

# Navigazione autonoma indoor mediante campi di forza

**Facoltà di Ingegneria dell'informazione, informatica e statistica**  
**Dipartimento di Ingegneria Informatica, Automatica e Gestionale**  
**Corso di laurea in Ingegneria Informatica e Automatica**

**Martina Evangelisti**  
**Matricola 1796480**

Relatore  
Prof. Giorgio Grisetti

Correlatore  
Dott. Tiziano Guadagnino

# Sommario

INTRODUZIONE.....	3
CAPITOLO 1.....	5
CONCETTI BASE.....	5
1.1-Navigazione autonoma: tassonomia.....	5
1.2-APF: Artificial Potential Fields .....	7
1.3-Importanza della rappresentazione dell'ambiente sulla mappa.....	8
CAPITOLO 2.....	10
TECNOLOGIE UTILIZZATE .....	10
2.1-ROS: Robot Operating System .....	10
2.2-ActionLib: move_base package .....	11
2.3-Gmapping .....	15
2.4-Open CV .....	18
CAPITOLO 3.....	20
ANALISI STRUTTURALE DEL PROGETTO .....	20
3.1-Roadmap con rqt graph .....	20
3.2-Componenti.....	21
CAPITOLO 4.....	23
REALIZZAZIONE DEL SISTEMA: ELABORAZIONE DELLA MAPPA.....	23
4.1-Costruzione della Distance Map .....	24
4.2-Applicazione dei Campi di Forza Repulsivi .....	28
4.3-Cenni sul Path-Planning e setting delle velocità al robot .....	30
CAPITOLO 5.....	32
Sperimentazione.....	32
5.1-Esperimenti con simulatore.....	32
5.2-Esperimenti in ambiente reale .....	39
CAPITOLO 6.....	44
CONCLUSIONI.....	44
BIBLIOGRAFIA.....	46



## Introduzione

I robot autonomi si definiscono come macchine intelligenti in grado di svolgere compiti senza il controllo di un operatore esterno. In una prospettiva futura nella quale i robot possano operare in svariati campi applicativi come agenti autonomi, è necessario che essi siano autocoscienti e autosufficienti. Allo scopo di ampliare il loro campo di impiego e dotarli di un elevato grado di autonomia, un requisito fondamentale è la capacità di navigare nello spazio acquisendo una percezione di esso. In particolare, un robot mobile è una struttura meccanica in grado di muoversi autonomamente.

In robotica questo concetto è conosciuto come “Problema della navigazione autonoma” e può essere descritto come il raggiungimento di uno specifico punto nello spazio, data una posizione iniziale, evitando gli ostacoli.

È in questo scenario che si colloca il progetto di questa tesi: lo sviluppo di un nodo ROS (Robot Operating System) il quale obiettivo principale è la navigazione autonoma del robot in ambienti indoor realizzata tramite l'applicazione di campi di forza attrattivi e repulsivi.

Nell'ottica di una metodologia “Sense Plan Act” il progetto si pone inizialmente lo scopo di rappresentare l'ambiente circostante tramite una mappa e di mantenere informazioni riguardanti gli ostacoli all'interno di essa associandogli le rispettive forze repulsive e, successivamente, di configurare la forza attrattiva che permetta al robot di trovare un percorso verso il “goal”, ovvero l'obiettivo richiesto.

Per raggiungere un task è necessario che il robot comprenda l'ambiente tramite misurazioni di sensori e prenda coscienza di esso catturando le informazioni al livello di astrazione necessario.

È fondamentale apprendere la conformazione dell'ambiente, dove esso si trova e dove si trova il robot rispetto ad esso per poi poter navigare e trovare un percorso verso il target.

Ho sviluppato questo progetto con la collega Silvia Bianchini che ha curato la parte riguardante il “path planning” e, seppur l’argomentazione è suddivisa in due parti, si costituisce come un unico lavoro in cui ogni componente è necessaria allo scopo di raggiungere il risultato finale.

Il presente lavoro di tesi è stato possibile grazie alle conoscenze apprese durante il corso di *Laboratorio di intelligenza artificiale e grafica interattiva*, tenuto dai professori Giorgio Grisetti e Daniele Nardi.

I capitoli che costituiscono la tesi sono organizzati come segue. Nel primo capitolo verranno esposti i concetti generali della navigazione autonoma e come essa viene resa possibile tramite dei campi di forza.

Nel capitolo successivo si analizzeranno le tecnologie che sono alla base del progetto stesso per poi introdurre nel terzo capitolo la struttura del progetto ed esporre nel quarto una metodologia di navigazione che abbiamo sviluppato. In questa tesi si analizzerà nello specifico la creazione della distance map e l’*obstacle avoidance* tramite campi repulsivi.

Il quinto capitolo presenta la sperimentazione realizzata sia tramite un simulatore che in un ambiente reale, per concludere con i possibili sviluppi futuri del sistema che si possono implementare per rendere la navigazione autonoma il più robusta e accurata possibile.

# Capitolo 1

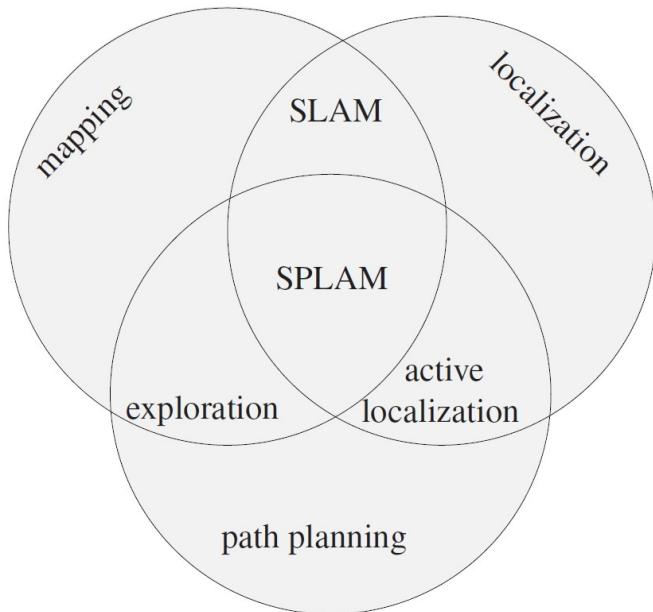
## Concetti Base

### 1.1-Navigazione autonoma: tassonomia

Una visione d'insieme del problema si ottiene analizzando gli elementi base della navigazione autonoma e le domande alle quali il sistema deve rispondere:

“Dove mi trovo?”, “Cosa c’è intorno a me?”, “Come raggiungere un determinato punto?”.

Le risposte a queste domande si configurano come il concetto di: localizzazione, costruzione della mappa e pianificazione del moto.



- *Localization*: stima la posizione corrente del robot tramite i dati sensoriali, con mappa nota.
- *Mapping*: processo di realizzazione della mappa, conoscendo la posizione del robot.
- *Path Planning*: pianifica la traiettoria per raggiungere un obiettivo conoscendo la localizzazione e la mappa.
- *Slam*: localizzazione e mapping simultanei.

**Figura 1.1:** schema riassuntivo del problema della navigazione

Per poter navigare correttamente sfruttando questi moduli è necessario che la mappa rifletta in dettaglio la totalità dell’ambiente circostante, che l’ambiente sia statico e che non ci siano errori nella stima delle posizioni.

Ci occupiamo della navigazione del robot in ambienti “human-like” che possiedono quindi tutte le caratteristiche tipiche delle stanze di un normale edificio.

Bisogna quindi considerare che in ambienti reali le stime subiscono disturbi a causa del rumore e che questi stessi ambienti non sono statici, ma possiedono delle componenti variabili.

Questa variabilità è introdotta dalla presenza degli ostacoli che il robot può trovarsi nella traiettoria verso il target. Questi possono essere di due tipologie:

- Ostacoli fissi: fanno parte della struttura dell'ambiente circostante e devono essere registrati nella mappa, ad esempio muri o armadi.
- Ostacoli mobili: la loro presenza non è prevedibile, ad esempio persone in movimento oppure porte che possono essere aperte o chiuse in maniera indistinta.

È necessario un modulo che si occupi della “obstacle detection” e della “obstacle avoidance”. Il primo si occupa di rilevare la presenza di un ostacolo, mentre il secondo di evitarlo.

La obstacle detection si realizza mediante opportuni sensori di distanza che possono essere ottici o acustici, oppure tramite telecamere e algoritmi di computer vision.

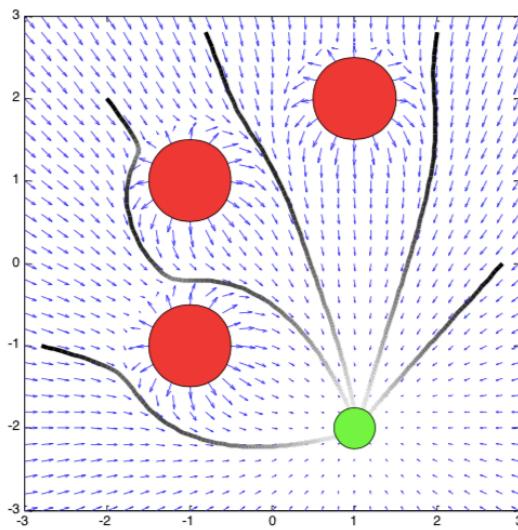
I meccanismi di avoidance sono di varie tipologie e dipendono spesso dalle caratteristiche stesse dell'ostacolo. Se consideriamo, ad esempio, la presenza di una porta, questa, pur essendo considerata un ostacolo dinamico, presenta solo due configurazioni possibili e, se queste sono durature nel tempo, è possibile modificare la mappa per considerare la porta come un ostacolo fisso (nel caso sia chiusa) o non considerarla (nel caso sia aperta).

Se invece l'ostacolo non presenta le caratteristiche precedenti e non se ne conosce la geometria è possibile usare un “bug algorithm” che permette di circumnavigare l'ostacolo fino a superarlo tramite l'utilizzo di un sensore a 360 gradi.

Il nostro progetto prevede la mappatura tramite sensori dell'ambiente circostante solo in una prima fase, ovvero quella della strutturazione della mappa. A tale proposito si presuppone che nell'ambiente ci siano solo ostacoli fissi e la navigazione è resa possibile grazie ad opportuni campi di forza artificiali.

## 1.2-APF: Artificial Potential Fields

L'approccio dei campi potenziali per la navigazione prevede una modellazione dello spazio di navigazione, quello rappresentato dalla mappa, come invaso da campi di forze. In particolare, da un campo di forze repulsive che allontanano il robot dagli ostacoli e da un campo di forze attrattive che lo trascinano verso il target.



*Figura 1.2: Rappresentazione grafica in 2D dei campi di forza*

Definiamo il campo attrattivo e il suo gradiente tramite le seguenti funzioni:

$$U_{\text{att}} = \xi d^2 / 2$$

$$\nabla U_{\text{att}} = \xi (\mathbf{q} - \mathbf{q}_a)$$

dove  $\mathbf{q}$  è la posizione del robot e  $\mathbf{q}_a$  è il centro del campo attrattivo, quindi del target, e  $d$  è la distanza tra il punto  $\mathbf{q}$  e  $\mathbf{q}_a$   $d=|\mathbf{q}-\mathbf{q}_a|$  e  $\xi$  una costante.

Il campo repulsivo, con il rispettivo gradiente, è invece definito come segue:

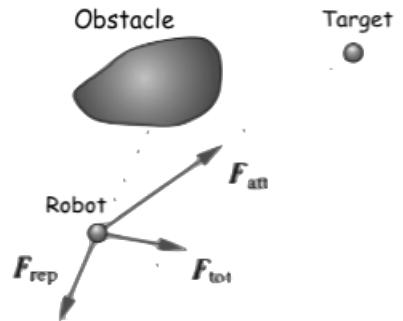
$$U_{\text{rep}} = \begin{cases} \frac{1}{2} \eta \left( \frac{1}{d} - \frac{1}{d_o} \right)^2 & d \leq d_o \\ 0 & d > d_o \end{cases}$$

$$\nabla U_{\text{rep}} = \begin{cases} \eta \left( \frac{1}{d} - \frac{1}{d_o} \right) \frac{(\mathbf{q} - \mathbf{q}_o)}{d^3} & d \leq d_o \\ 0 & d > d_o \end{cases}$$

dove  $d=|\mathbf{q} - \mathbf{q}_o|$  e  $\mathbf{q}_o$  è la posizione dell'ostacolo considerato, che scegliamo essere quello più vicino ed  $\eta$  è una costante.

Le forze attrattive e repulsive che il robot subisce sono definite come  $F = -\nabla U^1$ .

Come si può osservare dalle formule dei campi, la forza repulsiva sarà inversamente proporzionale alla distanza del robot dall'ostacolo e risulterà quindi essere inferiore a distanze maggiori da esso.



*Figura 1.3: somma vettoriale delle forze*

Al contrario la forza attrattiva sarà maggiore all'avvicinarsi del target prescelto.

Il robot sarà quindi soggetto alla sommatoria delle due tipologie di forze: è la composizione dei vettori di queste che permette il movimento verso l'obiettivo evitando gli ostacoli e che stabilisce quindi il *"motion planning"* che tradurrà il task iniziale in dei comandi di velocità e di rotazione ai giunti del robot.

Le componenti attrattiva e repulsiva possono essere pesate con dei fattori di guadagno che permettono di stabilire il coefficiente di influenza dei rispettivi campi ed adattare il comportamento del robot alle specifiche esigenze e situazioni.

### 1.3-Importanza della rappresentazione dell'ambiente sulla mappa

La mappa è la rappresentazione dell'ambiente all'interno del quale il robot sta operando e deve contenere tutte le informazioni necessarie affinché il robot possa conseguire il task al quale siamo interessati. La mappa identifica quindi la conoscenza che si possiede del mondo.

Questa conoscenza dell'ambiente è una componente necessaria ed imprescindibile per il robot al fine di compiere appieno le proprie funzioni, ed è per questo motivo che lo studio

---

<sup>1</sup> Gradiente negativo del campo potenziale preso in considerazione:  $U_{att}$  o  $U_{rep}$

delle tecnologie di mapping e delle metodologie di rappresentazione della mappa è un campo centrale della ricerca e dello sviluppo nell'ambito della robotica.

In questo progetto abbiamo scelto di sfruttare una occupancy grid che rappresenta l'ambiente tramite una griglia bidimensionale dove ogni cella possiede un valore che rappresenta il suo stato, in particolare se è libera oppure se essa identifica un ostacolo. L'ambiente risulta essere quindi discretizzato e memorizzato su una matrice.

La occupancy map è rappresentata tramite un'immagine che mostra la planimetria dell'ambiente e tramite un file di configurazione (con estensione yaml) che contiene i metadati che rappresentano le informazioni della mappa stessa (risoluzione, origine ecc).

Questa mappa è stata realizzata sfruttando il pacchetto Gmapping del sistema ROS che risolve il problema di SLAM [*Figura 1.1*].

Data questa configurazione è stato possibile implementare un algoritmo che permette la costruzione di una distance map, ovvero una mappa dove ogni cella memorizza la sua distanza dall'ostacolo più vicino.

L'implementazione e lo sviluppo di quest'ultima sono quindi il fulcro centrale del progetto in quanto essa costituisce la base fondamentale dalla quale è possibile applicare i campi di forza precedentemente descritti e quindi permettere la navigazione autonoma del robot.

# Capitolo 2

## Tecnologie utilizzate

### 2.1-ROS: Robot Operating System

L'intero progetto è stato realizzato sfruttando il framework ROS, una collezione di librerie e tool sviluppati per costruire sistemi robotici complessi.

Si tratta di un sistema meta-operativo open source che gestisce autonomamente l'astrazione dell'hardware e implementa le funzionalità base della comunicazione inter processo come la creazione e l'infrastruttura di scambio di messaggi e gestione di pacchetti. Definisce inoltre strumenti per la costruzione e la gestione di file system.

Il codice del progetto è stato sviluppato come package catkin all'interno di un catkin workspace, ovvero un folder in cui installare, modificare e costruire i rispettivi package. Questi ultimi sono la struttura principale per l'organizzazione del software ROS e contengono processi, librerie, file di configurazione e tutti gli eseguibili utilizzati dal sistema. Gli eseguibili del progetto sono implementati come nodi ROS.

Un nodo in ROS è un processo in grado di comunicare con gli altri nodi. L'obiettivo è la scomposizione di attività complesse in funzionalità specifiche gestite ognuna da diversi nodi intercomunicanti, rendendo così la programmazione modulare ed il codice più robusto.

La comunicazione può esprimersi sotto forma di pubblicazione e sottoscrizione ad un topic o tramite l'utilizzo o la messa a disposizione di un servizio. I topic rappresentano stream di messaggi della stessa tipologia e, tramite il paradigma di publish/subscribe, la produzione ed il consumo del messaggio sono disaccoppiati rendendo il meccanismo asincrono. I servizi implementano invece un pattern di richiesta/risposta inerentemente sincrono definito da una coppia di messaggi che realizzano un meccanismo di RPC (remote procedure call) tra nodi.

Alla base della comunicazione tra nodi vi è dunque lo scambio di messaggi. Ogni diversa tipologia di messaggio è descritta da un file.msg memorizzato nella sotto-directory msg all'interno del pacchetto ROS nel quale esso è implementato.

```
fieldtype1 fieldname1
fieldtype2 fieldname2
fieldtype3 fieldname3
```

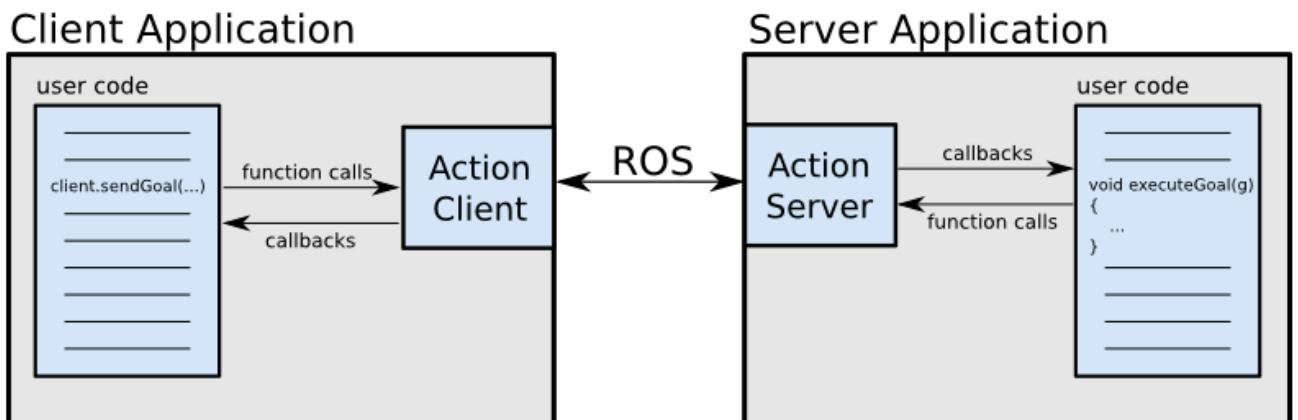
*Figura 2.1: Struttura tipica di un messaggio*

Il broker di gestione di questa infrastruttura è il Roscore, componente necessaria per stabilire la connessione tra i nodi ed è un pre-requisito necessario per ogni programma basato su un sistema ROS.

## 2.2-ActionLib: move\_base package

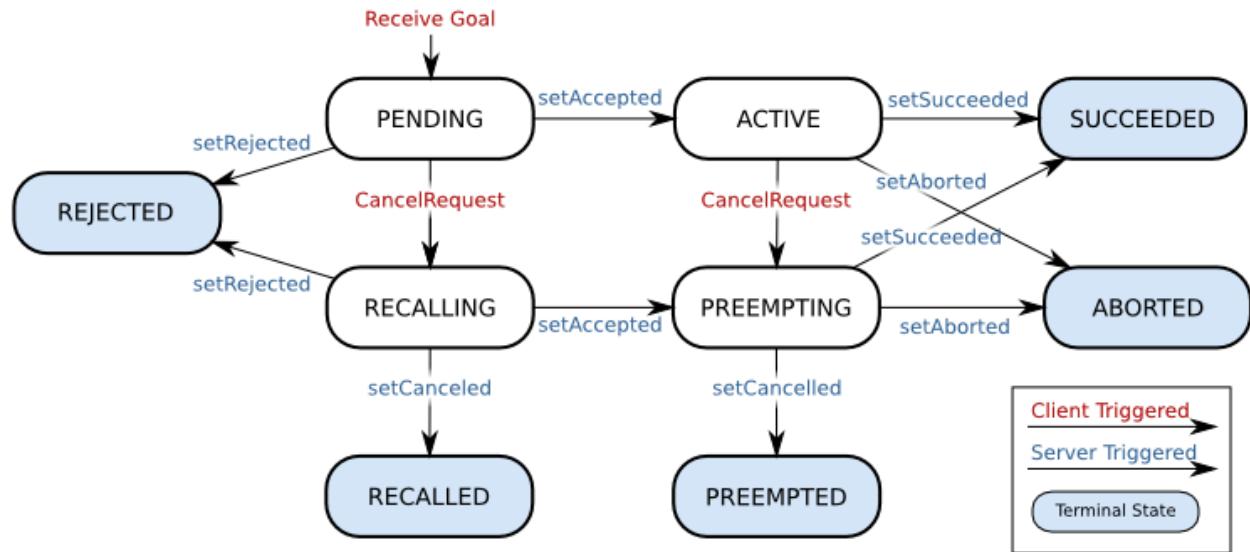
ActionLib è un pacchetto che fornisce strumenti per la realizzazione di task di lunga durata, definiti come *actions*, e permette di monitorare, attraverso dei feedback, i progressi di questi task durante la loro esecuzione. Offre, inoltre, la possibilità di interrompere tale esecuzione tramite la cancellazione dell'obiettivo richiesto.

ActionLib mette a disposizione un'interfaccia standardizzata per una comunicazione Client/Server allo scopo di conseguire un obiettivo tramite un *ROS Action Protocol* che permette all'ActionClient e all'ActionServer di comunicare utilizzando messaggi ROS. Le opzioni di richiesta e di esecuzione di un goal sono messe a disposizione tramite API.



*Figura 2.2: Interfaccia di comunicazione tra Action Client e Action Server*

La comunicazione viene inizializzata dal Client tramite la richiesta di un Goal. Alla ricezione di questo, il Server costruisce al suo interno una macchina di stati per tenere traccia dei vari stadi della computazione.



*Figura 2.3: State Machine dell' Action Server*

La Figura 2.3 mostra il diagramma degli stati e delle transizioni che il Server ActionLib mantiene per ogni goal che gli viene richiesto.

Gli stati del server si possono suddividere in:

Stati Intermedi:

- Pending: il goal richiesto non è stato ancora elaborato.
- Active: il goal è in fase di processamento.
- Preempting: mentre il goal era in elaborazione il client ne ha richiesto la cancellazione.
- Recalling: il client ha richiesto la cancellazione del goal prima che il server ne iniziasse l'elaborazione.

Stati Terminali:

- Succeeded: l'obiettivo è stato raggiunto correttamente.
- Rejected: il server ha rifiutato il goal prima di elaborarlo e senza nessuna richiesta di annullamento da parte del client.

- Aborted: il goal è stato annullato da parte del server durante l'elaborazione.
- Recalled: stato terminale raggiunto a partire dal Recalling quando il server accetta la cancellazione richiesta dal client.
- Preempted: stato terminale raggiunto a partire dal Preempting quando il server accetta la cancellazione richiesta dal client.

Le varie tipologie di transizioni rappresentano il set di possibili comandi eseguibili dal Server:

- *setAccepted*: il server accetta il goal dopo averlo ispezionato e decide di iniziare l'elaborazione.
- *setRejected*: il server rifiuta il goal dopo averlo ispezionato perché la richiesta non è valida.
- *setSucceeded*: notifica l'avvenuto raggiungimento del goal con successo.
- *setAborted*: notifica che il goal è stato interrotto a causa di errori durante la sua elaborazione.
- *setCanceled*: notifica che c'è stata una richiesta di cancellazione per quel goal che non è più in elaborazione.

Una transizione può avvenire anche in maniera asincrona a causa di un trigger da parte del client con “*CancelRequest*” attraverso il quale richiede l'interruzione del processamento di quel goal.

La macchina degli stati del Server è la macchina primaria, ad essa è associata una macchina secondaria che rappresenta gli stati del Client.

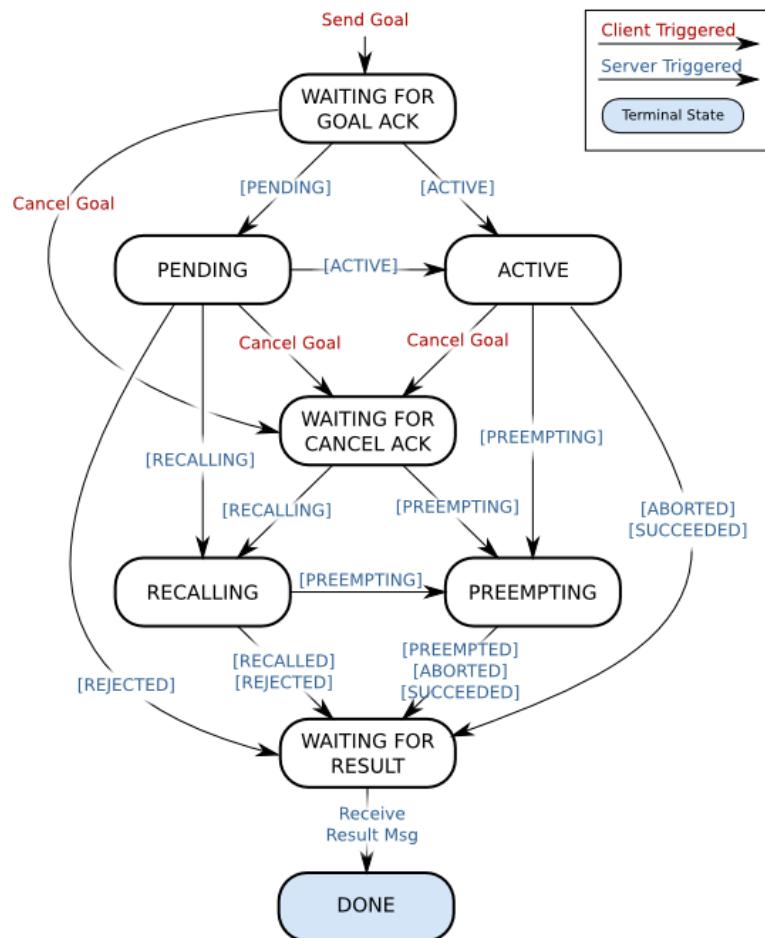
Questa seconda macchina cerca di riprodurre l'attuale stato di elaborazione del goal nel lato Server, di conseguenza la maggioranza delle transizioni sono attivate da segnalazioni di Stato che provengono dal Server e non da azioni scelte lato Client.

Transizioni indotte dal Server:

- Reported[state]: report ricevuto dal server sullo stato di processamento.

- Receive Result Message: il server segnala il termine della computazione riportandone l'esito.

Le uniche transizioni indotte dal Client sono la “*Send Goal*” e la “*Cancel Goal*” che rispettivamente istanziano o cancellano un Goal.



*Figura 2.4: State Machine dell' Action Client*

Le *Action Specification* definiscono i messaggi tramite i quali Client e Server comunicano tra loro in un file .action, specificando le tipologie di messaggio: Goal, Feedback e Result.

I messaggi di Goal definiscono la nozione di attività che si vuole eseguire e quindi l’obiettivo che si vuole raggiungere. Nel caso della navigazione di una base mobile questo conterrà informazioni riguardo la posizione che si vuole far raggiungere al robot.

I messaggi di Feedback e Result contengono rispettivamente informazioni riguardo lo stato di avanzamento dell’elaborazione del Goal e il risultato ottenuto al completamento

dell’obiettivo. Si osservi che i messaggi di Feedback sono numerosi mentre il Result viene inviato una ed una sola volta.

Nel progetto sviluppato in questa tesi le *action* da eseguire sono lo spostamento del robot da una posizione di partenza ad una posizione di arrivo: abbiamo quindi sfruttato messaggi di tipo move\_base.

Il move\_base package si occupa delle azioni che riguardano la navigazione di una base mobile nel mondo, in particolare il raggiungimento di un goal che rappresenta una posizione specifica nello spazio.

```
geometry_msgs/PoseStamped target_pose
---
---
geometry_msgs/PoseStamped base_position
```

*Figura 2.5: MoveBase.action*

Le tipologie di messaggi scambiati tra nodi di tipo move\_base sono auto generate dal MoveBase.action. Il *target\_pose* identifica la posa obiettivo che il robot deve raggiungere, la *base\_position*, invece, rappresenta la posizione corrente della base nel mondo, inviata come feedback, secondo la rappresentazione del tf.

Il tf è un pacchetto che, sfruttando un albero delle trasformate, consente di tenere traccia nel tempo di più frame di coordinate permettendo di ottenere le posizioni relative tra due oggetti nello spazio.

## 2.3-Gmapping

Gmapping è un filtro particolare Rao-Blackwellized altamente efficiente che fornisce una risoluzione allo SLAM, ovvero il problema della mappatura e della localizzazione simultanea, sfruttando un laser.

La mappa creata è una occupancy grid map di due dimensioni costruita sulla base dei dati raccolti dal laser posto a bordo del robot mobile. In ambiente indoor il risultato è strutturato come se fosse una planimetria dell’edificio ispezionato.

La rappresentazione della mappa come occupancy grid richiede una grande quantità di memoria, è però in grado di fornire una descrizione dettagliata dell'ambiente e può contenere informazioni utili allo svolgimento di un particolare task.

Questo approccio prende in input dati grezzi di un laser e dell'odometria ed è ottimizzato per scanner laser a lungo raggio come SICK LMS<sup>2</sup>.

Si sfrutta un filtro in cui ogni particella trasporta una copia della mappa e si utilizzano tecniche adattative che portano a ridurre drasticamente il numero di particelle nel filtro. Questo si rende necessario in quanto ogni particella ha bisogno di uno spazio di memoria che possa contenere l'intera mappa e, se il numero di particelle è troppo elevato, questa può eccedere il footprint di memoria del computer. Ciò è reso possibile calcolando una distribuzione di probabilità che tenga conto del movimento del robot e dell'osservazione più recente, allo scopo di ridurre l'incertezza sulla posa del robot nella fase di predizione del filtro. Mediante la ripetizione di un campionamento selettivo si può inoltre ridurre il problema dell'esaurimento delle particelle.

Il problema della creazione della mappa, compito fondamentale dei robot mobili, viene spesso ricondotto al problema di SLAM, la complessità del quale deriva dal fatto che per acquisire la mappa il robot ha bisogno di una stima abbastanza accurata della sua posizione, ma, allo stesso tempo, per localizzarsi ha bisogno di una mappa coerente.

Il filtro particellare Rao-Blackwellized è una soluzione in uno spazio ad alta dimensione, efficace per risolvere il problema simultaneo di localizzazione e mappatura. Il limite principale di questo approccio risiede nella sua complessità, ovvero nel numero di particelle necessarie per la costruzione della mappa.

---

<sup>2</sup> Laser ad alta precisione, ampio raggio visivo (~270°), rilevamento rapido ed affidabile di oggetti in qualsiasi condizione ambientale, sicurezza approvata per rilevamento delle collisioni.

Per aumentare le prestazioni del filtro applicato a SLAM con Grid Map si può sfruttare una tecnica di ricampionamento adattivo che mantenga una ragionevole varietà di particelle, volta a ridurre il rischio di “particle depletion” ovvero di eliminare, nel successivo campionamento, particelle contenenti informazioni corrette e coerenti.

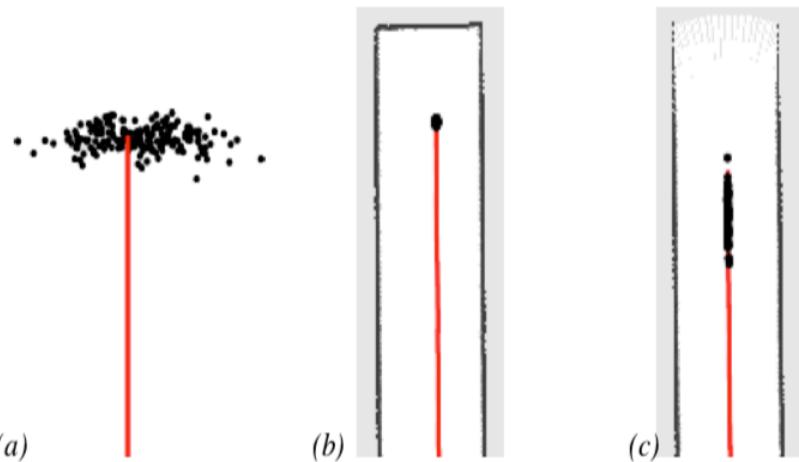
Un'altra modalità è l'utilizzo di una distribuzione di probabilità che consideri l'accuratezza dei sensori del robot permettendo così di selezionare le particelle nella maniera più precisa ed accurata possibile.

La “*proposal distribution*” è calcolata valutando la probabilità di una posa dipendente dalla distribuzione delle particelle ottenuta da una procedura di scansione precedente e confrontandola con i rilevamenti dei sensori a bordo della base mobile. Durante la generazione della particella viene quindi presa in considerazione la lettura effettuata precedentemente, consentendo quindi di stimare l'evoluzione del sistema basandosi su un modello che contiene più informazioni rispetto ad un modello che si basa solamente sull'ultima lettura dell'odomетria.

Questo garantisce maggiore accuratezza nella mappa in quanto la scansione laser viene incorporata nella mappa di ogni particella dopo aver calcolato l'effetto sulla posa del robot, generando quindi un errore di stima nel tempo significativamente più basso.

In un RBPF, Rao-Blackwellized particle filter, ogni particella rappresenta una possibile posa del robot e una mappa. Quest'algoritmo prevede una proposal distribution dinamica, calcolata “*on the fly*” invece di una distribuzione fissa, permettendo di sfruttare direttamente tutte le informazioni ottenute dai sensori durante la generazione delle particelle.

*Figura 2.6: Proposal distribution osservata tipicamente durante la mappatura*



- a) Open space: distribuzione secondo il modello dell'odometria
- b) Corridoio chiuso: incertezza ridotta in tutte le dimensioni
- c) Corridoio aperto: particelle distribuite lungo il corridoio

I vantaggi fondamentali di questo approccio si riassumono nell'accuratezza della proposal distribution che permette di utilizzare il numero di particelle come un robusto indicatore per decidere se effettuare o meno un successivo ricampionamento.

## 2.4-Open CV

Open Source Computer Vision Library è una libreria software per visione artificiale in tempo reale e machine learning.

La struttura di OpenCV è modulare e le funzionalità di base sono messe a disposizione dal modulo *core*. Questo definisce le strutture essenziali per tutti gli altri moduli come, ad esempio, l'array multi-dimensionale Mat.

L'elaborazione delle immagini è delegata a *imgproc*, modulo che include filtri lineari e non lineari insieme a trasformazioni geometriche come distorsione, ridimensionamento o rimappatura dello spazio basata su tabelle. Per le analisi di video esiste un modulo specifico che contiene funzionalità di stima del movimento e tracciamento di oggetti.

OpenCV mette inoltre a disposizione moduli specifici per la object detection con istanze di classi già predefinite.

Il modulo High Level GUI and media, *highgui*, fornisce un'interfaccia utente che permette di creare e manipolare finestre per mostrare a video immagini. Garantisce, inoltre, la possibilità di aggiungere a queste finestre delle trackbar per spostarsi al loro interno tramite mouse o tastiera. Queste immagini, o video, si possono prelevare da disco o memoria oppure leggere da camera.

Nel progetto sviluppato in questa tesi si sfruttano i moduli *imgproc* ed *highgui* per costruire e, successivamente, mostrare un'immagine rappresentante la distance Map in sfumature di grigi sfruttando un canale unico ed 8 bit per pixel.

## Capitolo 3

### Analisi strutturale del progetto

#### 3.1-Roadmap con rqt graph

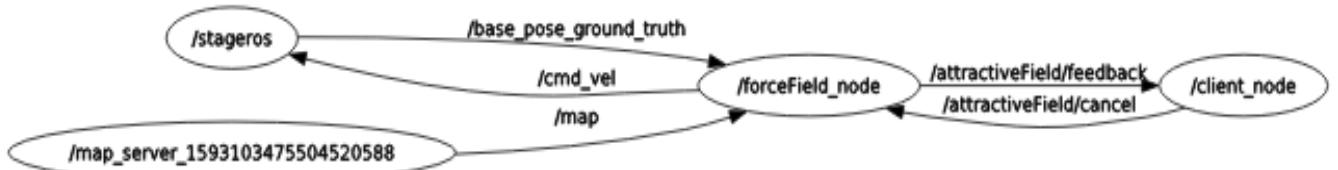
La raccolta dei dati dei sensori avviene tramite la navigazione in un ambiente inizialmente sconosciuto. I dati raccolti vengono memorizzati in una bag<sup>3</sup>, che verrà successivamente riprodotta come input di gmapping allo scopo di risolvere il problema dello slam ed ottenere una occupancy grid map dell'ambiente rappresentato dalla bag.



*Figura 3.1: Rqt\_graph nella fase di creazione della mappa*

La raccolta dei dati e la rispettiva registrazione verranno spiegati in seguito, nel capitolo relativo alla sperimentazione.

Il nodo principale del progetto è il forceField\_node. La Figura 3.2 mostra l'rqt graph, grafo di computazione di ROS, che rappresenta l'interazione tra questo e gli altri nodi presenti nel progetto. Ogni arco del grafo rappresenta il topic sul quale i due nodi si scambiano i messaggi.



*Figura 3.2: Rqt\_graph durante la navigazione*

<sup>3</sup> File format di ROS per la memorizzazione di messaggi ROS. Permette di registrare e successivamente riprodurre tutti i messaggi pubblicati su un determinato topic. Nel progetto è stato utilizzato il package rosbag.

Il forceField\_node si sottoscrive al topic /map sul quale il nodo map\_server mette a disposizione la mappa precedentemente ottenuta. All'acquisizione della mappa, il nodo costruisce la distance map e applica i campi repulsivi agli ostacoli registrati in essa.

Con una comunicazione che avviene tramite lo scambio di messaggi di tipo move\_base il client\_node comunica al forceField\_node il goal che vuole che il robot raggiunga.

Una volta ottenuta la posizione del goal e riportata nelle coordinate della mappa, viene creato il campo attrattivo verso di essa e, tramite la composizione delle velocità generate dai due campi presenti sulla mappa, vengono inviati al robot comandi di velocità tangenziale e di velocità angolare attraverso la pubblicazione di messaggi sul topic /cmd\_vel e ricevendo la posizione del robot sul topic /base\_pose\_ground\_truth.

In questo grafo il robot si trova in un ambiente di simulazione realizzato dal simulatore stageros, la struttura rimane invariata, seppur con topic differenti per quanto riguarda la localizzazione, se si utilizza un robot che naviga in ambiente reale.

### 3.2-Componenti

L'intero codice del progetto è scritto in linguaggio C++, vantaggioso per la sua struttura *object-oriented* e adatto allo sviluppo di software maggiormente evoluti come quelli nell'ambito della robotica e facilmente modulabili.

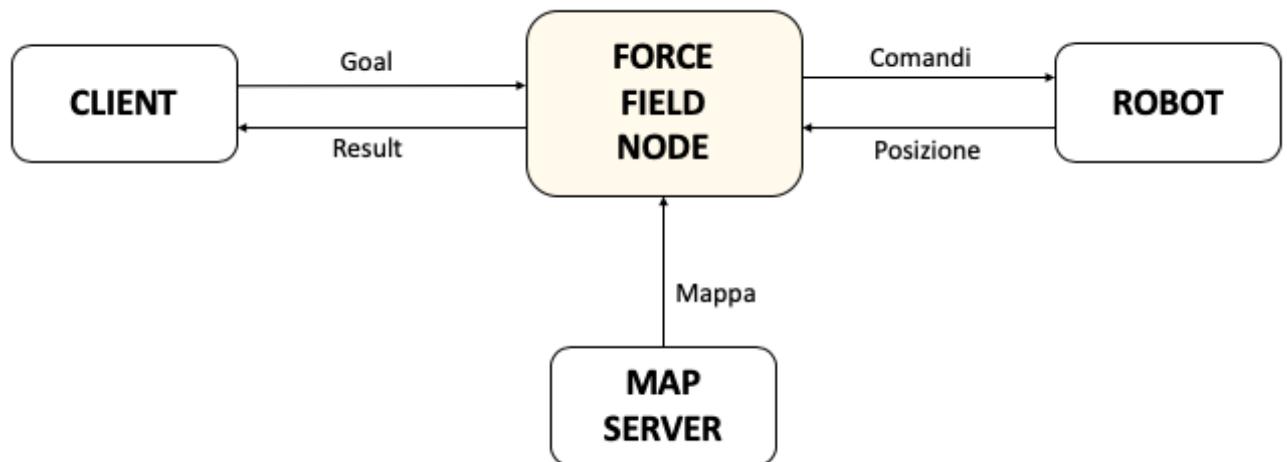


Figura 3.3: Struttura del progetto.

Il progetto prevede lo sviluppo di due nodi ROS: il client\_node e il forceField\_node. Il primo interagisce con l'ambiente esterno tramite il nodo map\_server per quanto riguarda l'acquisizione della mappa e con un nodo che gestisce il simulatore, o il movimento del robot reale, tramite comandi di velocità per quanto riguarda la navigazione.

Il client\_node si occupa invece di inviare un goal in una posizione configurabile da linea di comando oppure graficamente tramite il tool RVIZ messo a disposizione da ROS. Al termine dell'esecuzione del task, il forceField\_node, configurato come Action Server, comunica l'esito della computazione al Client sotto forma di successo o insuccesso.

## Capitolo 4

### Realizzazione del sistema: elaborazione della mappa

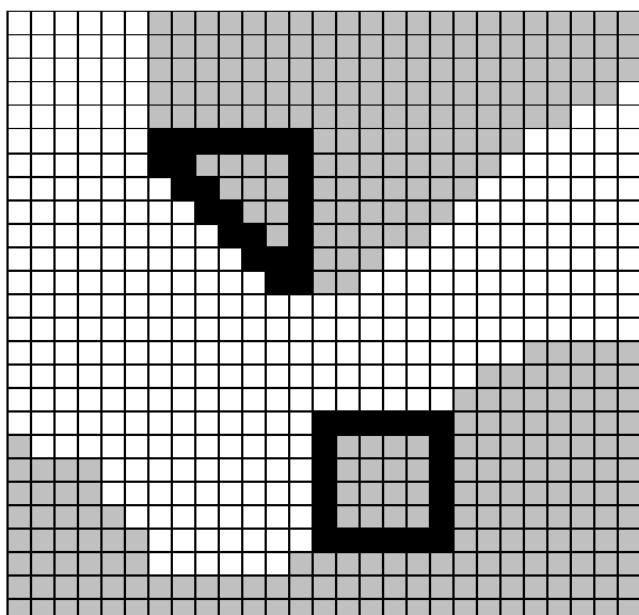
Come sottolineato nel capitolo precedente, la mappa su cui lavora l'algoritmo è di tipo metrico, in particolare abbiamo sfruttato una occupancy grid allo scopo di definire in modo chiaro e semplice le zone alle quali robot può accedere e quelle a lui inaccessibili.

L'ambiente è quindi rappresentato tramite una griglia bidimensionale nella quale ogni cella della mappa può avere tre possibili configurazioni:

- Cellula libera: nel caso in cui lo spazio che essa rappresenta sia totalmente accessibile.
- Cellula occupata: nel caso quello spazio rappresenti un ostacolo.
- Cellula indefinita: se l'algoritmo di SLAM non ha individuato la configurazione di essa.

Il numero di pixel della planimetria che sono contenuti in una cella definisce la risoluzione della mappa.

L'occupancy grid è necessaria per identificare quello che in letteratura si definisce come "free space" ovvero quelle parti dell'ambiente che sono calpestabili e quindi navigabili liberamente da una base mobile.



Le celle bianche identificano il free-space

Le celle nere rappresentano gli ostacoli

Le celle grigie sono quelle indefinite

*Figura 4.1: Esempio di Occupancy Grid Map*

## 4.1-Costruzione della Distance Map

L'utilizzo di una distance Map per rappresentare lo spazio segue il concetto espresso dai Voronoi Diagram, diagrammi che definiscono le configurazioni di "free space" equidistanti dalla regione della mappa in cui sono presenti gli ostacoli. In matematica risultano essere la decomposizione di uno spazio metrico definita come luogo dei punti che hanno distanza massima rispetto ad un determinato insieme discreto di elementi, che, nel nostro caso, sono gli ostacoli.

Ogni cella della mappa memorizza la distanza dal vincolo ad essa più vicino ed i punti equidistanti da uno o più ostacoli formano un picco nel diagramma. L'unione dei punti di picco costituisce uno spazio di possibili percorsi percorribili dal robot.

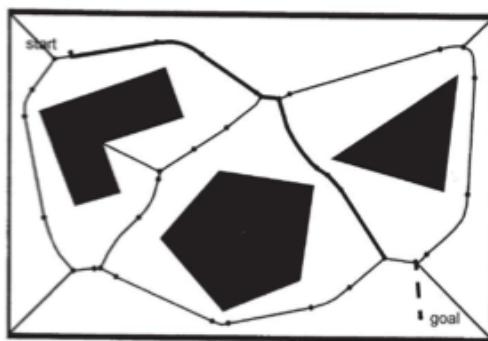


Figura 4.2: Voronoi Diagram

All'interno del progetto questo è stato reso possibile tramite l'implementazione di una classe *DistanceMap* definita come segue:

```
class DistanceMap {  
  
public:  
    struct Cella {  
        Cella();  
        int r;  
        int c;  
        double dist;  
        Cella* parent;  
    };  
  
    struct Origin{  
        Origin(double x_=0.0, double y_=0.0, double yaw_=0.0);  
        double x;  
        double y;  
        double yaw;  
    };  
};
```

```

protected:
    int width;
    int height;
    float resolution;
    Origin origin;

public:
    Cella* dmap;

    DistanceMap(int* map,int w,int h,float res,double xo,double yo,double
yaw,std::vector<int> oc);
    ~DistanceMap();
    int getWidth(){return width;};
    int getHeight(){return height;};
    float getResolution(){return resolution;};
    Origin getOrigin(){return origin;};

};

```

*File DistanceMap.h*

Il costruttore di questa classe viene chiamato dal nodo principale all'arrivo di un messaggio sul topic /map.

La mappa dell'ambiente viene registrata sfruttando gmapping, con il funzionamento descritto nel capitolo precedente. Questo algoritmo ha come output due file che rappresentano la mappa: un file con estensione .pgm che rappresenta un'immagine di essa ed un file descrittivo con estensione .yaml contenente i metadati associati alla mappa, in particolare il suo punto di origine e la sua risoluzione ed i valori di soglia secondo cui le celle sono considerate libere oppure occupate.

La distance map viene inizializzata con i valori della occupancy grid ottenuti dal messaggio di tipo nav\_msgs/OccupancyGrid ricevuto sottoscrivendosi al topic /map sul quale il map\_server lo ha pubblicato.

Vengono quindi inizializzate le tre tipologie di celle possibili come: undefined, ostacolo o cella non visitata.

Ogni cella tiene memoria, oltre che delle sue coordinate, di:

- dist: distanza dell'ostacolo
- parent: puntatore alla cella ostacolo più vicina

Ogni cella ostacolo sarà inizializzata con distanza nulla dall'ostacolo ed avrà se stessa come parent.

A partire da questa configurazione per computare le distanze tra ogni cella e il rispettivo ostacolo più vicino abbiamo implementato una versione modificata dell'algoritmo di Dijkstra.

A partire da ogni cella nell'insieme degli ostacoli abbiamo avviato un'espansione che, considerando la struttura della mappa come matrice, definisce come "vicine" le 8 celle che circondano la cella di partenza.

Analizziamo per ogni cella la distanza memorizzata in essa e la confrontiamo con la distanza che essa avrebbe dall'ostacolo passando per una cella vicina, allo scopo di cercare la minima. Con successive iterazioni riusciamo ad ottenere la configurazione desiderata.

Per ogni cella c espansa la relativa computazione viene presentata sotto forma di pseudocodice come segue:

```
d=distance(p,c)
dost=dmap(p).dist
if(!isvisited(c)){
    dmap(c).dist=d+dost
    dmap(c).parent=dmap(p).parent
}
else if(dmap(c).dist > d+dost){
    dmap(c).dist=d+dost
    dmap(c).parent=dmap(p).parent
}
}
```

*Pseudocodice: computazione di ogni cella espansa*

dove la distanza d identifica la distanza tra la cella p di partenza e la cella c appena espansa, la variabile dost, pari a  $dmap(c).dist$ , rappresenta la distanza dalla cella c al suo ostacolo più vicino che è memorizzata nella distance map;  $dmap(c).parent$  rappresenta invece il puntatore al parent, che identifica l'ostacolo.

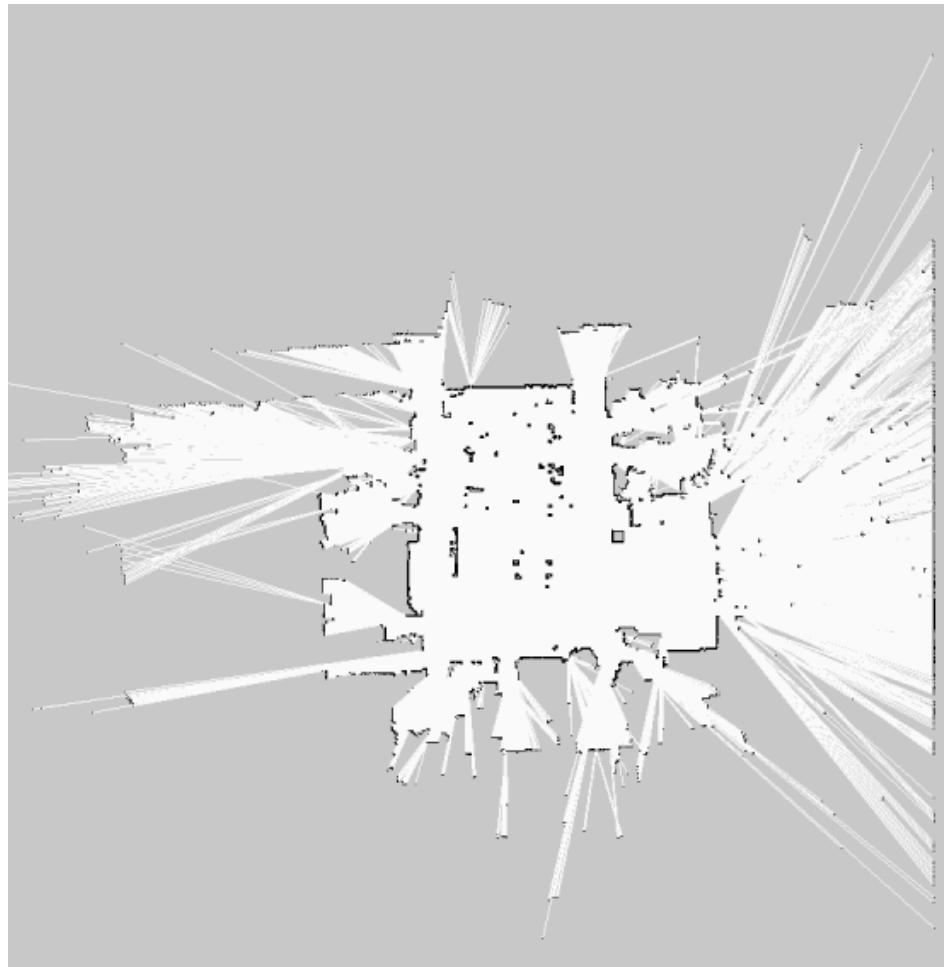
Ad ogni computazione viene valutato se la distanza memorizzata nella distance map relativa alla cella c, è maggiore della somma della distanza tra c e d e la distanza della cella d dall'ostacolo, ed, in caso affermativo, la distanza memorizzata nella distance map viene aggiornata con quella minore.

In questo modo al termine dell'espansione la distance map conterrà, per ogni cella, la distanza relativa all'ostacolo più vicino.

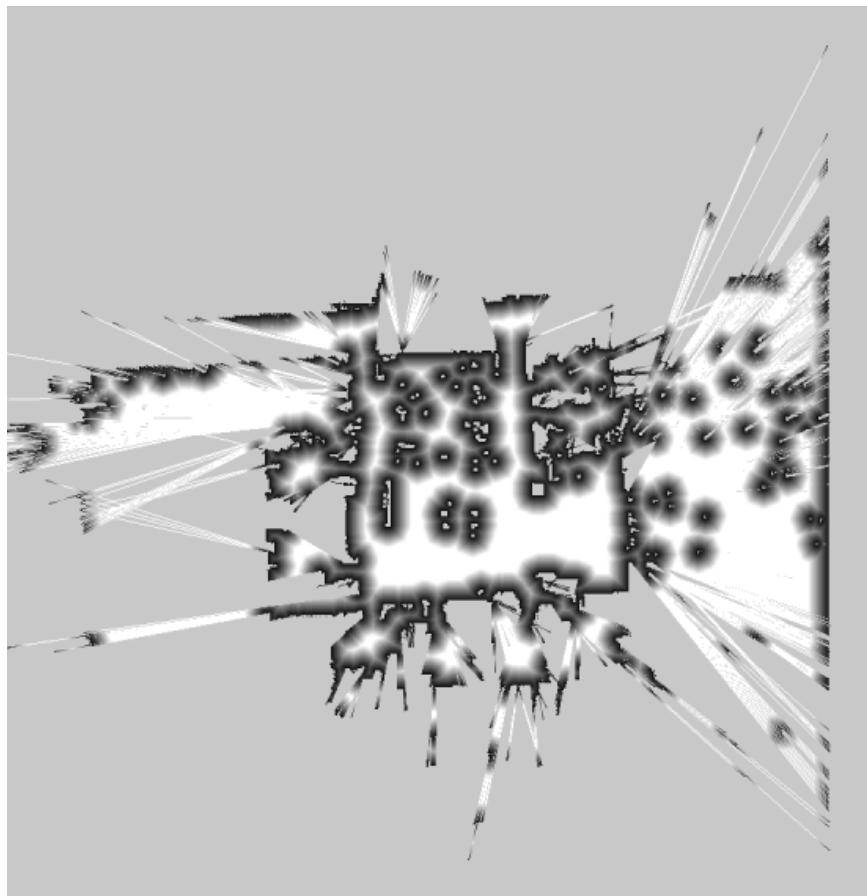
Sfruttando OpenCV, abbiamo realizzato una visualizzazione della distance map tramite un array di tipo CV\_8UC1, ovvero a singolo canale con 8 bit per ogni pixel.

La mappa evidenzia gli ostacoli come pixel di colore nero, le celle libere con colore bianco e con le sfumature di grigio abbiamo mostrato la distanza incrementale dall'ostacolo, con tonalità tanto più scura quanto più la cella è vicina ad esso.

Si mostra come esempio la mappa registrata nell'ambiente virtuale willow-erratic.world all'interno del simulatore stageros di ROS a confronto con la sua rispettiva distance map costruita con l'utilizzo dell'algoritmo precedentemente descritto.



*Figura 4.3: willow-erratic.world*



*Figura 4.4: Distance map della Figura 4.3*

Essendo il progetto pensato e sviluppato per un robot con base mobile di 30 cm abbiamo scelto di considerare l'influenza degli ostacoli fino ad un massimo di 1 metro da essi.

## 4.2-Applicazione dei Campi di Forza Repulsivi

Ad ogni movimento del robot il nodo principale riceve un messaggio di tipo `nav_msgs/Odometry` contenente la posa pubblicata dal localizzatore, messo a disposizione dal *Dipartimento di Ingegneria Informatica e Automatica – Laboratorio RoCoCo*, tramite la quale si possono ottenere le sue coordinate in quello specifico istante.

A quelle coordinate spaziali abbiamo applicato una trasformazione per riportare le coordinate dal sistema di riferimento relativo all'origine degli assi della mappa, con la rispettiva risoluzione, al sistema di celle implementato nella distance map, sotto forma di indici ( $i_r, j_r$ ).

$$i_r = \frac{(y_r - y_{origin})}{risoluzione}$$

$$j_r = \frac{(x_r - x_{origin})}{risoluzione}$$

Per considerare lo spazio dal punto di vista della base mobile abbiamo applicato una matrice di rotazione al vettore che rappresenta la distanza tra il robot e l'ostacolo più vicino, sfruttando come angolo di rotazione quello del robot rispetto all'ambiente circostante, *theta\_r*, e spostandoci quindi nel suo sistema di riferimento.

Il quaternione relativo all'orientazione del robot si ottiene dallo stesso messaggio dal quale si sono precedentemente ottenute le coordinate.

```
Quaternion q = pose.orientation
a= 2* (q.w * q.z + q.x *q.y);
b= 1 - 2*( q.y * q.y + q.z *q.z );
theta_r= arctan(a / b);
```

*Pseudocodice: angolo di rotazione del robot*

I campi di forza presenti in quel punto specifico vengono calcolati durante la navigazione del robot. In questa fase l'obstacle avoidance viene realizzata tramite una velocità tangenziale ed una velocità angolare dirette lungo la retta che congiunge il robot e l'ostacolo a lui più vicino, ma in verso opposto.

Le coordinate dell'ostacolo più vicino si ottengono dal puntatore al parent relativo alla cella c, in cui si trova la base mobile, e sono espresse nel sistema della distance map sotto forma di indici (io, jo). Valutiamo quindi la distanza tra la cella parent e la posizione del robot nella mappa, tenendo sempre in considerazione la risoluzione della mappa e, tramite l'applicazione di una matrice di rotazione, calcoliamo l'angolo di rotazione necessario per impostare la traiettoria lungo la retta tra il vincolo e il robot.

```
dist= dmap(c).dist
dist_obst_y = (ir -io)*res
dist_obst_x = (jr-jo)*res

rot_obst = (dist_obst_x/dist)*cos(theta_r) - (dist_obst_y/dist)*sin(theta_r)
```

*Pseudocodice: rotazione in direzione dell'ostacolo*

### 4.3-Cenni sul Path-Planning e setting delle velocità al robot

Il progetto complessivo prevede che il robot eviti gli ostacoli grazie al campo repulsivo e contemporaneamente, tramite un opportuno campo di forza attrattivo, raggiunga il goal richiesto.

Il client\_node è il nodo responsabile della scelta del goal da inviare al robot ed è realizzato come “*simple\_action\_client*” tramite la libreria ActionLib e sfruttando messaggi di tipo move\_base.

In quest’ottica il nodo principale, forceField\_node, si configura come il server Actionlib che esegue il task richiesto, ovvero la navigazione verso il goal, alla ricezione del messaggio da parte del client.

Dopo il settaggio del goal la componente attrattiva e la componente repulsiva vengono combinate tra loro in modo tale da imporre al robot delle velocità proporzionali alla componente risultante delle forze.

Le coordinate del goal nella mappa e la sua distanza dal robot vengono calcolate come fatto precedentemente per l’ostacolo.

Essendo la forza repulsiva presente nelle vicinanze degli ostacoli, questo può generare cambiamenti di velocità bruschi e può innescare oscillazioni nel caso in cui ci siano ostacoli molto vicini tra loro.

Nelle vicinanze di un ostacolo, quando la distanza da esso è minore di un metro, la velocità del robot viene ridotta e impostata al 30% di quella precedente. Questo per garantire un margine di sicurezza maggiore ed avere il tempo di effettuare la rotazione necessaria per evitare l’ostacolo. In particolare, la velocità angolare, omega  $\omega$ , sarà pari a:

$$\omega = -k2 (rot_{obst})$$

dove k2, è una costante di gain adattabile all’ambiente nel quale avviene la navigazione, nei nostri esperimenti impostata a 1.

La legge di controllo sviluppata per quanto riguarda il campo attrattivo è la seguente:

$$v = k1 (x_{dist\_goal} \cos \theta_r - y_{dist\_goal} \sin \theta_r)$$

$$\omega = k2 \left( \arctan \left( \frac{y_{goal}}{x_{goal}} \right) \right)$$

dove  $x_{dist\_goal}$  e  $y_{dist\_goal}$  rappresentano le distanze dall'obiettivo nel sistema di riferimento globale;  $\theta_r$  rappresenta l'angolo di orientazione  $\text{theta\_r}$  del robot ,e  $k1$  è una costante di gain in questo caso specifico impostata a 5.

$Y_{goal}$  e  $X_{goal}$  sono invece le coordinate dell'obiettivo nel sistema di riferimento del robot, calcolate mediante:

$$\begin{pmatrix} x_{goal} \\ y_{goal} \end{pmatrix} = \begin{pmatrix} \cos \theta_r & \sin \theta_r \\ -\sin \theta_r & \cos \theta_r \end{pmatrix} \begin{pmatrix} x_{dist\_goal} \\ y_{dist\_goal} \end{pmatrix}$$

I comandi di velocità vengono inviati al robot tramite la pubblicazione sul topic `/cmd_vel` di messaggi di tipo `geometry_msgs/Twist` contenenti la risultante della somma dei comandi di velocità repulsivi e attrattivi.

```
geometry_msgs::Twist twist;
twist.linear.x = v;
twist.angular.z = omega;
twist_pub.publish(twist);
```

*Geometry\_msgs::Twist inviato*

Prima della pubblicazione del messaggio, sono stati implementati dei controlli di saturazione, necessari per effettuare delle sperimentazioni in ambiente reale in sicurezza, allo scopo di non superare una soglia di velocità traslazionale e angolare che può variare secondo le caratteristiche specifiche del robot utilizzato.

# Capitolo 5

## Sperimentazione

### 5.1-Esperimenti con simulatore

L'ambiente di simulazione in cui è stato svolto il *testing* del sistema progettato è reso disponibile dal nodo stageros del framework ROS contenuto all'interno del package stage\_ros. Questo nodo sfrutta un simulatore multi-robot Stage 2-D tramite libstage<sup>4</sup>.

La definizione dell'ambiente virtuale si trova in un file.world che mette a disposizione tutte le informazioni necessarie alla navigazione come il robot e gli ostacoli, di norma rappresentati mediante una bitmap mostrata come sfondo del mondo.

Il simulatore Stage garantisce un trade-off tra simulazioni ad alta fedeltà e grid-world simulations, si considera inoltre il rumore che è ottenuto indirettamente attraverso la discretizzazione dello spazio.

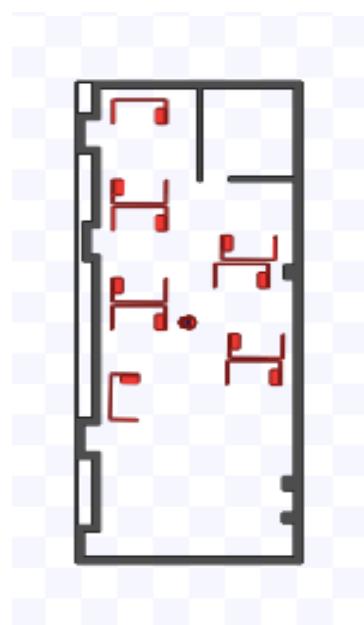


Figura 5.1: uoa\_robotics\_lab.world

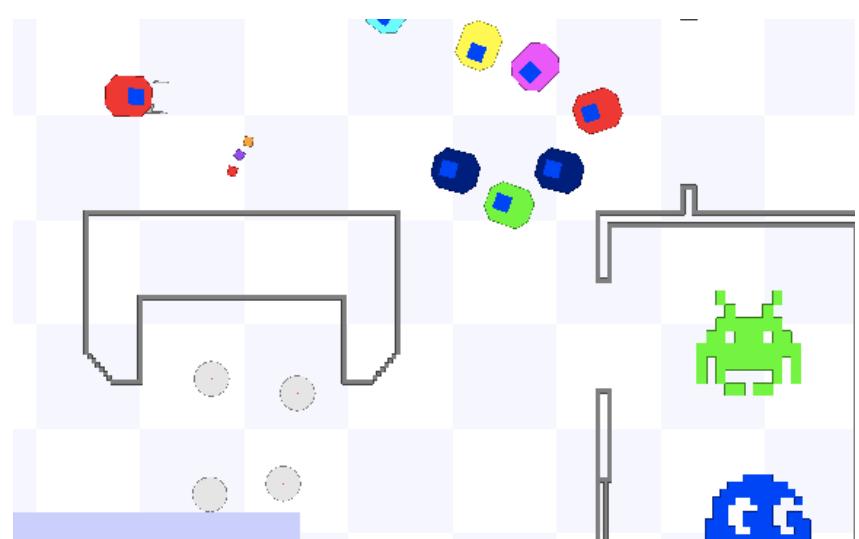
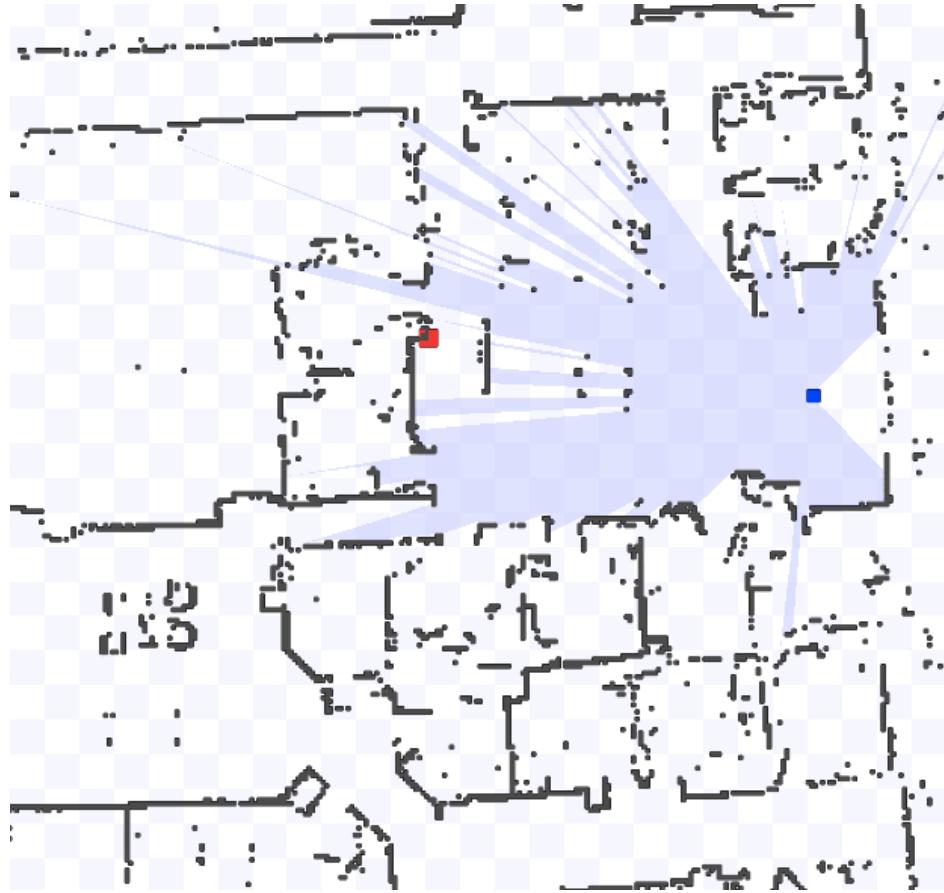


Figura 5.2: everything.world

<sup>4</sup> Libreria C utilizzata per implementare una simulazione robot in user code

Il mondo utilizzato per il test del progetto è `willow-erratic.world`.



*Figura 5.3: willow-erratic.world*

Questo ambiente fornisce un robot con a bordo un laser ranger che viene sfruttato per l'acquisizione della mappa.

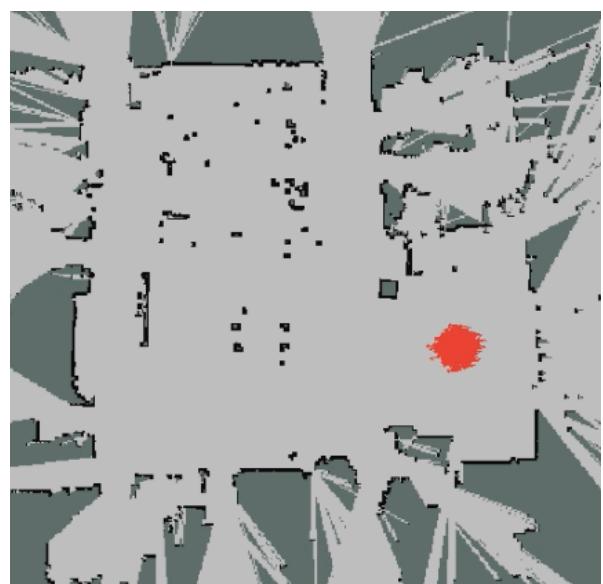
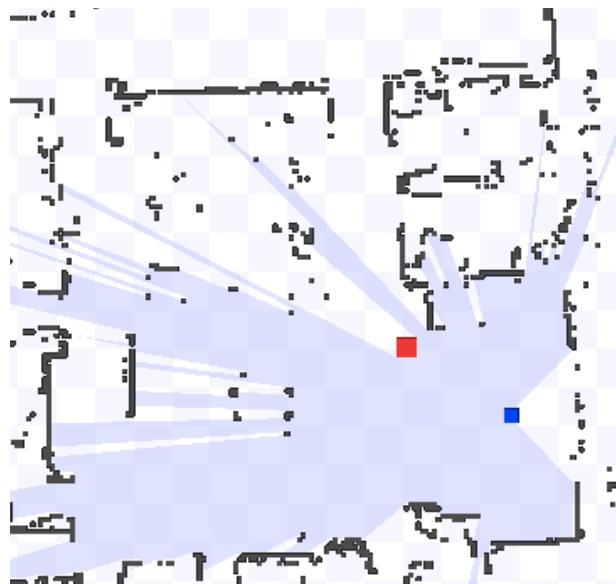
Le informazioni sensoriali del laser si ottengono navigando per il mondo sfruttando un nodo `teleop_twist_keyboard` che permette di guidare la base mobile all'interno dello spazio di lavoro. Questi dati vengono raccolti tramite la registrazione di una `rosbag` che memorizza i messaggi scambiati su tutti i topic attivi durante la fase di esplorazione del mondo. Riproducendo la `bag` durante l'esecuzione dell'algoritmo di SLAM, realizzato dal nodo `gmapping`, è possibile ottenere la mappa del mondo alla quale è possibile applicare l'algoritmo descritto nel Capitolo 4.

Per visualizzare lo spostamento del robot all'interno della mappa durante la fase di sperimentazione abbiamo sfruttato il tool RVIZ, un'interfaccia grafica fornita da ROS.

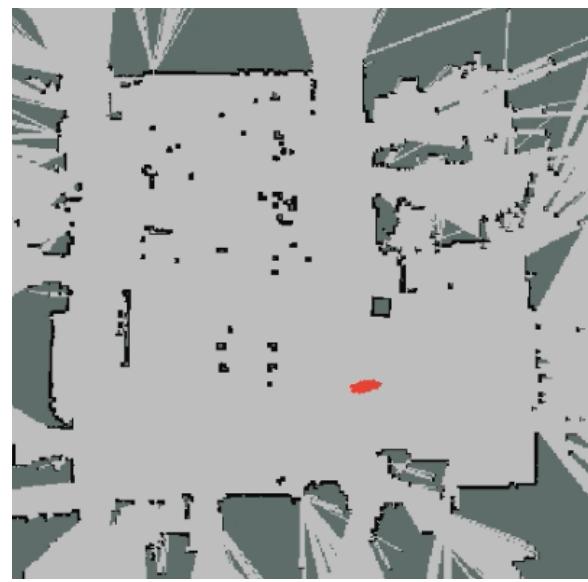
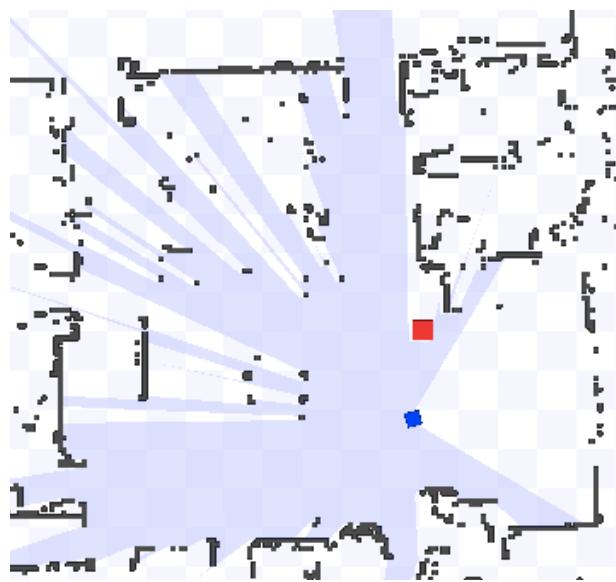
La sperimentazione si articola di svariati test, ne vengono mostrati tre di seguito:

- Test 1:

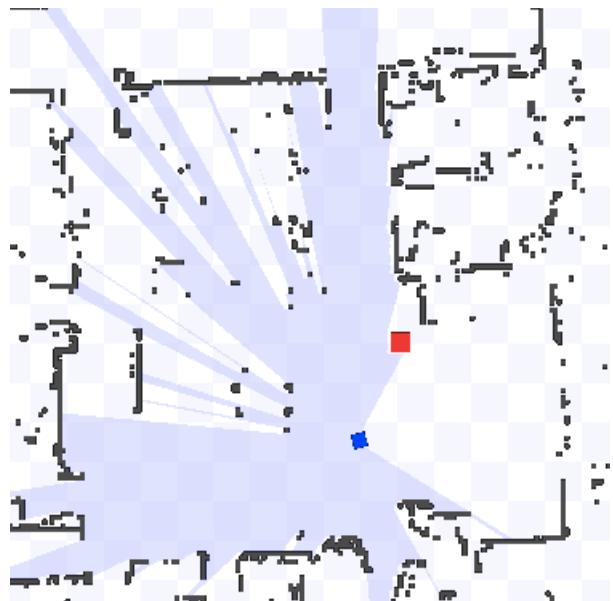
- posizione di partenza:  $x=-11.28$ ;  $y=23.27$ ;  $z=0.0$ ;  $w=180.0$
- posizione di arrivo:  $x=-20.875$ ;  $y=20.920$ ;  $z=0.998$ ;  $w=-0.067$



a) punto iniziale: visualizzata in stageros (sinistra) e RVIZ (destra)



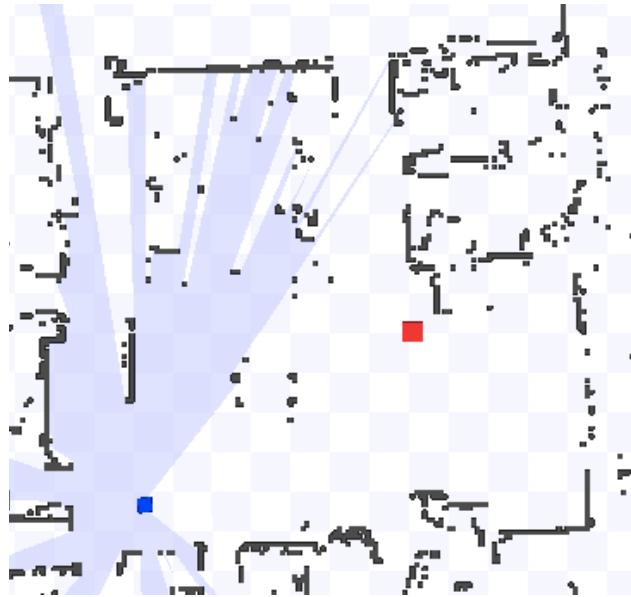
b)



c)



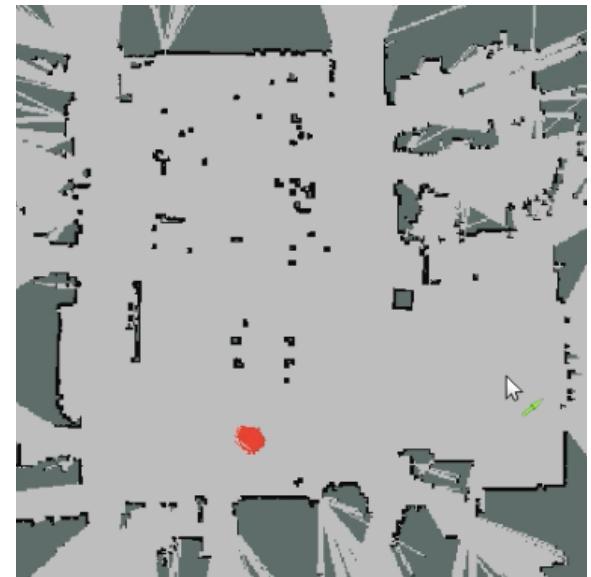
d)



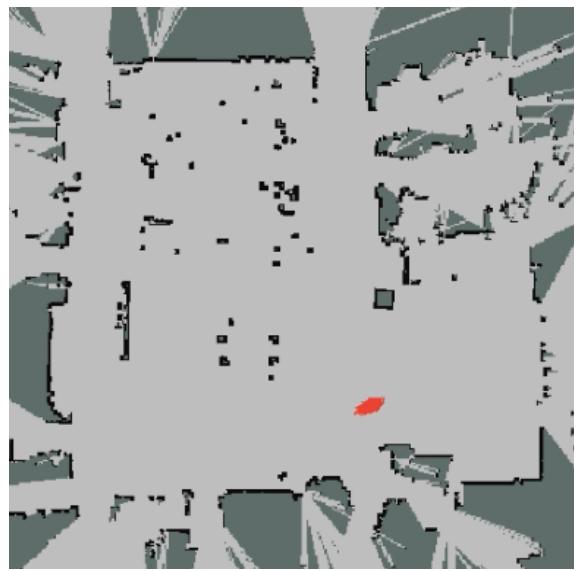
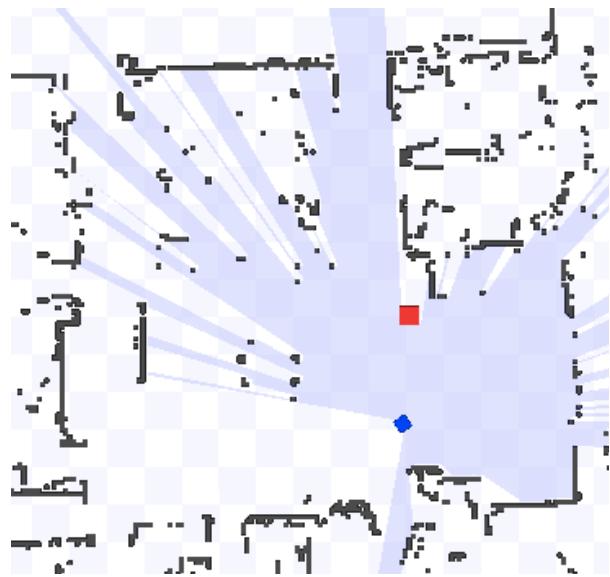
e) punto di arrivo: obiettivo raggiunto

- Test 2: percorso inverso

- posizione di partenza:  $x=-20.875$ ;  $y=20.920$ ;  $z=0.998$ ;  $w=-0.067$
- posizione di arrivo:  $x=-11.28$ ;  $y=23.27$ ;  $z=0.0$ ;  $w=0.0$



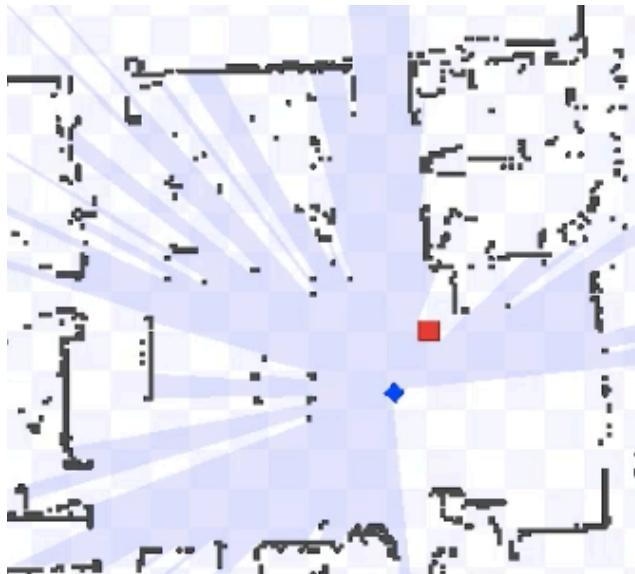
a2)



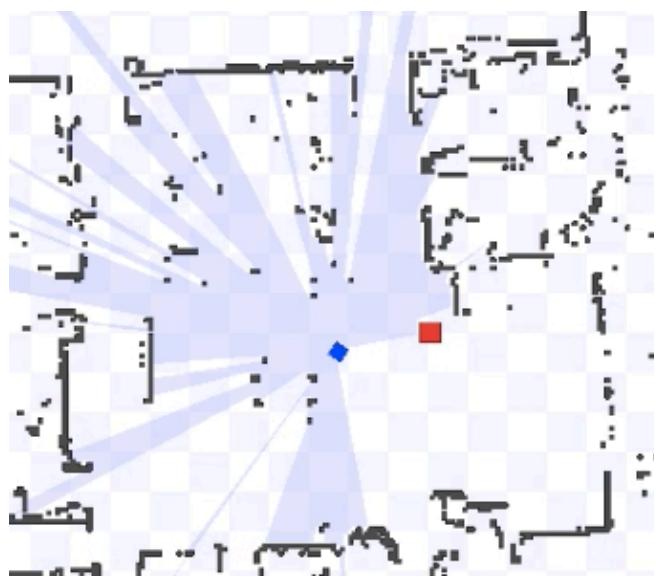
b2)

- Test 3:

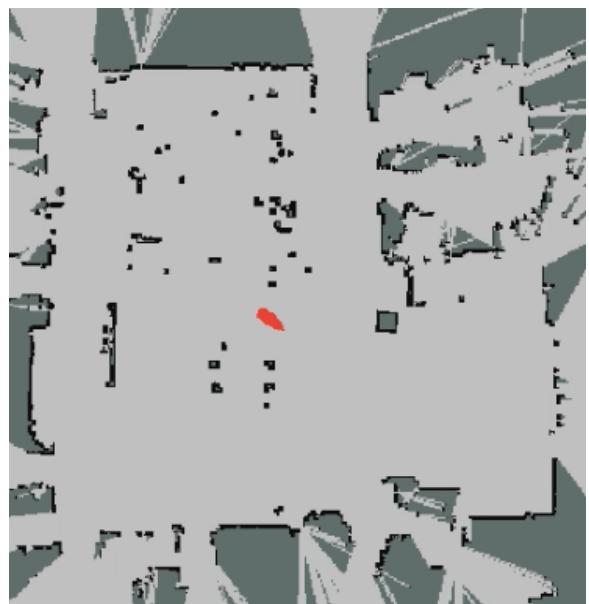
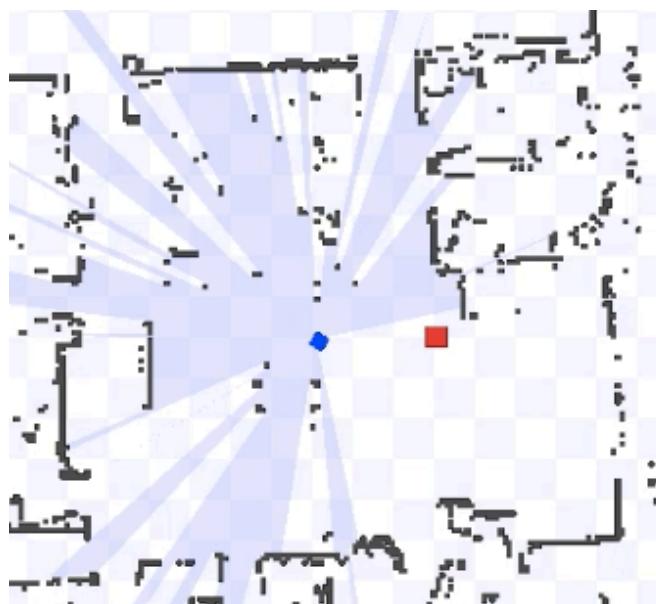
- posizione di partenza:  $x=-11.28; y=23.27; z=0.0; w=0.0$
- posizione di arrivo:  $x=-17.20; y=25.0; z=0.0; w=0.0$



a3)



b3)



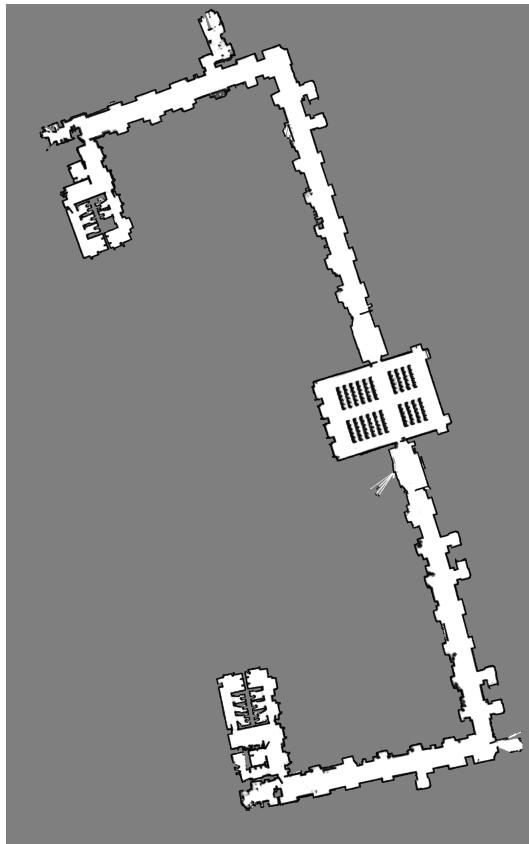
c3)

## 5.2-Esperimenti in ambiente reale

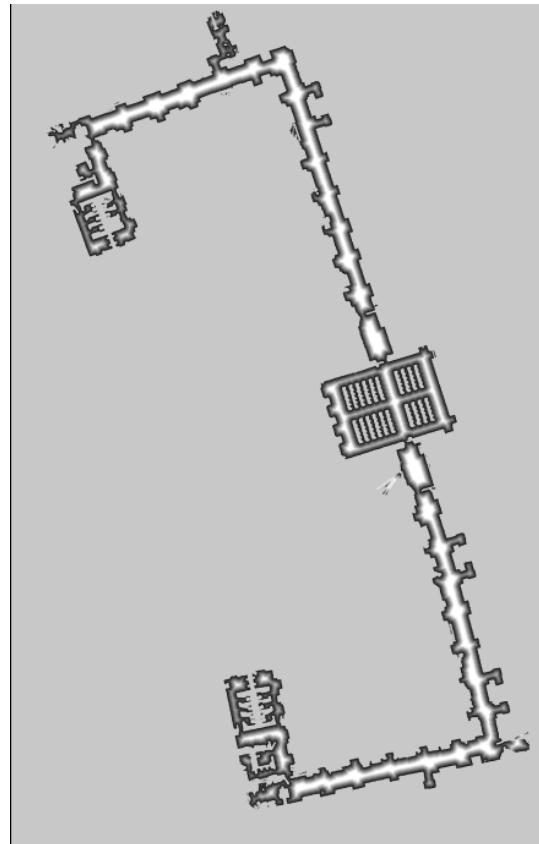
La sperimentazione è stata realizzata tramite il robot differential drive MARRtino: piattaforma robotica a basso costo basata sul sistema ROS, messa a disposizione dal *dipartimento di Ingegneria Informatica e Automatica – Laboratorio RoCoCo*.

La progettazione di questa piattaforma è finalizzata alla costituzione di un robot facile da programmare e costruire, ma che, allo stesso tempo, possa essere adatto a svolgere tipici task di intelligenza artificiale e robotica, quali navigazione o analisi di immagini.

Il test del progetto in ambiente reale è stato eseguito nell'edificio del DIAG, del quale è stata realizzata la mappa del primo piano, tramite la metodologia spiegata nel Capitolo 4, dopo aver registrato una bag sfruttando i dati raccolti dal laserscan a bordo di MARRtino.



*Figura 5.4: mappa del corridoio DIAG: I piano*



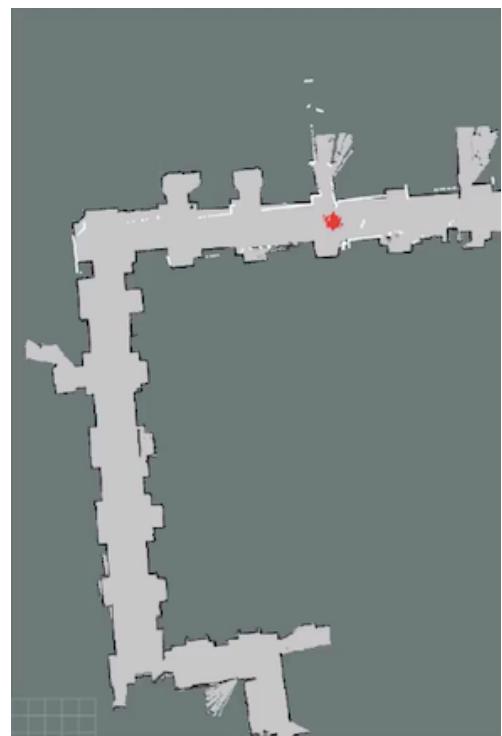
*Figura 5.5: Distance map della mappa in Figura 5.4*

L'influenza del campo repulsivo degli ostacoli sulla navigazione del robot è stata considerata fino ad un massimo di un metro, considerando la larghezza di MARRtino di 30 cm.

I casi di test sono stati realizzati lungo il corridoio dell'ala B del primo piano dell'edificio.

L'obiettivo, in questo caso, è stato settato, tramite il tool RVIZ, dalla mappa.

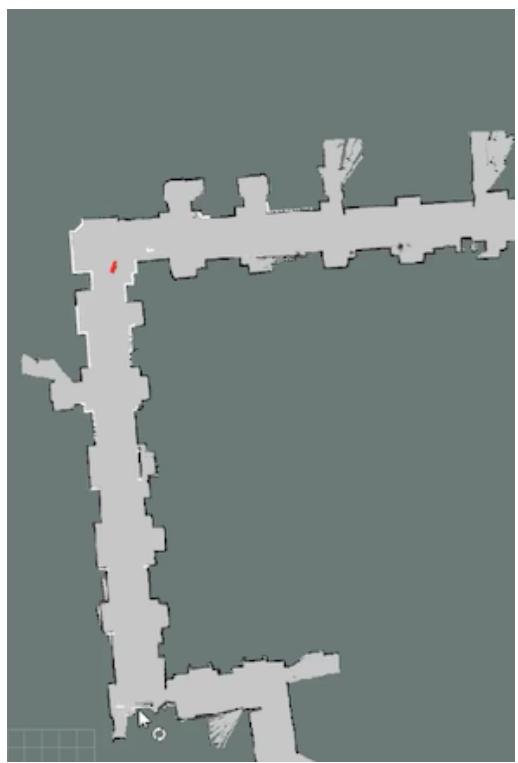
Seguono le immagini che illustrano la navigazione: a sinistra fotografie scattate durante la fase di sperimentazione e a destra fermo-immagine della navigazione mostrata su RVIZ.



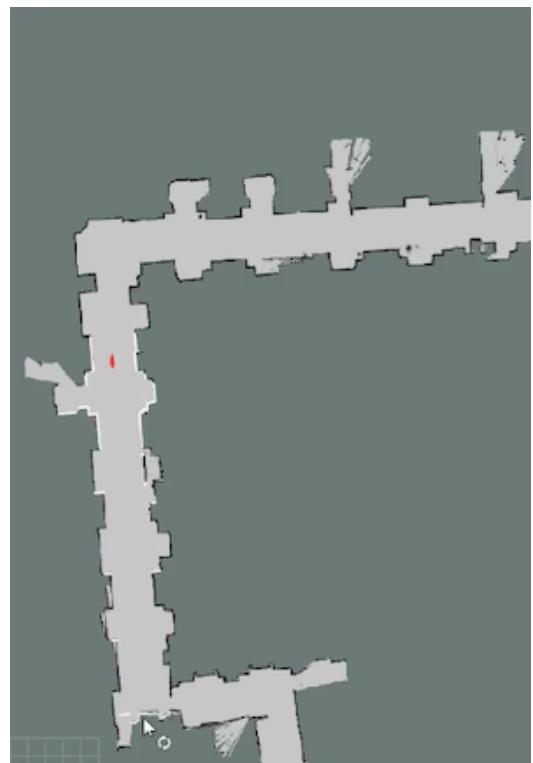
a)



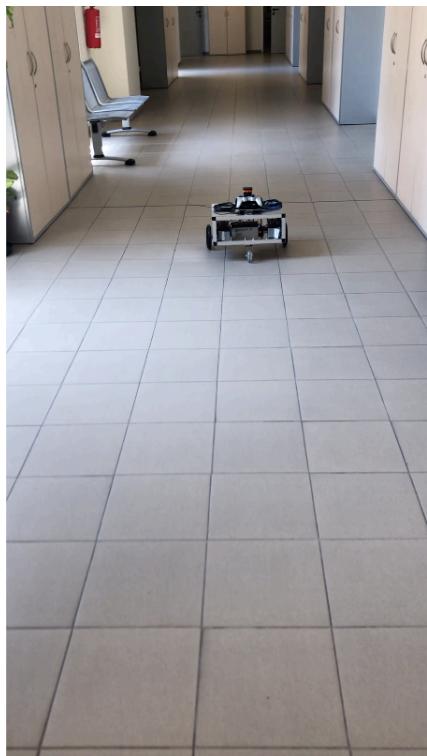
b)



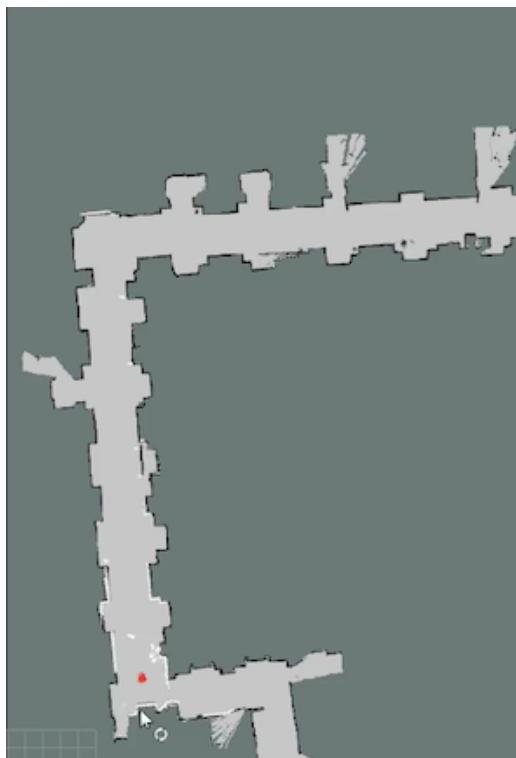
c)



d)



e)



f)

Nelle immagini di RVIZ si può osservare il movimento del robot tramite lo spostamento delle particelle in rosso che rappresentano la stima della sua posizione istante per istante.

Come si può osservare negli esempi mostrati precedentemente, il robot segue la traiettoria dettata dall'angolo che congiunge la sua posizione alla posizione di goal e procede seguendo quest'ultima fin quando non si trova in prossimità di un ostacolo.

In questa condizione, il robot viene ruotato in un angolo opposto a quello tra lui e l'ostacolo, evitando così una collisione e procedendo, successivamente, lungo la traiettoria verso il goal [cambio di direzione evidente nelle immagini *a2* e *b2* del simulatore e *c* dell'ambiente reale].

Nei pressi di un ostacolo, inoltre, la velocità settata al robot viene ridotta, questo per evitare che non abbia il tempo di effettuare la rotazione necessaria per sottrarsi allo scontro.

# Capitolo 6

## Conclusioni

Questa tesi si è proposta di realizzare un modulo di navigazione per un robot mobile in ambienti statici indoor. Nello svolgimento della tesi, dopo un'introduzione al problema volta a metterne in luce le caratteristiche principali, abbiamo sviluppato una possibile soluzione basata sull'applicazione di due diverse tipologie di campi di forza artificiali. In particolare, di un campo di forza repulsivo per ogni ostacolo presente nell'ambiente mappato e di uno repulsivo centrato nell'obiettivo finale della navigazione.

Per raggiungere questo scopo abbiamo implementato una distance map che si caratterizza come una rappresentazione discreta dell'ambiente mediante una matrice formata da celle, ognuna contenente la sua distanza dall'ostacolo più vicino e un puntatore alla cella che identifica quest'ultimo.

Lo studio approfondito delle tecnologie e dei software disponibili ha permesso di sviluppare, come mostrato nel quarto capitolo, un algoritmo in grado di risolvere il problema mediante un settaggio delle velocità tangenziale e angolare al robot.

L'utilizzo di campi potenziali per risolvere il problema della navigazione ha mostrato numerosi vantaggi, tra i più importanti c'è la possibilità di predire con facilità il comportamento del robot all'interno dell'ambiente e la flessibilità del meccanismo implementato. Quest'ultima è evidente osservando le equazioni di controllo nelle quali, tramite una modifica dei parametri di gain è possibile evidenziare un sotto-comportamento del robot: attrattivo o repulsivo. Si può pensare, in un'ottica di miglioramento dell'algoritmo, di non pre-determinare questi pesi ma, di settarli tramite altri meccanismi che operano real-time nel robot e di implementare un meccanismo che permetta all'agente autonomo di imparare e ricordare approcci che ha precedentemente utilizzato per evitare ostacoli con configurazione simile.

Tuttavia, un problema che si presenta nell'utilizzo dei campi potenziali è la presenza dei minimi locali, ovvero delle configurazioni nelle quali la sommatoria delle forze attrattiva e repulsiva sono nulle e il robot si trova bloccato in una posizione diversa da quella di goal.

Ad esempio se il robot si trova in una posizione in cui ha l'obiettivo di fronte e l'ostacolo posto dritto nella sua traiettoria verso l'obiettivo, la forza repulsiva dall'ostacolo annulla quella attrattiva verso l'obiettivo e, di conseguenza, la navigazione si interrompe in quel punto.

Un approccio basilare per ridurre la possibilità che il robot rimanga bloccato in un minimo locale è l'aggiunta di un *random noise*, ovvero di un campo di forze presente in ogni punto della mappa con dei vettori di forze direzionali in maniera casuale. In questo modo il robot verrà trascinato via dal minimo. Sfruttando questa metodologia non è però garantito che non possano esistere comunque dei minimi locali dai quali il robot non riesce ad uscire; è inoltre possibile che l'introduzione di *random noise* generi un comportamento ciclico in cui il robot, pur spostandosi, torna sempre nella posizione di minimo. Per risolvere il problema si può fare in modo che il robot memorizzi le posizioni dove è stato e si allontani da esse, secondo un approccio *avoid-past behavior*.

Il progetto presentato in questa tesi fornisce una metodologia di navigazione che garantisce di evitare gli ostacoli statici presenti nell'ambiente e quindi mappati nella distance map, un *future work* per espandere questo progetto è l'introduzione di un meccanismo di *obstacle avoidance* che permetta di evitare anche gli ostacoli dinamici che il robot può incontrare nel suo percorso. Il rilevamento degli ostacoli si può realizzare tramite diverse tipologie di sensori come, ad esempio: sonar, laser range finder, visione stereo o sensore di profondità 3D. La reazione agli ostacoli imprevisti deve essere rapida, efficace ed affidabile.

## Bibliografia

- [1] Liam Paull, Gaetan Severac, Guilherme V. Rao, Julian M. Angel, Harold Boley, Maki K. Habib, Bao Nguyen, Veera R. S. Kumar, Sajad Saeedi, Ricardo Sanz, Mae Seto, Aleksandar Stefanovski, Michael Trentini, Howard Li. "**Towards An Ontology for Autonomous Robots**", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Algarve, Portugal, Oct 7-12, pp. 1359-1364, 2012.
- [2] Thomas Hellstrom, Department of Computing Science Umea University, "**Robot Navigation with Potential Fields**", Dec 19, 2011 UMINF-11.18 ISSN-0348-0542.
- [3] Documentazione ROS, <http://wiki.ros.org>
- [4] Giorgio Grisetti, Dipartimento Informatica e Sistemistica Università "La Sapienza" I-00198 Rome, Itali; Cyrill Stachniss, Wolfram Burgards, University of Freiburg Department of Computer Science D-79110 Freiburg, Germany "**Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptative Proposals and Selective Resampling**".
- [5] Mojtabahedzadeh, Rasoul. (2011). "**Robot Obstacle Avoidance using the Kinect**".
- [6] Corrado Santoro. "**Navigation, path planning e obstacle avoidance**".
- [7] Documentazione OpenCV, <https://docs.opencv.org>
- [8] Sito Web MARRtino <https://www.marrtino.org>
- [9] Balch, Tucker & Arkin, Ronald. (1995). "**Avoiding the Past: A Simple but Effective Strategy for Reactive Navigation**", Proceedings from IEEE International Conference on robotics and automation.