizzando una legge di controllo non lineare. Vengono trattati alcuni problemi basilari dei sistema di robot multipli, tra cui il controllo della distanza vettoriale tra robot. Esso consiste nel controllo della distanza di sicurezza tra robot nel piano attraverso l'uso di opportuni input di controllo comunicati ad ogni robot del team.
Alcuni degli approcci più usati sono basati sul controllo automatico, il quale si può decomporre in controllo longitudinale (regolazione della distanza) e controllo laterale (regolazione angolare), [1].
Spesso le tecniche di controllo longitudinale e laterale sono basate su controllori PID (proporzionale-integrativo-derivativo) o su altri metodi di controllo basati su retroazione, [1].

Contributi

Questa tesi vuole presentare un'architettura per il controllo di un team di robot mobili cooperativi sviluppata utilizzando l'innovativo framework ROS.

Ogni robot è composto da un simulatore che simula la dinamica del robot, un controllore e un comunicatore della posizione desiderata a cui lo stato del robot deve convergere. Viene preso in esame uno scenario leader-follower in cui la traiettoria del robot leader converge verso una posizione desiderata fissa, mentre l'evoluzione dello stato dei robot follower insegue la traiettoria del robot leader, avendo come posizione desiderata la posizione del leader.
La tesi propone una simulazione dell'architettura eseguita con un robot leader e uno follower.

Organizzazione della tesi

Nel primo capitolo viene introdotto ROS, lo strumento principale dell'architettura sviluppata, e ne vengono descritte le componenti e le funzionalità. Il secondo capitolo della tesi introduce il modello matematico della dinamica del robot con controllore di tipo uniciclo. Innanzitutto vengono introdotte le equazioni dello stato del sistema e si fornisce la descrizione di ogni variabile, successivamente si descrive la legge di controllo utilizzata e gli ingressi forniti al robot. Nella seconda parte del secondo capitolo si descrive l'obiettivo prefissato dell'architettura, ovvero il controllo di un team multi robot con uno scenario leader-(multi)follower, e si descrive l'implementazione dell'architettura su ROS mostrando quali entità sono state implementate e come comunicano tra loro.
Nel terzo capitolo si mostrano e si discutono i risultati delle simulazioni dello scenario proposto.
Nel quarto capitolo si conclude commentando i risultati ottenuti.

Capitolo 1

Concetti preliminari su ROS

1.1 ROS

ROS, acronimo di *Robotic Operating System*, è un framework flessibile per la scrittura di software dedicata ai robot. Esso si compone in un set di tool e librerie software allo scopo di semplificare la complessa creazione di applicazioni robot. La descrizione ufficiale di ROS è:

"*ROS è un meta-sistema operativo open source per il tuo robot. Fornisce i servizi tipici di un sistema operativo, inclusa l'astrazione hardware, il controllo dei dispositivi di basso livello, l'implementazione di funzionalità largamente utilizzate, la comunicazione tra processi, e la gestione dei pacchetti. Inoltre fornisce tools e librerie per ottenere scrittura, compilazione, e esecuzione del codice su varie piattaforme.*"

Gli obiettivi principali che ROS si pone possono essere riassunti in:

- **Peer-to-peer**: un sistema ROS consiste in un gran numero di processi collegati tra essi in runtime in una topologia peer-to-peer.
- **Tool-based**: ROS utilizza un gran numero di piccoli tool che forniscono diverse funzionalità tra le quali anche la compilazione e l'esecuzione del codice di un software basato su ROS.
- **Multi lingua**: ROS è "language neutral" e supporta attualmente quattro linguaggi di programmazione quali: C++, Python, Octave e LISP.
- **Snello**: Molti progetti per la robotica non garantiscono la portabilità del codice sorgente a causa del fatto che molte parti dei software sono legati a dei middleware, rendendo complicato il riutilizzo del codice al di fuori del suo contesto. ROS pone virtualmente tutta la complessità delle sue funzionalità nelle librerie, rendendo così più semplice

l'estrazione e il riutilizzo di tali funzionalità provenienti da un progetto ROS.

- **Free e Open-Source**: ROS è distribuito sotto i termini della licenza BSD, la quale permette lo sviluppo di progetti commerciali e non.

1.2 Entità e funzionalità di ROS

Le entità fondamentali dell'implementazione di ROS sono nodi, messagi e topics.

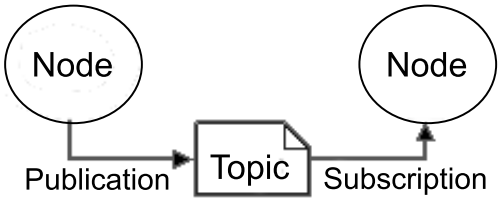


Figura 1.1: Modello di comunicazione tra entità di ROS

1.2.1 Nodi

I nodi sono i processi dove viene svolta la computazione dei file eseguibili. Solitamente un sistema ROS ha diversi nodi per il controllo di specifiche funzioni per ogni nodo. La comunicazione è di tipo peer-to-peer ed è perciò conveniente rappresentarla come un grafo, dove i processi sono i nodi del grafo e i collegamenti tra loro sono gli archi, i quali prendono il nome di *topic*. Per eseguire un nodo bisogna che esso sia dichiarato ad un nodo *master* e che il nodo master sia in esecuzione precedentemente. I nodi sono scritti con una libreria client di ROS, per esempio `roscpp` o `rospy`.

1.2.2 Topic

I messaggi sono instradati attraverso un sistema di trasporto di tipo *publish and subscribe*. Un nodo manda un messaggio *pubblicando* su di un certo topic, il quale è univocamente identificato nel grafo con un nome. Un nodo interessato ad una tipologia di informazioni si *iscrive* al topic appropriato. Da qui si evince che ci possono essere diversi nodi che si iscrivono ad un certo topic e diversi nodi che pubblicano su di un topic. L'idea generale è di disaccoppiare la produzione e l'uso delle informazioni, difatti i *publisher* e i *subscriber* non sono a conoscenza dell'esistenza gli uni degli altri.

1.2.3 Messaggi

I messaggi ROS rappresentano l'informazione che viene scambiata tra un nodo ed un altro. Vi sono una moltitudine di messaggi predefiniti ma si possono definire dei nuovi tipi di messaggi inserendo nella cartella `msg` un file con estensione `.msg`, il quale deve avere una struttura composta in due parti: tipo del campo e nome del campo.

1.2.4 Alcune funzionalità di ROS

Roslaunch ROS fornisce uno strumento utile al lancio ordinato di istanze di nodi chiamato roslaunch, attraverso l'esecuzione di un file di tipo `.launch` scritto in XML, nel quale si specifica quali nodi eseguire. ROS non permette l'esecuzione dello stesso nodo più volte contemporaneamente ma ha bisogno che ogni istanza di esso abbia un nome diverso e che sia specificato nel tag XML del launchfile del nodo in questione.
Roslaunch fornisce la possibilità di eseguire alcune operazioni come il passaggio di parametri ad un'istanza di un nodo o il ragruppamento di nodi attraverso la definizione di un "namespace", un nome di dominio comune a tutti i nodi dello stesso gruppo e molto altro.

Rosbag Rosbag è un tool di ROS utile per registrare tutti i messaggi che vengono pubblicati su di uno o più topic specificati. Esso produce dei file di tipo `.bag`, che possono essere eseguiti successivamente per riprodurre la comunicazione registrata.

Rqt Rqt è un framework software che implementa diversi tools grafici in forma di plugin. In particolare si è utilizzato il tool 'rqt_graph' che restituisce il grafo dell'architettura ROS in uso.

Capitolo 2

Sviluppo architettura ROS per il controllo multi robot

In questo capitolo si presenta l'architettura sviluppata. Si fornisce, innanzitutto, il modello matematico della dinamica per la tipologia di robot preso in esame, ovvero robot con controllore di tipo uniciclo. Successivamente si propone il modello dell'architettura proposta e si discute della sua implementazione su ROS analizzando l'obiettivo di controllo nello scenario leader-follower preso in esame e fornendo dettagli sull'implementazione effettuata.

2.1 Modello matematico di un robot uniciclo

Il modello cinematico descrive l'evoluzione dello stato del robot a tempo continuo. Il robot di tipo uniciclo preso in considerazione nell'architettura si muove in un spazio a due dimensioni.
Il modello cinematico di un robot di tipo uniciclo è descritto dal sistema di equazioni differenziali non lineari in (2.1).

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \cos\theta(t) & 0 \\ \sin\theta(t) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} \tag{2.1}$$

In questo modello $x(t)$, $y(t)$ e $\theta(t)$ rappresentano gli stati del sistema, dove $x(t)$ e $y(t)$ sono coordinate planari che definiscono la posizione del robot nel sistema di riferimento Cartesiano, mentre $\theta(t)$ è l'orientazione del robot, in termini angolari, rispetto ad un sistema di riferimento inerziale ($\widetilde{x}$, $\widetilde{y}$) centrato nel centro di massa del robot in esame.
Gli ingressi di controllo del sistema sono *v(t)* e $\omega(t)$, e rappresentano, rispettivamente, la velocità tangenziale e angolare del robot.
Nel caso dell'architettura proposta in questa tesi, i robot devono raggiungere una posizione desiderata nel piano, la quale può essere fissa o può evolvere

nel tempo. Perciò si ha la necessità di definire $\varphi(t)$ come l'angolo di direzione formato dall'asse delle ordinate del sistema inerziale ($\widetilde{x}$, $\widetilde{y}$) introdotto precedentemente e la congiungente tra la posizione corrente del robot e l'attuale posizione desiderata, come in Figura 2.1.

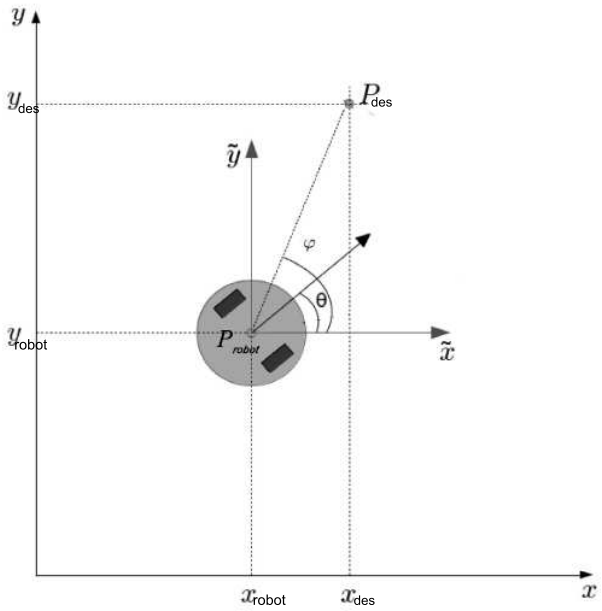


Figura 2.1: Rappresentazione grafica di robot uniciclo nello spazio

L'angolo di direzione $\varphi(t)$ è ottenuto come in (2.2).

$$\varphi(t) = \text{atan2}(y_{des}(t) - y_{robot}(t), x_{des}(t) - x_{robot}(t)) \tag{2.2}$$

Nella Figura 2.1 è rappresentato graficamente un robot uniciclo nel piano cartesiano. Il robot ha posizione $\text{P}_{robot}(x_{robot},y_{robot})$ e deve raggiungere una posizione desiderata $\text{P}_{des}(x_{des}, y_{des})$.
Nell'architettura proposta gli ingressi di controllo del sistema sono $v(t)$ e $\omega(t)$, i quali rappresentano rispettivamente la velocità tangenziale e la velocità angolare del robot.
La strategia di controllo utilizzata vuole far convergere la traiettoria del robot verso la posizione desiderata.
Il valore della velocità tangenziale che il robot riceve in ingresso dipende dalla sua distanza rispetto alla sua posizione desiderata: se il robot si trova a distanza maggiore di un certo valore riceve in ingresso una velocità tangenziale costante; se si trova nell'intorno della posizione desiderata, quindi a distanza minore di una certo valore prefissato, riceve in ingresso una velocità tangenziale inferiore rispetto a quella di partenza. La condizione sulla velocità tangenziale in ingresso è che sia non nulla per ogni istante prima

dell'effettivo arrivo alla posizione desiderata.
La velocità angolare $\omega(t)$ dipende dall'orientazione del robot e dall'angolo di direzione $\varphi(t)$ per ogni istante, ed è calcolato come in (2.3).

$$\omega(t) = K \cdot (\varphi(t) - \theta(t)) \tag{2.3}$$

dove K è una costante di guadagno.

2.2 Implementazione su ROS

L'architettura di riferimento per ogni singolo robot implementata su ROS è mostrata in Figura 2.2.

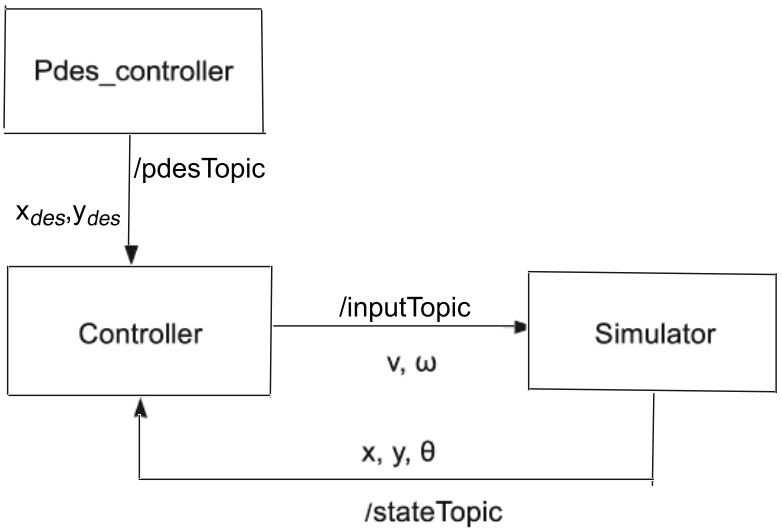


Figura 2.2: Modello architettura singolo robot

I nodi e le loro funzionalità nell'architettura sono:

- Simulator: è il nodo che simula la dinamica di un robot di tipo uniciclo. Integra le equazioni della dinamica (2.1) e comunica in retroazione lo stato corrente al controllore affinchè calcoli i nuovi ingressi di controllo.
- Controller: è il controllore, si occupa di fornire gli ingressi di controllo al simulator utilizzando la posizione desiderata fornitagli dal pdes_comunicator.
- Pdes_comunicator: si occupa di comunicare al controller la posizione desiderata. Inoltre è a conoscenza delle posizioni degli altri robot ed è l'entità che scambia informazioni con gli altri robot del team attraverso ciascun nodo comunicatore della posizione.

Ogni robot è rappresentato da simulator, controller e pdes_comunicator. I messaggi pubblicati su /stateTopic sono di tipo `Position.msg`, ovvero un custom message codificato come segue:
`float64 x`
`float64 y`
`float64 theta`.
I messaggi pubblicati su /inputTopic sono, invece, di tipo `Input.msg`:
`float64 v`
`float64 w`.
Il pdes_comunicator pubblica messaggi di tipo `PdesPosition.msg`:
`float64 x`
`float64 y`
`bool safe_flag`.
L'ingresso safe_flag è una variabile il cui scopo verrà spiegato successivamente.
All'avvio dell'applicazione il simulator pubblica il suo stato iniziale. Il controllore, sottoscritto a /stateTopic e /pdesTopic, riceve lo stato iniziale del simulatore e la posizione desiderata dal comunicatore della posizione.
Il controllore esegue le verifiche sulla distanza dal simulatore alla posizione desiderata e, in funzione di essa, fa assumere a v il valore corrispondente.
Successivamente, il controllore calcola la velocità angolare w utilizzando la (2.3).
Calcolati gli ingressi di controllo, il controller pubblica la coppia (v, ω) su /inputTopic, dove è in ascolto il simulator. Esso riceve gli ingressi e va a integrare le equazioni della dinamica descitte in (2.1), utilizzando lo stato precedente (stato iniziale se è la prima integrazione) e come tempo d'integrazione t e $t+dt$, dove all'inizio $t=0$ ma nelle successive integrazioni è proporzionale a dt.
Calcolato il nuovo stato, si aggiornano le condizioni iniziali ponendole uguali allo stato attuale e si pubblica il messaggio su /stateTopic.
Il controllore riceve il nuovo stato e la nuova posizione desiderata, effettua il controllo sulla distanza, calcola gli input di controllo e li pubblica sul topic /inputTopic.
Il simulatore riceve i segnali di controllo e calcola il nuovo stato con gli ingressi appena ricevuti.
Le operazioni si ripetono finchè non viene interrotta la comunicazione.

2.2.1 Scenario leader-follower

L'obiettivo dell'architettura proposta è il controllo di sistemi multi robot cooperativi. Nel caso preso in esame si è considerato uno scenario in cui vi è un robot *leader*, il quale è controllato affinchè la sua traiettoria converga ad una posizione desiderata preimpostata, e uno o più robot detti *follower*,

i quali inseguono la traiettoria del robot leader.
Nell'implentazione, il robot leader è determinato prima dell'avvio dall'utente.
Il robot leader riceve come posizione di destinazione dal pdes_comunicator quella prefissata dall'utente.
Per quanto riguarda il robot follower, il comunicatore della posizione desiderata si sottoscrive a /stateTopic del robot leader per acquisire la posizione di quest'ultimo e comunica al controller la posizione corrente del robot leader come posizione desiderata, affinchè calcoli i nuovi ingressi di controllo in funzione della posizione corrente del leader, allo scopo di inseguire la traiettoria di quest'ultimo.
Inoltre, il comunicatore della posizione del follower è sottoscritto a /stateTopic del proprio simulatore. Dal topic acquisisce la propria posizione e verifica la distanza di sicurezza tra robot leader e follower. Comunica al controllore il risultato dell'operazione attraverso il flag di controllo /safe_flag, affinchè abbassi la velocità tangenziale se affermativo.

Tramite rqt_graph, si ottiene il grafo di una architettura (nel caso in esame con due robot), mostrato in Figura 2.3.

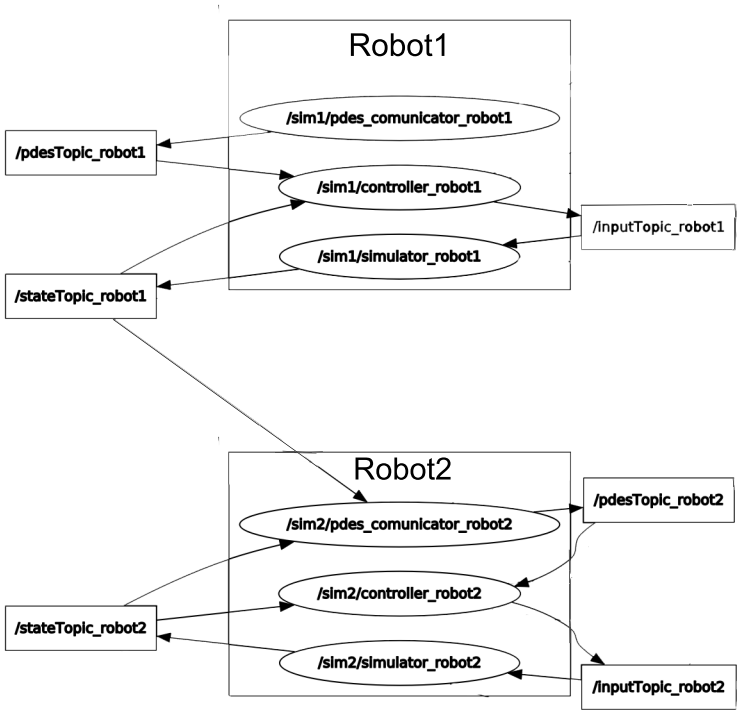


Figura 2.3: Grafo architettura con due robot

Nel grafo i rettangoli rappresentano i topic e gli ellissi i nodi. Le frecce che partono dai topic per arrivare ai nodi indicano che quei nodi sono sottoscritti a quel topic, mentre le frecce che vanno dai nodi ai topic indicano che il nodo sta pubblicando su quel topic.
I nodi dei robot sono raggruppati utilizzando dei namespace, rispettivamente '/sim1/' per il robot1 e /sim2/ per il robot2.
Nell'esempio mostrato il robot leader è robot1. Lo si può notare facilmente anche dal fatto che sul topic /stateTopic_robot1 è sottoscritto anche il comunicatore della posizione del robot. Questo perchè, come già spiegato, essendo robot follower, deve essere a conoscenza della posizione del robot leader per inseguirne la traiettoria ad ogni iterazione.

2.3 Dettagli implementativi

L'architettura presentata è stata implementata utilizzando Ubuntu 16.04, ROS Kinetic Kame e Python 3 come linguaggio di programmazione.
Per simulare il comportamento di un team di robot, si è utilizzato il launcher 'roslaunch' di ROS che permette di far eseguire più istanze dello stesso nodo. Questa funzionalità dà la possibilità di simulare il comportamento di un numero potenzialmente molto elevato di robot codificando solo tre nodi.
Per integrare la dinamica si è utilizzato la funzione odeint della libreria scipy.integrate di Python, la quale integra numericamente un sistema di equazioni differenziali detto ODE (Ordinary Differential Equations). Essa richiede in input: le derivate dello stato calcolate esplicitamente, un array di condizioni iniziali (stato precedente), una sequenza di punti per i quali risolvere l'integrale e, facoltativamente, degli argomenti da passare alla funzione per il calcolo delle derivate dello stato. Odeint restituisce una matrice nxm in cui n è il numero delle equazioni differenziali date in input mentre m è il risultato dell'integrazione per ogni punto della sequenza fornita. Perciò lo stato appena calcolato sarà dato dall'ultimo elemento di ciascuna riga.
Nell'architettura si è utilizzato rosbag che va a registrare i messaggi sui topic di posizione e degli input per fini di debug e di simulazione.
I grafici relativi alle traiettorie che i robot descrivono sono stati effettuate utilizzando MATLAB, il quale dà la possibilità di fare i plot a partire da file .csv. Questi ultimi sono stati ottenuti convertendo i file .bag di rosbag.
Il rate di comunicazione tra robot è impostato a 100Hz.

Capitolo 3

Simulazioni

In questo capitolo si mostrano le simulazioni effettuate prendendo in considerazione uno scenario leader-follower e si analizza il comportamento di ogni variabile del modello matematico proposto.

3.1 Simulazione con due robot

L'obiettivo della simulazione proposta è mostrare la traiettoria che i due robot uniciclo (un leader e un follower) tracciano partendo da stati iniziali differenti. L'obiettivo del robot leader è di far convergere il suo stato verso una posizione desiderata prefissata mentre l'obiettivo del robot follower è di far evolvere il suo stato affinchè insegua la traiettoria tracciata dal robot leader.
Il leader, che in questo caso è robot1, parte con uno stato inziale pari a $\begin{pmatrix} x_{1,0} \\ y_{1,0} \\ \theta_{1,0} \end{pmatrix} = \begin{pmatrix} 4\text{m} \\ 4\text{m} \\ 0^\circ \end{pmatrix}$, mentre il follower, rappresentato da robot2, ha come stato iniziale $\begin{pmatrix} x_{2,0} \\ y_{2,0} \\ \theta_{2,0} \end{pmatrix} = \begin{pmatrix} 2\text{m} \\ 1.5\text{m} \\ 0^\circ \end{pmatrix}$. La posizione desiderata che il leader deve raggiungere è: $\begin{pmatrix} x_{des} \\ y_{des} \end{pmatrix} = \begin{pmatrix} 3\text{m} \\ 3\text{m} \end{pmatrix}$.
Nella simulazione viene definita una distanza di arrivo alla posizione desiderata pari a 1m, come è evidenziato dalla circonferenza tratteggiata in Figura 3.1.
Quando un robot si trova ad una distanza maggiore di 1m dalla posizione desiderata, il suo controllore invia costantemente in ingresso al simulatore il segnale di velocità tangenziale con valore $v = 1\text{m/s}$. Quando il robot si trova ad una distanza inferiore il controllore invia come input di controllo al simulatore una velocità tangenziale pari a $v = 0.1\text{m/s}$.
Il comunicatore della posizione del follower ha il compito, inoltre, di verificare se il robot è troppo vicino al robot leader: se i robot si trovano ad una distanza minore di 2cm, il pdes_comunicator invia il flag di controllo safe_flag con valore affermativo allo scopo di comunicare al controllore di

abbassare la velocità tangenziale, fornendo al simulatore $v = 0.05$m/s, per evitare che i robot collidano.

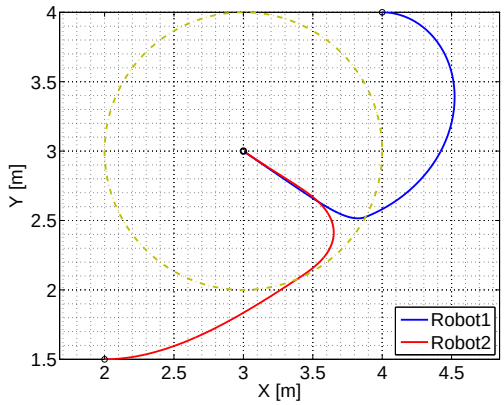


Figura 3.1: Traiettorie dei robot

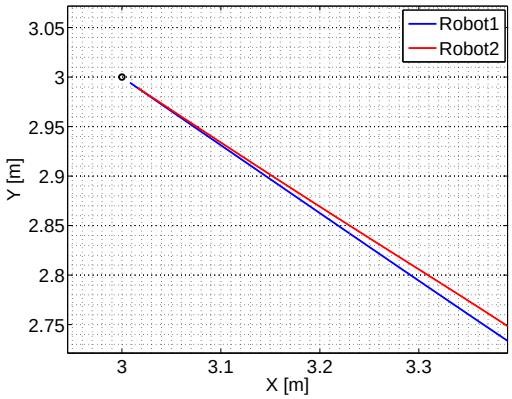


Figura 3.2: Zoom intorno a posizione desiderata

In Figura 3.1 la circonferenza tracciata di raggio 1m e centrata in (3m, 3m) rappresenta l'intorno della posizione desiderata nel quale il robot leader rallenta la sua velocità se si trova all'interno.
I robot partono con una velocità tangenziale di 1m/s e tracciano una traiettoria nel piano. Quando entrano nell'intorno della posizione desiderata rallentano la loro velocità a 0.1m/s e convergono verso la propria posizione desiderata. In questa simulazione si è imposto che, al momento dell'arrivo, il follower si fermi ad una distanza di un 1cm dalla posizione del leader. Per fare ciò, il controllore fornisce al simulatore una velocità tangenziale in ingresso $v = 0$m/s, perciò il robot si ferma.

Seguono i grafici dell'evoluzione dello stato dei robot in cui sono segnati particolari istanti di tempo. Le prime due rette verticali, una a $t = 1.68$s e una a $t = 2.12$s, indicano, rispettivamente, l'istante di tempo in cui il robot follower e il robot leader entrano nell'intorno della loro rispettiva posizione desiderata e cambiano velocità. Le successive due a $t = 13.27$s e $t = 17.91$s, invece, indicano l'istante in cui il leader prima, il follower poi, si fermano perchè arrivati a destinazione.

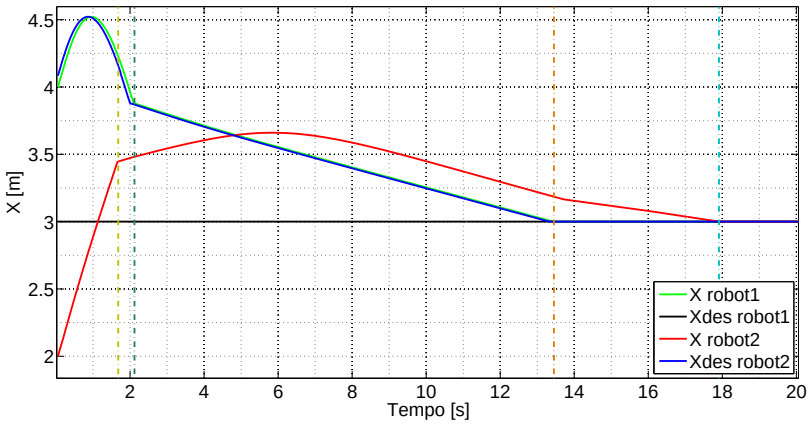


Figura 3.3: Sviluppo nel tempo di x dei robot

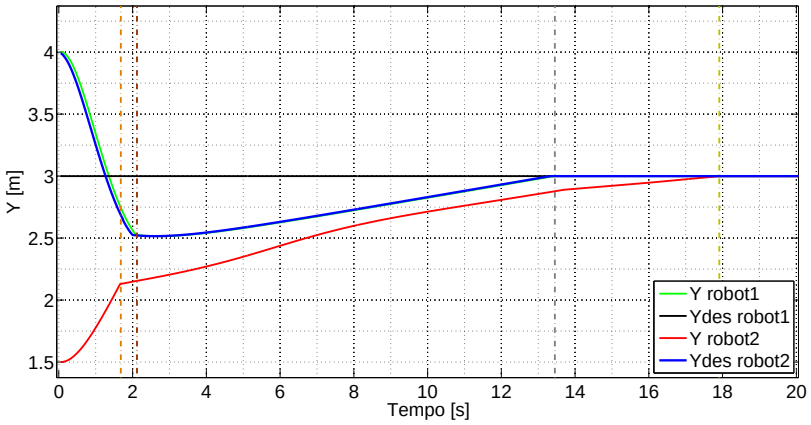


Figura 3.4: Sviluppo nel tempo di y dei robot

Nelle Figure 3.3 e 3.4 sono stati riportati l'andamento nel tempo degli stati (x, y) dei robot e le relative coordinate (x_{des}, y_{des}) desiderate.
Il robot leader ha coordinate desiderate fisse nel tempo, mentre quelle del follower sono pari alle coordinate correnti del leader. Questo è evidenziato

dalle figure prese in esame, nelle quali si nota che lo sviluppo delle coordinate desiderate del follower si sovrappone allo sviluppo della posizione del leader, a meno di un piccolo errore dovuto al ritardo di comunicazine tra nodi.

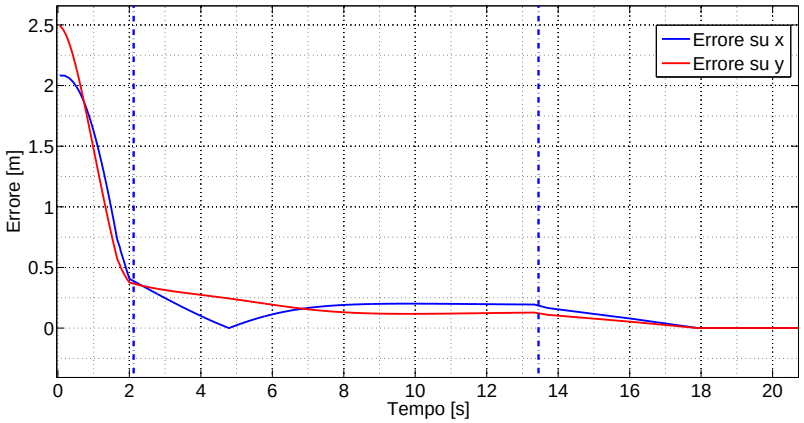


Figura 3.5: Grafico errore della posizione follower

L'errore tra la posizione del follower e la sua desiderata indica la distanza tra il robot leader e il follower, come in Figura 3.5.
L'errore è nell'ordine dei due metri all'inizio ma decresce rapidamente a meno di particolari curve nella traiettoria del robot. Quando il leader si ferma, il follower converge alla sua posizione e l'errore tende a zero.

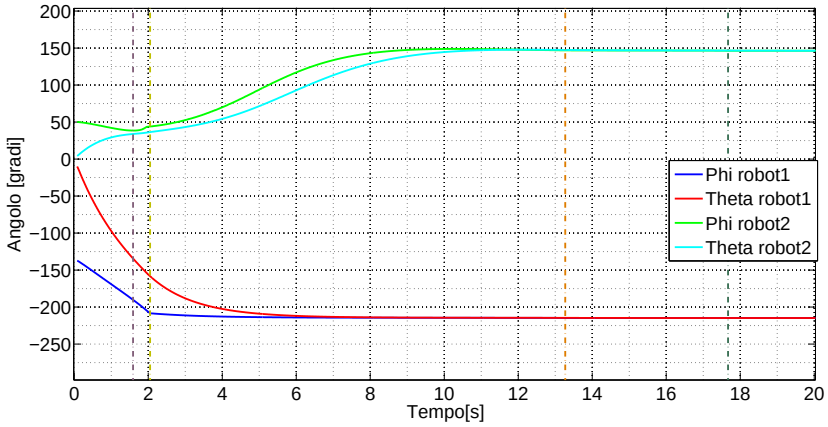


Figura 3.6: Sviluppo nel tempo di φ e θ dei robot

Nel grafico in Figura 3.6, sono riportati gli andamenti nel tempo, in gradi, dell'angolo di direzione φ e dell'angolo di orientazione θ del robot leader, rispettivamente in blu e rosso, e φ e θ del robot follower, rispettivamente in verde e giallo.
Come si osserva dai grafici, vi è una tendenza del robot ad orientarsi verso la posizione desiderata e, di conseguenza, l'orientazione θ dei due robot tende a convergere al valore dell'angolo di direzione φ. Quest'ultimo rappresenta l'angolo desiderato affinchè il robot descriva una traiettoria rettilinea per arrivare alla posizione desiderata.
Si osserva come la differenza tra i φ e θ sia ampia in partenza, ma decresce quando il robot entra nell'intorno della posizione desiderata e si annulla quando il robot si ferma.

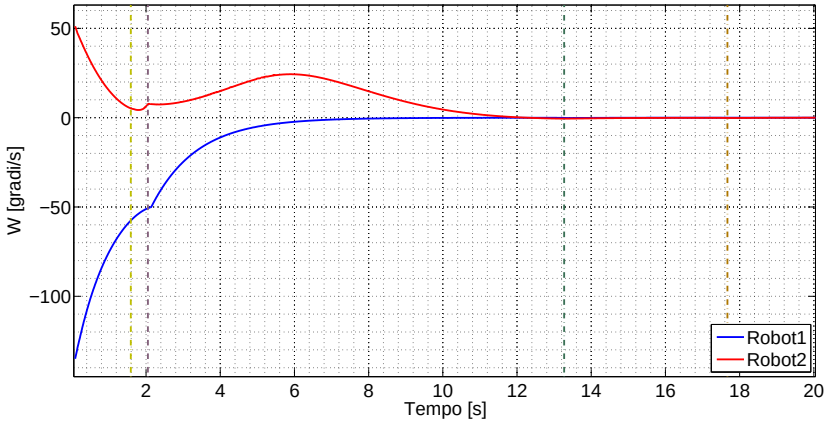


Figura 3.7: Sviluppo nel tempo di ω dei robot

La Figura 3.7 descrive l'andamento nel tempo della velocità angolare ω che, si ricorda, è definita come in (2.3).
Il robot leader è rappresentato dal robot1 (in blu in figura) mentre il robot follower è rappresentato dal robot2 (in rosso in figura).
La velocità angolare dipende dall'orientazione del robot e dall'angolo desiderato per ogni istante. In Figura 3.7 si osserva la tendenza di ω a convergere al valore di $0°/s$ a meno di particolari curve nella traiettoria. Questo è dovuto alla tendenza dei robot ad orientarsi verso la posizione desiderata come l'angolo desiderato φ e a tracciare una traiettoria rettilinea. Ovviamente, più la traiettoria è rettilinea, più velocemente la velocità angolare del robot tenderà a $0°/s$. Questo lo si può osservare, ad esempio, nel grafico del robot leader: quando entra nell'intorno della posizione desiderata traccia una traiettoria rettilinea, di conseguenza la sua velocità angolare tende a $0°/s$ e rimane nulla quando il robot si ferma.

Conclusioni

La tesi ha presentato un'architettura per il controllo di un team di robot mobili cooperativi, sviluppata utilizzando l'innovativo framework ROS, utile alla creazione di applicazioni software per la robotica.
L'obiettivo di controllo dell'architettura presentata è far convergere lo stato del robot verso una posizione desiderata. Ogni robot è modellato da un simulatore che simula la dinamica del robot, un controllore che fornisce e un comunicatore della posizione desiderata a cui lo stato del robot deve convergere.
Nella prima parte sono stati introdotti i concetti fondamentali di ROS e si è discusso sulla modellazione matematica dei robot e sulla modellazione implementativa dell'architettura proposta.
Si è preso in esame uno scenario in cui vi è un rapporto di tipo leader-follower tra i robot. L'architettura proposta vuole controllare l'evoluzione dello stato dei robot nel tempo, allo scopo di far convergere la traiettoria del robot leader verso una posizione desiderata fissa, e di far evolvere lo stato dei robot follower in modo che insegui la traiettoria del leader, avendo come posizione desiderata quella corrente del robot leader.
Nella seconda parte sono stati mostrati i risultati delle simulazioni effettuate prendendo in considerazione due robot. Si è osservato l'effettiva convergenza della traiettoria del leader verso la posizione desiderata e la tendenza del follower ad inseguire lo stato del leader nel tempo. Si è mostrato, inoltre, la tendenza dei robot ad orientarsi verso la posizione desiderata.

Bibliografia

[1] Gregor Klančar, Drago Matko, Sašo Blažič, *A control strategy for platoons of differential drive wheeled mobile robot*, University of Ljubljana, Slovenia, 2010, `http://motion.me.ucsb.edu/book-lns`

[2] Lorenzo Sabattini, *Nonlinear control strategies for cooperative control of multi-robot systems*, University of Bologna, 2012.

[3] Lynne Parker, *Current State of the Art in Distributed Autonomous Mobile Robotics*, University of Tennessee, 2002, `https://www.researchgate.net/publication/2475886`

[4] Victor Adolfsson, *The State of the Art in Distributed Mobile Robotics*, Blekinge Institute of Technology, Ronneby, Sweden, 2001.

[5] Aaron Martinez, Enrique Fernández *Learning ROS for Robotics Programming*, 2013, Packt Publishing