SAPIENZA
UNIVERSITÀ DI ROMA

Costruzione di una mappa semantica di navigazione

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Dipartimento di Ingegneria Informatica, Automatica e Gestionale
Corso di laurea in Ingegneria Informatica

Stefano Catozi
Matricola 1716956

Relatore
Daniele Nardi

A.A. 2018-2019

Indice

Capitolo 1

Introduzione

Questa Tesi, sviluppata nell'ambito del Laboratorio di Intelligenza Artificiale e Grafica Interattiva del corso di Laurea in Ingegneria Informatica, ha lo scopo di illustrare come sia possibile per un robot orientarsi e muoversi in un ambiente a lui inizialmente sconosciuto e come estrapolare all'interno di questo ambiente delle informazioni utili alla sua comprensione, come potrebbero essere la disposizione nello spazio delle varie stanze e degli oggetti presenti, che permettono di costruire la cosidetta Mappa Semantica dell'ambiente.

Si vuole sottolineare quindi come lo sviluppo di soluzioni simili in ambito commerciale ed industriale sia di estrema rilevanza al giorno d'oggi in quanto permetterebbe di semplificare molto aspetti estremamente rilevanti per la nostra società, da quelli legati ad un ambiente casalingo a quelli delle grande industrie. Si pensi ad esempio ai robot che sono in grado di aiutare persone anziane e chiunque abbia difficoltà, questi tipi di robot sono già molto usati e poggiano proprio su questa tecnologia per mappare ad esempio le stanze della casa ed essere così molto più utili e facili da utilizzare per l'utente.

Un'altra applicazione che si basa sull'utilizzo di Mappe Semantiche e sta conoscendo un' incredibile avanzamento tecnologico e commerciale negli ultimi anni è quello dei veicoli autonomi.

Google, attraverso la sua azienda Waymo, sta infatti investendo molte risorse sullo sviluppo di automobili in grado di muoversi in qualunque situazione di traffico e in qualunque ambiente in modo completamente autonomo, escludendo quindi ogni interazione necessaria da parte dell'utente[1]. Questi veicoli sono in grado di esplorare ed analizzare l'ambiente circostante con tecniche come radar, GPS e visione artificiale. A queste tecnologie si affiancano sistemi di controllo molto avanzati basati sull'uso di sensori presenti nel veicolo. Questi sistemi permettono di interpretare le informazioni ricevute per individuare i percorsi migliori in cui muoversi o anche la presenza di ostacoli. La tecnologia usata da questi veicoli per la localizzazione è proprio la SLAM (Simultaneous Localization And Mapping) che verrà approfondita nel paragrafo 3.1, i sistemi più avanzati però utilizzano una sua variante (SLAM con DATMO) che riesce a mappare e localizzare anche oggetti in movimento.[2]

Infine un'ultima applicazione potrebbe essere quella dell'uso interno di robot industriali. Questi robot vengono utilizzati appunto all'interno di grandi aziende per semplificare lavori che sarebbero estremamente pesanti ed usuranti per i dipendenti come ad esempio il trasporto di materiale pesante.[3]

I robot ti questo tipo fino ad ora utilizzati sono stati del tipo AGV (Veicolo a Guida Automatica) i quali però avevano bisogno di una struttura fisica fissa per lavorare come dei fili di acciaio lungo i quali muoversi, o sensori nel pavimento per permettere l'orientamento nello spazio, limitando così notevolmente le applicazioni di questa tecnologia.

Si sta quindi facendo molta ricerca su robot di tipo AMR (Robot Mobili Autonomi) che invece implementano sistemi che li rendono indipendenti da strutture esterne potendo essere così utilizzati in qualunque situazione con estrema flessibilità.[4]

E' proprio su questi aspetti che si fonda questa Tesi, sulla possibilità di creare delle Mappe Semantiche che possano servire allo sviluppo di sistemi simili a quelli descritti che avranno nell'immediato futuro un enorme espansione e sulla loro navigazione autonoma.

Nel Capitolo 2 verrà presentato ROS (Robot Operating System), un ambiente di sviluppo software che rende molto più facile lo sviluppo di robot di quanto non fosse in precedenza fornendo una grande varietà di sistemi e tool per la robotica. Verranno anche presentati nel dettaglio i vari strumenti messi a disposizione dal sistema che sono stati utilizzati per lo sviluppo di questa Tesi. E' il sistema sul quale è stata costruita questa Tesi.

Nel Capitolo 3 verrà analizzato cose vuol dire e come si può realizzare una Mappa Semantica, partendo dalla Mappa Metrica per poi analizzare i vari metodi di creazione e sviluppo di una Mappa Semantica.

Nel Capitolo 4 invece si avrà modo di spiegare nel dettaglio quale è stato l'approccio e lo sviluppo utilizzato nella presente Tesi per affrontare questo genere di problemi, quindi come sono stati utilizzati gli strumenti di ROS all'interno del progetto e quali sono state le sperimentazioni a cui è stato sottoposto il sistema creato.

Nell'ultimo Capitolo vengono tratte le conclusioni sul progetto, ricapitolando cosa è stato analizzato e sottolineando le potenzialità dell'argomento fornendo dei possibili ampliamenti a questo genere di sistemi che potrebbero essere sviluppati in futuro.

Capitolo 2

ROS (Robot Operating System)

ROS (Robot Operating System) è un insieme di framework per lo sviluppo e la programmazione di robot basato su Linux e distribuito con licenza open-source (BSD). I linguaggi attualmente supportati sono C/C++, Python e Lisp, mentre Java e Lua sono in via di sviluppo.[5]

Nasce nel 2007 sulla base di alcune ricerche della Stanford University sulla robotica e sull'intelligenza artificiale, come Stanford AI Robot (STAIR) e il Personal Robots (PR) program, che vennero ampliate dalla compagnia Willow Garage portando così alla creazione di ROS.[6]
Lo scopo principale della Willow Garage era quello di creare un sistema che permettesse a chi sviluppava robot di poter risparmiare molto tempo evitando di dover ripartire dalla progettazione di base per ogni nuovo modello. Il sistema ROS permette infatti di avere una solida base sulla quale sviluppare il proprio robot, formata da un grande numero di tool diversi che possono essere messi in connessione tra loro per poter essere poi utilizzati nel proprio progetto, il che rende il sistema estremamente flessibile e condivisibile. Infatti questo concetto di modularità è ciò che più di tutto ha permesso a ROS di guadagnare popolarità ed essere oggi usato in moltissime Università e centri di ricerca per lo sviluppo della robotica, inoltre essendo completamente open-source ogni gruppo di ricerca può avere la sua repository ROS e mantenere il completo controllo su di essa, creando così un folto ecosistema di ricercatori, ad oggi decine di migliaia, che scambiano idee e progetti dai più semplici a grandi sistemi industriali permettendo così un massiccio avanzamento della robotica.[7]

ROS fornisce tutti gli strumenti di un sistema operativo come astrazione dell'hardware, comunicazione tra processi attraverso invio di messaggi, gestione delle applicazioni, librerie e strumenti per la scrittura, compilazione ed esecuzione del codice e controllo dei device di basso livello.

2.1 Concetti Fondamentali in ROS

ROS è diviso in 3 livelli concettuali i quali gestiscono rispettivamente: l'organizzazione dei file su disco, la rete peer-to-peer dei nodi e infine lo scambio di software tra i gruppi di sviluppatori.
Di seguito vengono descritti nel dettaglio.[8]

File system Level:

I pacchetti sono il cuore del file system di ROS e possono contenere un eseguibile (nodo), una libreria di ROS, un dataset o anche file di configurazione e per questo rappresentano l'unità più piccola del sistema ROS. Il loro scopo è quello di rendere la struttura del sistema più modulare possibile potendo essere riutilizzati per diversi progetti e rilasciati per la comunità degli sviluppatori.
Ogni package ha un package manifest cioè un file .xml che contiene i metadata del pacchetto come ad esempio nome, versione, licenza ma anche le dipendenze da altri pacchetti.

Ci sono poi i Packages Resources Names che sono usati per semplificare il riferimento ai tipi di dato sul disco. Sono formati semplicemente dal nome del package dove è la resource seguito dal nome della resource stessa, ad esempio il Package Resource Name del tipo di messaggio (resource) String del package "std_msgs" sarà "std_msgs/String".

Gli Stack sono collezioni di package che collaborano per un unica funzionalità come può essere ad esempio il "navigation stack" che è l'insieme di package che si occupa della navigazione del robot. Anche gli stack hanno il loro manifesto.

Infine ci sono i Service types e Message types che definiscono la struttura rispettivamente dei Servizi e dei Messaggi.

Computational Graph Level:

Questo livello rappresenta la rete peer-to-peer dei processi ROS che si scambiano informazioni.
I concetti principali di questo livello sono implementati nello stack ros_comm e sono:

- Nodi: I nodi sono i processi ROS che eseguono operazioni che quindi, partendo dai dati ricevuti dai vari topic e server, sono in grado di eseguire determinate azioni. Data la forte modularità del sistema ROS ogni nodo si occupa di una specifica operazione del sistema generale che può comprendere molti nodi: ad esempio in un robot un nodo può gestire la navigazione, un nodo la visualizzazione e un altro ancora elaborare i dati del sensore laser.

- Master: Il Master è il centro del Computational Level ed è il responsabile della gestione dei nodi e delle le comunicazioni tra di loro attraverso un sistema di messaggi e servizi. Una parte importante del Master è il Parameter Server che permette di salvare Iidati che devono essere condivisi in uno spazio accessibile a tutti i nodi.
  Il Master si comporta fondamentalmente come un DNS della rete dei nodi: dopo aver ricevuto le informazioni da ogni nodo, le fornisce agli altri permettendo così di creare

connessioni all'interno della rete. E' fondamentale avviare il nodo Master prima di ogni altro nodo in ROS. Il comando che permette di eseguirlo è semplicemente

```
$ roscore
```

- Messaggi: I Nodi comunicano tra di loro attraverso un sistema di scambio di messaggi. I messaggio sono formati da semplici strutture dati che possono essere formati da tipi primitivi e/o da array di questi che possono essere abinate a piacere.

```
Header header           # timestamp in the header is the acquisition time of
                        # the first ray in the scan.
                        #
                        # in frame frame_id, angles are measured around
                        # the positive Z axis (counterclockwise, if Z is up)
                        # with zero angle being forward along the x axis

float32 angle_min       # start angle of the scan [rad]
float32 angle_max       # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]

float32 time_increment  # time between measurements [seconds] - if your scanner
                        # is moving, this will be used in interpolating position
                        # of 3d points
float32 scan_time       # time between scans [seconds]

float32 range_min       # minimum range value [m]
float32 range_max       # maximum range value [m]

float32[] ranges        # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities   # intensity data [device-specific units].  If your
                        # device does not provide intensities, please leave
                        # the array empty.
```

Figura 2.1.1 Esempio di file .msg, in particolare /std_msgs/LaserScan

- Topics: I Messaggi sono distribuiti ai nodi attraverso un sistema di publishers/subscribers in cui ogni nodo pubblica un determinato tipo di messaggi, ad esempio i messaggi sulla posizione del robot nella mappa. A questo topic si sottoscriveranno poi tutti i nodi interessati a questa informazione i quali riceveranno tutti i messaggi pubblicati sul topic. Possiamo immaginare i topic quindi come un bus di comunicazione in cui possono viaggiare solo messaggi dello stesso tipo. In generale ogni nodo può pubblicare/sottoscriversi a uno o più topic contemporaneamente e ogni topic può avere uno o più nodi che si sottoscrivono/ pubblicano. I publisher/subscribers non possono sapere quali altri nodi sono connessi. I topic hanno quindi un paradigma di comunicazione molti-a-uno o uno-a-molti.

- Servizi: I Servizi vengono usati per la comunicazione tra nodi quando sono richiesta dalla applicazioni interazioni che richiedono una richiesta e una risposta. Sono definiti come una coppia di messaggi: uno per la richiesta e uno per la risposta. Un nodo può

offrire un servizio che altri nodi usano semplicemente mandando un messaggio e aspettando la risposta. Hanno quindi un paradigma di comunicazione uno-a-uno.

```
# Get a plan from the current position to the goal Pose

# The start pose for the plan
geometry_msgs/PoseStamped start

# The final pose of the goal position
geometry_msgs/PoseStamped goal

# If the goal is obstructed, how many meters the planner can
# relax the constraint in x and y before failing.
float32 tolerance
---
nav_msgs/Path plan
```

Figura 2.1.2 Esempio di file .srv. I due messaggi di richiesta e risposta sono divisi dalla riga "---"

- Bags: Le bag sono un formato di storage e riproduzione dei messaggi di ROS. Nella fase di registrazione della bag verranno quindi salvati tutti i messaggi inviati fra i vari nodi, come possono essere quelli relativi alla navigazione, al laser o alle telecamere, che verranno salvati e potranno essere riprodotti in un secondo momento. Questo velocizza molto il processo di sviluppo e test degli algoritmi eliminando così di fatto il bisogno di fare continue prove sul robot fisico ma permettendo semplicemente di appoggiarsi al sistema delle bag.

Il sitema dei nomi ha una grande importanza nel Computational Graph Level. Ogni resource (Messaggio, Servizio, Nodo) ha il suo nome univoco. Hanno una struttura fortemente gerarchica, molto simile alla gestione dei file in un Sistema Operativo. In generale le risorse possono essere create e accedute solamente all'interno dello stesso namespaces. Le connessioni invece possono essere fatte tra namespace distinti.

Un' esempio di connessioni della rete del Computation Graph Level:
Possiamo avere un nodo che gestisce il Laser del robot e pubblica su /diago/laser messaggi di tipo sensor_msgs/LaserScan. Possiamo creare un nuovo nodo che si sottoscrive a questo topic a cui arrivano tutti i messaggi publicati su quel topic, se un nuovo nodo si sottoscrive a questo topic ad entrambi arriveranno esattamente gli stessi messaggi.

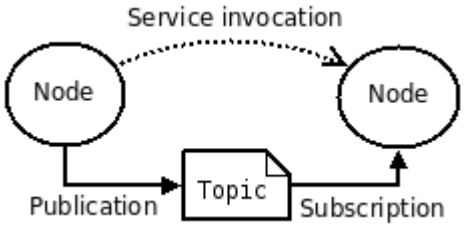

Figura 2.1.3 Comunicazione tra due nodi ROS. Viene messa in evidenza la differenza che c'è tra topic e service

Community Level:

E' la parte di ros che si occupa dello scambio di informazioni e progetti tra le varie comunità di sviluppatori.

- Distribuzioni: Sono collezioni di stack che rendono più facile l'installazione e l'aggiornamento del sistema ROS

- Repositories: Sono collezioni di codice che vengono pubblicate dai vari componenti della comunità i quali rilasciano parti di software utilizzabili da chiunque.

- ROS Wiki: E' la fonte di informazioni principale e forum di ROS nel quale ognuno può contribuire con nuovi articoli tutorial o correzioni.

Figura 2.1.4 Alcune delle ultime distribuzioni rilasciate

2.2 Strumenti Utilizzati

Il sistema ROS come già detto mette a disposizione degli sviluppatori un grande numero di packages e tool per lo sviluppo di robot, questi strumenti agevolano notevolmente il lavoro dei ricercatori.
Si possono trovare pacchetti per la navigazione, per la visualizzazione, per il mapping etc.
In questa sezione andremo ad analizzare gli strumenti più importanti che sono stati usati all'interno di questo progetto per costruire la mappa semantica.

Stage

Stage è un simulatore di uno spazio bidimensionale che include anche molti modelli di sensori come sonar, laser e camere così da poter essere il più possibile coerente con il mondo reale.[9]

Stage prende in input un file ".world" che descrive un "mondo". Questo file comprende tutte le informazioni di cui il simulatore ha bisogno per descrivere l'ambiente come l'immagine della mappa , gli ostacoli presenti e la posizione dei vari robot.
Il nodo stageros è sottoscritto al topic robot/cmd_vel in modo da poter comandare il robot attraverso joystick o tastiera inviandogli messaggi che contengo informazioni sulla velocità lineare e angolare da impartire al robot. [10]

Pubblica invece ai seguenti topic:

- odom : Fornisce l'odometria del robot nella mappa, questa dipende dal file .world passato per argomento
- base_scan : Dati dello scanner laser
- base_pose_ground_truth: Posizione del robot assoluta, indipendente dal file .world.
- image : Immagini della camera
- depth: Immagini della camera di profondità
- camera_info: Fornisce dati sulla calibrazione della camera

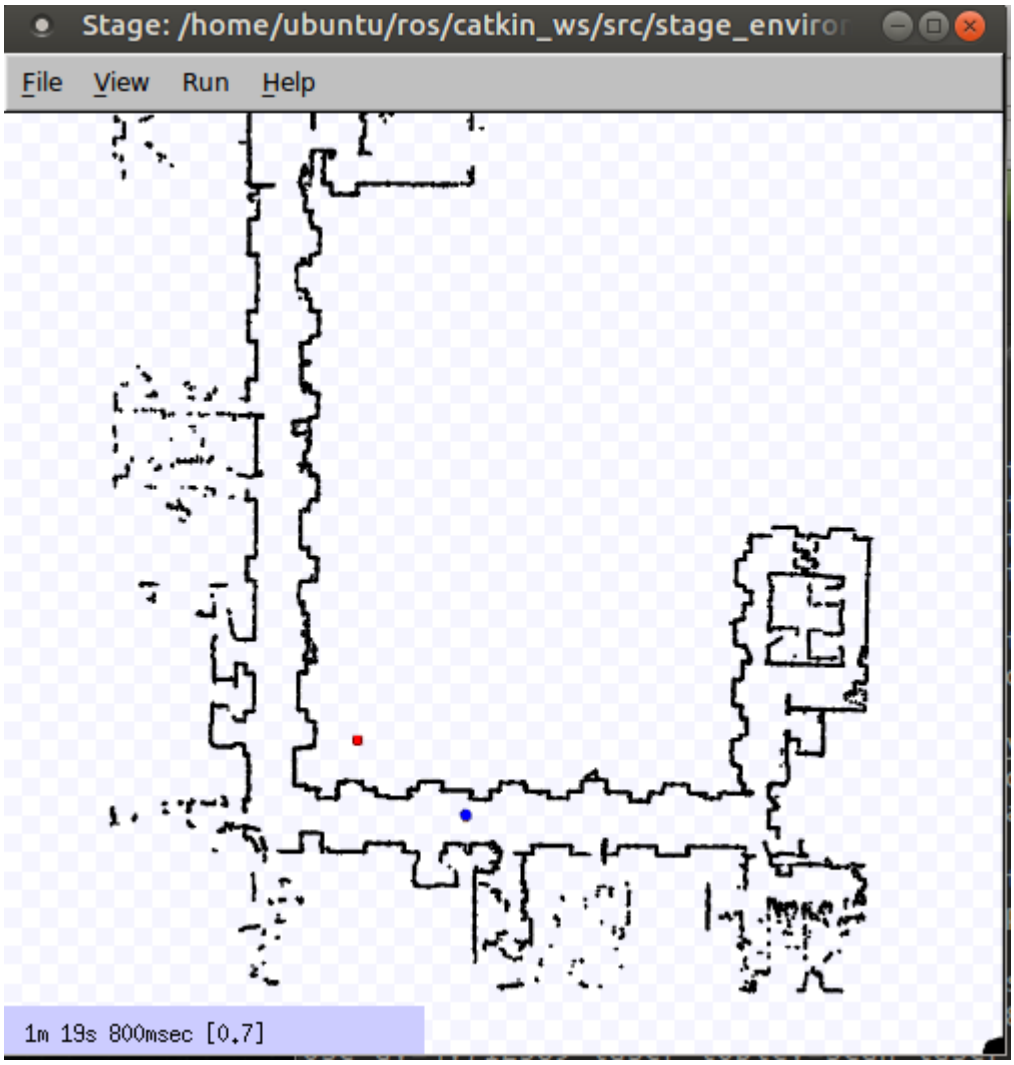

Figura 2.2.1 Simulazione in Stage. Il cerchio blu rappresenta il robot comandato dall'utente. Il quadrato rosso è invece un ostacolo che è possibile spostare con il mouse.

Roslaunch

Roslaunch è un tool che permette di eseguire più nodi ROS contemporaneamente sia localmente che da remoto e consentendo anche di settare i relativi parametri. Prende come argomento uno o più file XML con estensione .launch. Questi file contengono I dati sui nodi da lanciare e sui rispettivi parametri. [11]
Questo tool è estremamente comodo poiché in sistemi complessi è spesso richiesta l'esecuzione di un gran numero di nodi che sarebbe estremamente difficile da gestire altrimenti.

Il comando per eseguire un file .launch è:

$ roslaunch package_name file.launch

```
<launch>

  <arg name="world_file" default="AUTOGEN_DISB1_diago.world" />
  <arg name="base_frame" default="base_frame" />
  <arg name="laser_topic" default="scan" />
  <arg name="laser_frame" default="laser_frame" />

  <master auto="start"/>
  <param name="/use_sim_time" value="true"/>

  <node pkg="stage_environments" type="stageros_experimental" name="stageros" args="-u $(find stage_environments)/maps/$(arg world_file)"
respawn="false" required="true" output="screen">
    <param name="base_watchdog_timeout" value="0.2"/>
    <param name="base_frame" value="$(arg base_frame)"/>
    <param name="laser_topic" value="$(arg laser_topic)"/>
    <param name="laser_frame" value="$(arg laser_frame)"/>
  </node>

</launch>
```

Figura 2.2.2 Esempio di file .launch

Come si può vedere dalla Figura 2.2.2 il dati contenuti nel file sono compresi nel tag :

<launch> … </launch>.

Per creare un nodo invece si usa:

<node pkg = "..." type = "…" name = "..."/>

- pkg: Package in cui si trova il nodo che si vuole eseguire
- type: Nome del eseguibile del nodo.
- name: Permette di sovrascrivere il nome del nodo

Ci sono poi una serie di parametri e argomenti che possono essere passati nella creazione del nodo.

AMCL

AMCL è un localizzatore che ci permette di stimare la posizione del robot in una mappa bidimensionale. Prende in input i dati del sensore laser, una mappa e i messaggi della trasformata con cui calcola , attraverso il metodo Monte Carlo per la localizzazione, una stima della posizione del robot nell'ambiente.[12]

Move Base

Il package move_base è una parte fondamentale del navigation stack interno a ROS. Ci permette di dare in input delle coordinate verso le quali vogliamo far muovere il robot il quale vi si dirigerà autonomamente. Questo è eseguito tramite un sistema di Motion Plannig

che riesce ad elaborare il percorso migliore per spostarsi all'interno della mappa riuscendo anche ad evitare eventuali ostacoli presenti lungo il tragitto.

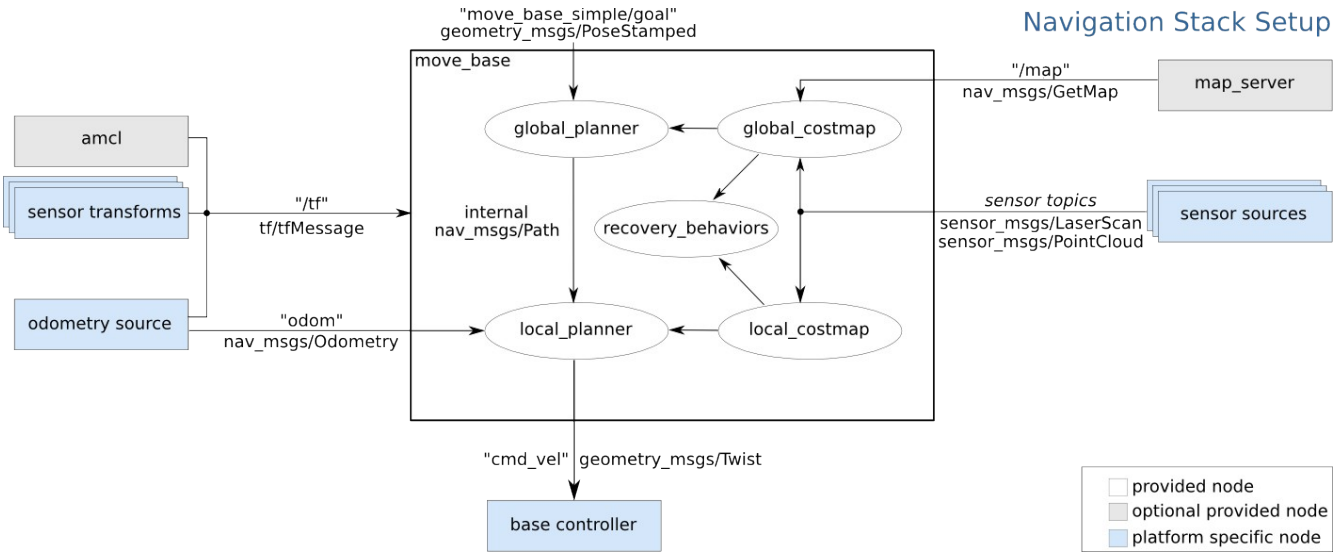

Figura 2.2.3 Grafico del nodo move_base e della sua interazione con il navigation stack.

Si può comunicare con il nodo move_base attraverso un SimpleActionClient, che dà la possibilità di tenere sotto controllo lo stato del goal permettendoci di interromperlo in qualsiasi momento. Pubblicando nel topic move_base/goal si può indicare una nuovo goal che potrà essere cancellato attraverso il topic move_base/cancel. Quando invece il robot arriva a destinazione move_base pubblica nel topic move_base/result l'avvenuta fine del goal.
Se non ci interessa tracciare l'andamento del goal è possibile comunicare con messaggi attraverso il topic move_base_simple/goal.
A questo topic vanno mandati messaggi del tipo std_msgs/PoseStamped i quali contengono al loro interno le coordinate del punto nel quale voglio far dirigere il robot.

RVIZ

Rviz acronimo di ROS Visualization è un visualizzatore 3D che è in grado di mostrare e calcolare dati provenienti dai sensori come ad esempio il laser o immagini di telecamere e tutte le altre informazioni inerenti il sistema di simulazione come possono essere mappe, robot e oggetti.

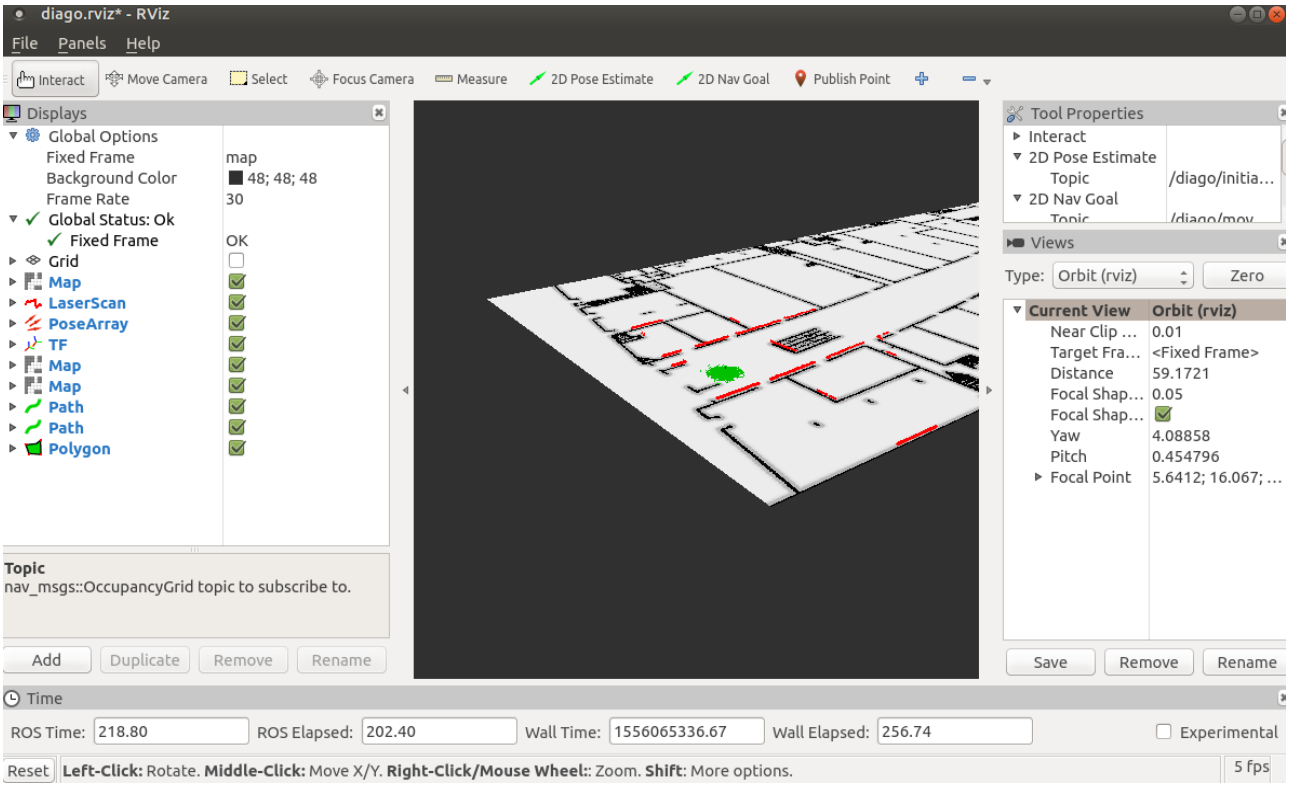

Figura 2.2.4 Interfaccia di Rviz

Come si può vedere nella Figura 2.2.4 nella colonna di sinistra abbiamo una serie di proprietà del simulatore tra cui il LaserScan che rappresenta appunto le proprietà del laser. Posso quindi sottoscrivermi al topic dedicato allo scambio di messaggi del sensore laser e grazie a questo comunicare i dati a Rviz.
Abbiamo poi TF che tiene traccia dello spostamento delle varie componenti fisiche del robot. Map invece è il gestore appunto della mappa il quale si sottoscrive al topic dedicato /map.

Rviz può essere usato insieme al navigation stack per muovere il robot all'interno di una mappa in modo autonomo.
Per fare questo è importante prima di tutto utilizzare un localizzatore, come AMCL, per stimare la posizione del robot nella mappa.
Per la navigazione posso usare il package move_base che mi permette di navigare autonomamente nella mappa, in particolare il topic in cui pubblica è move_base_simple/goal.
Per settare un goal basta cliccare sul pulsante 2D Nav Goal e poi sul punto desiderato sulla mappa.

LU4R

LU4R (*adaptive spoken Language Understanding chain For Robots* tool) è un tool per il riconoscimento e l'interpretazione del linguaggio parlato. E' scritto completamente in Java ed

è basato su una architettura Client/Server che lo rende indipendente dal robot specifico che lo utilizzerà: il Client è il robot utilizzatore mentre il Server è LU4R.
Il sistema LU4R riceve in input una trascrizione del comando e produce una interpretazione che è consistente con il sistema utilizzando una mappa semantica.[13]

La parte fondamentale di LU4R è il sistema SLU (Spoken Language Understanding), il suo compito è quello di tradurre le espressioni dell'utente in "semantic frames" , cioè in frammenti di frase che esprimono specifici concetti semantici di interesse per il sistema, per fare ciò LU4R si appoggia al sistema FrameNet [20] il quale fornisce le basi linguistiche e cognitive per l'interpretazione. Ad esempio nella frase

"take the book on the table"

"take" è un frame che rappresenta una azione, in particolare quelle di prendere,
"the book on the table" è invece il "THEME" cioè l'oggetto che riceve l'azione.
Questi frame servono quindi come connessione tra il linguaggio parlato ed il sistema che gestisce il robot.

Il sistema SLU si divide il 4 parti fondamentali:

- Analisi morfo-sintattica: E' l'analisi che viene realizzata su ogni espressione in input, fornisce informazioni morfologiche e sintattiche.
- Re-ranking: Vengono ordinate le varie ipotesi di trascrizione in base alla veridicità
- Action Detection: Vengono analizzati i vari frame generati e fra questi viene individuato quello che rappresenta l'azione da compiere.
- Argument Labeling: Assegna ad ogni frame la giusta etichetta come ad esempio THEME.

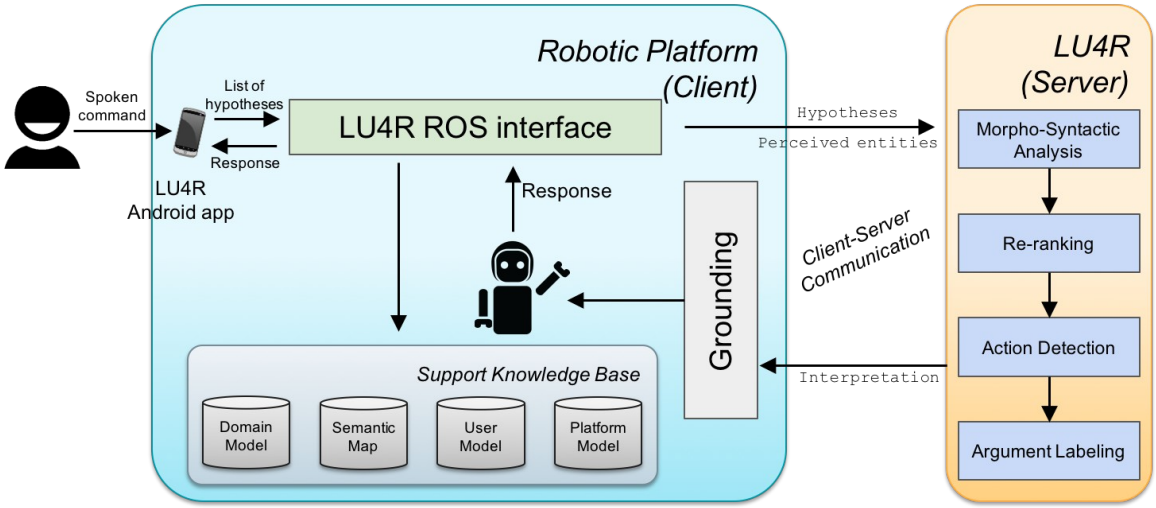

Figura 2.2.5 Rappresentazione architettura Client/Server di LU4R

Il sistema LU4R nel suo complesso si forma di 3 parti: il Server LU4R, la LU4R Android App e la LU4R ROS Interface.

- Il Server LU4R è il cuore del sistema ed è il responsabile dell'interpretazione degli input che gli arrivano dalla app Lu4r. E' la parte che contiene il sistema SLU.

- La LU4R Android App si occupa fondamentalmente della trascrizione delle espressioni vocali che vengono rilevate dal dispositivo attraverso un sistema ASR (Automatic Speech Recognition) il quale genera una o più ipotesi di trascrizione che verranno poi inviate al Server LU4R attraverso connessioni a protocollo TCP. E' disponibile anche un joystick che permette di comandare il robot.

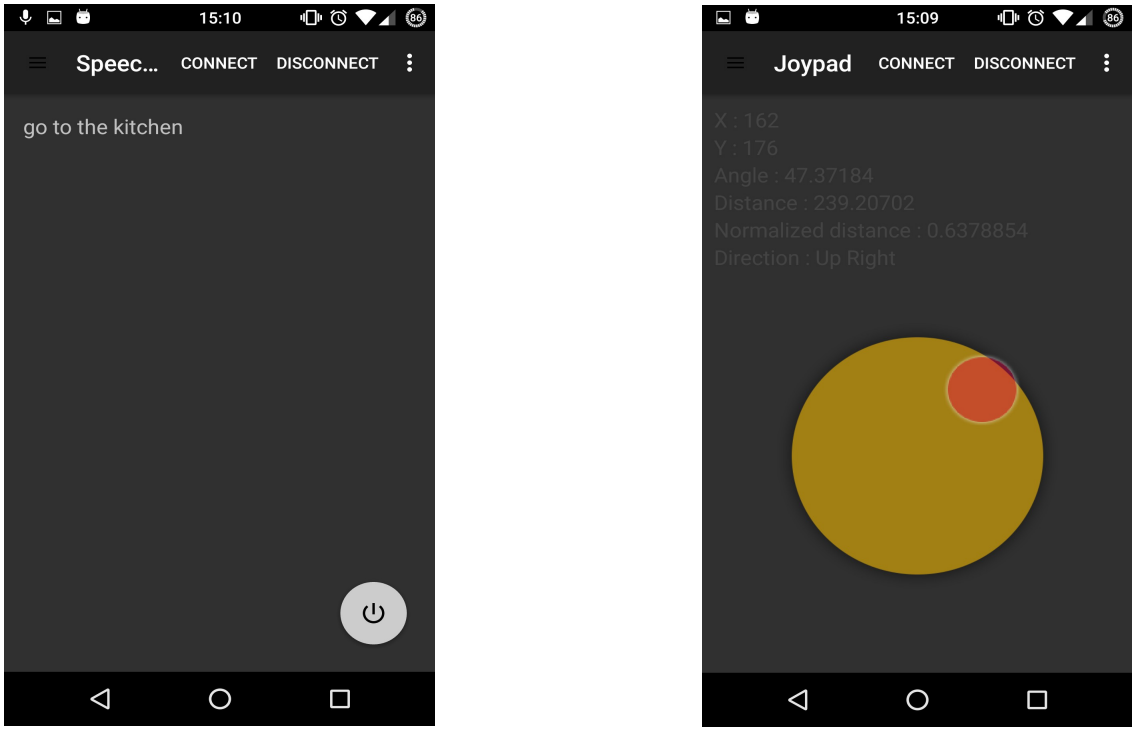

Figura 2.2.6 L'interfaccia dell'app LU4R: nel caso del riconoscimento vocale a sinistra e quello del joystick per la navigazione a destra

- LU4R ROS Interface è un insieme di nodi che permette l'implementazione di LU4R in ROS attraverso il sistema di publisher/subscriber e si divide a sua volta in 4 componenti.

1. android_interface: E' il server TCP al quale si connette la app Android e si occupa di estrarre le informazioni necessarie e tutte le ipotesi di trascrizione che gli arrivano dalla app e che vengono pubblicate dal nodo sul topic /speech_hypotesis, il sistema è

anche in grado di distinguere tra queste quella più verosimile che viene pubblicata come stringa nel topic /best_speech_hypothesis.
Si sottoscrive a /lu4r_response che contiene la risposta del Server Lu4r in termini di interpretazione semantica dei frame e a /dialogue_mager_responde che contiene invece la risposta generata dal gestore AIML.

2. lu4r_ros: E' il ponte tra il sistema LU4R e ROS, comunica attraverso HTTP con un server LU4R attivo

3. aiml_ros: E' il responsabile dell'interpretazione della espressione naturale e dell'elaborazione della risposta attraverso un file AIML (Artificial Intelligence Markup Language) il quale è in grado di instaurare un dialogo con l'utente.

4. framenet_ros_msgs: ponte con il sistema FrameNet

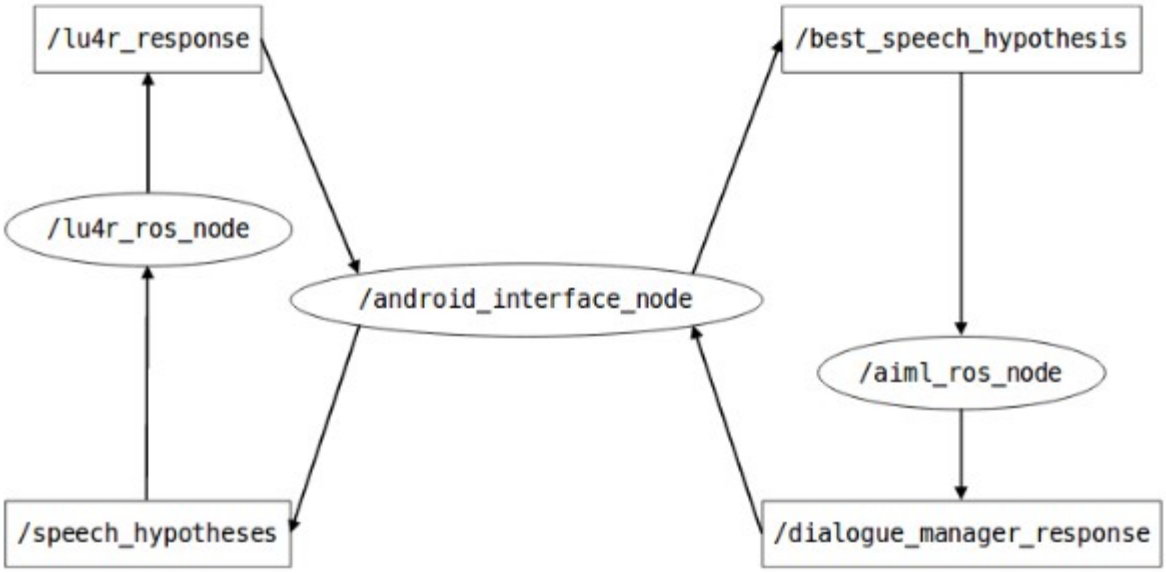

Figura 2.2.7 Grafo dell'interazione dei nodi all'interno di LU4R

La figura 2.2.7 spiega molto bene ciò che succede all'interno di LU4R. Il nodo android_interface è il centro del sistema. Ricevi i messaggi dalla app di riconoscimento vocale e li inoltra ai due nodi lu4r_ros e aiml_ros. Questi nodi gestiscono questi messaggi con le procedure indicate precedentemente e mandano i risultati di nuovo al nodo android_interface.

Capitolo 3

Mappe Semantiche

Mappa semantica è un termine utilizzato per indicare il processo che permette ad un robot di arricchire la mappa utilizzata per la navigazione con informazioni sull'ambiente in cui si trova come ad esempio dove si trova la cucina, l'ufficio etc. Questo ci permette di far muovere il robot in completa autonomia attraverso i vari punti di interesse della mappa utilizzando un sistema di navigazione. Questo aspetto è di grande interesse pratico basti pensare al caso di robot domestici che avendo una mappa semantica della casa possono svolgere compiti molto complessi ed essere molto più utili e facili da usare per gli utenti.

Per capire come realizzare una Mappa Semantica dobbiamo innanzitutto sapere come fa il robot ad orientarsi e muoversi nello spazio, dobbiamo quindi capire che cos'è e come funziona una mappa metrica, poi dobbiamo capire come aggiungere a questa mappa delle strutture che ci rendano possibile arricchirla di informazioni aggiuntive sull'ambiente circostante, costruire cioè lo scheletro della Mappa Semantica, infine dobbiamo capire quali sono i metodi per inserire le informazioni all'interno di queste strutture.

3.1 Mappa Metrica

La mappa metrica è la rappresentazione dell'ambiente fisico in cui il robot si trova. Questa mappa viene ricavata dalle informazioni date da sensori come laser o camere, le quali vengono elaborate attraverso vari metodi, ad esempio il metodo SLAM (Simultaneous Localization and Mapping). Questo metodo, come dice il nome, è in grado di creare una mappa metrica dello spazio man mano che viene esplorato attraverso i sensori laser e, allo stesso tempo, riesce a localizzare il robot nell'ambiente basandosi su degli oggetti fissi nel sistema come possono essere ad esempio dei muri.[15]
Nel sistema ROS un tool in grado di generare mappe metriche a partire da dati laser è GMapping il quale usa proprio il metodo SLAM.

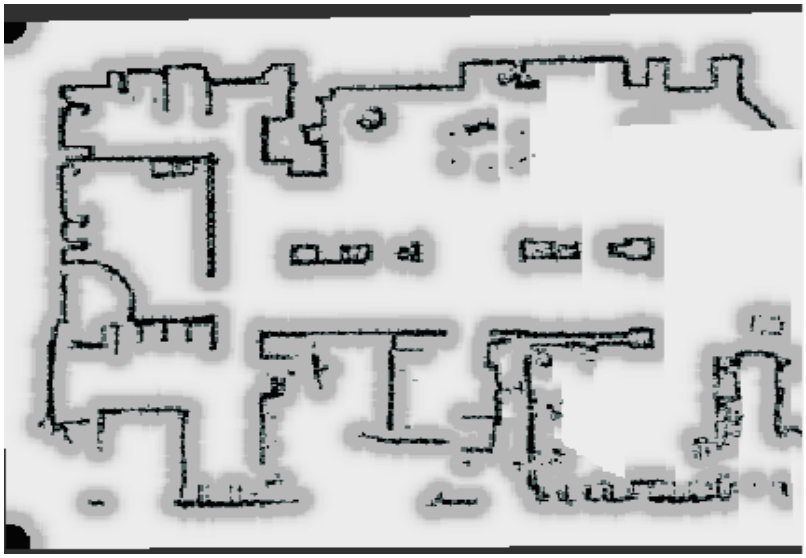

Figura 3.1.1 Mappa metrica creata con GMapper e visualizzata con RViz

3.2 Rappresentazione

Ci sono diversi metodi per realizzare una mappa semantica: la rappresentazione geometrica, la rappresentazione con griglia oppure quella attraverso la mappa topologica.

Nella rappresentazione geometrica vengono usate delle primitive geometriche per rappresentare e delimitare i luoghi di interesse della mappa.
Nella rappresentazione con griglia si divide la mappa in un insieme di celle di dimensioni uguali. Queste celle vengono etichettate una ad una in basandosi sul loro significato all'interno della mappa semantica. [14]

Sulla rappresentazione topologica ci fermeremo più approfonditamente in quanto è quella scelta per la realizzazione della Mappa Semantica in questo progetto.

Questa rappresentazione prova a gestire il problema utilizzando i grafi. Ogni nodo rappresenta un informazione in qualche modo significativa per il sistema, ad esempio può rappresentare una stanza, un oggetto che vogliamo inserire nella nostra mappa o semplicemente un punto di passaggio del robot. Gli archi invece rappresentano le connessioni esistenti tra i vari nodi, ci dicono quindi in che modo sono connessi fra loro gli oggetti dell'ambiente e come possono interagire tra loro. Si possono anche aggiungere informazioni agli archi, ad esempio dandogli un peso relativo alla distanza tra un nodo ed un altro a lui connesso, in modo tale da poter creare degli algoritmi che sappiano scegliere il percorso migliore per far muovere il robot da un nodo all'altro.

Formalizzando possiamo dire che la nostra mappa topologica è una coppia $T = (N , E)$ dove $N = \{n_1 , \ldots , n_n\}$ è un inseme di nodi $E = \{e_1 , \ldots , e_m\} \subseteq N \times N$ un inseme di archi. Un nodo è definito come $n_i = (pos, label)$ con pos = (x, y) e label un identificatore unico che definisce l'oggetto/luogo specifico, questo quindi definisce univocamente la sua posizione nella Mappa Metrica e il suo significato nella Mappa Semantica. Un arco $e_i = (src,dst)$ descrive il collegamento tra due nodi tale che src è la partenza e dst l'arrivo. I grafi possono essere sia orientati che non.[15]

Figura 3.2.1 Rappresentazione di Mappa Topologica insieme alla rispettiva Mappa Metrica

3.3 Identificazione di oggetti

A questo punto dobbiamo creare un metodo che ci permetta di riconoscere quali sono effettivamente i punti di interesse per la mappa semantica e quali no.

Il metodo sicuramente più immediato è quello di creare prima una mappa dell'ambiente e aggiungere in un secondo momento i punti di interesse ma questo è evidentemente poco utile ai fini pratici.

Un metodo più interessante è quello della Object Detection, cioè il riconoscimento autonomo da parte del robot, attraverso sensori come laser o camere, di oggetti di particolare rilievo. Dal riconoscimento di determinati oggetti possiamo ottenere molte informazioni utili sulla

semantica dell'ambiente, ad esempio individuare un microonde ci può far concludere con buona approssiamazione che ci troviamo in cucina.
Una libreria che ci permette di implementare questo genere di algoritmi è OpenCV.
Abbiamo poi bisogno anche di un sistema per capire cosa è una stanza e quando il robot vi entra. Un sistema (proposto da Jensfelt[16]) usa il laser per capire quando il robot passa attraverso una porta. Se il robot ha attraversato una porta vuol dire che ha cambiato stanza.

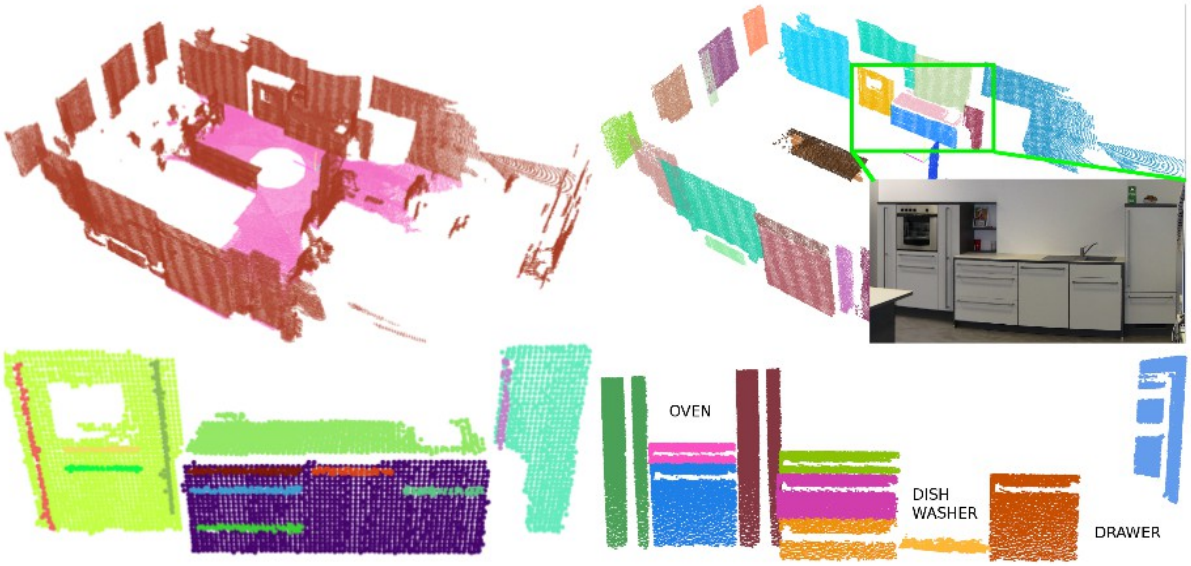

Figura 3.3.1 Esempio di Object Detection ([18])

Infine un altro metodo è quello dell'interazione diretta tra utente e robot, è il metodo scelto per questo progetto.
L'utente ha quindi la possibilità di creare la Mappa Semantica interagendo vocalmente con il robot. Ad esempio dicendo "Questa è la cucina" il robot, attraverso un sistema di interpretazione del linguaggio come LU4R, elabora l'informazione. In questo caso viene estratta "cucina" come informazione di rilievo ai fini della Mappa Semantica del sistema e il robot lo aggiunge alla Mappa.

Capitolo 4

Costruzione di una Mappa Semantica

In questo capitolo verrà descritto nello specifico il metodo utilizzato per la costruzione della Mappa Semantica in questo progetto. Partendo dalla struttura scelta per la rappresentazione e descrivendo poi in che modo si sono messi in connessione i vari strumenti di ROS per il riconoscimento vocale, la localizzazione e la navigazione del robot.
Verranno poi descritti gli esperimenti condotti con il robot nei simulatori interni a ROS.

4.1 Realizzazione

Per questo progetto si è creato un unico nodo ROS dal nome semantic nel package semantic che gestisce tutta la computazione.
Si è voluta implementare una realizzazione di mappa semantica attraverso un grafo con lista di adiacenza non orientato. Questo grafo viene creato "on-line" al momento dell'esplorazione dell'ambiente da parte del robot.

```
struct graph_prop {
  int n_vertices;
  int n_edges;
}graph_prop;

struct graph{
    linked_list * nodes;
  struct graph_prop* properties;
};
```

```
struct graph_node_prop{
  const char* label;
  int is_something;
  float x,y;
};

struct graph_node{
    void * value;
    linked_list * out_edges;
    STATUS state;
};
```

Figura 4.1.1 Strutture usate per la rappresentazione dei grafi. Come si può vedere la struttura grafo mantiene una lista di tutti i nodi presenti. A sua volta ogni singolo nodo contiene una lista di tutti i nodi a cui è collegato.

Il grafo registra in particolare per ogni nodo la sua posizione (x,y), una stringa che identifica il nodo ed un valore is_something che segnala l'importanza o meno del nodo nella mappa semantica.

La prima parte dell'attività del nodo è l'esplorazione. Il robot viene quindi comandato attraverso joystic/tastiera ed è costantemente aggiornato riguardo la sua posizione nella mappa dal topic /diago/amcl_pose del localizzatore AMCL.

Comincia quindi la creazione del grafo: un nuovo nodo è creato non appena il sistema rileva che il robot è a più di 0,5 metri di distanza dal nodo più vicino e a questo viene collegato con un arco il nuovo nodo. Si viene così a formare un grafo che segue il robot nella sua esplorazione e che è in grado di gestire incroci e deviazioni. Si creerà a questo punto una struttura del tutto simile di quella rappresentata nella Figura 3.2.1.

A questo punto si può cominciare ad immettere le informazioni della Mappa Semantica.
Il nodo è collegato al sistema LU4R attraverso il topic /best_speech_hypothesis grazie al quale riceve la stringa dell'interpretazione di ciò che l'utente ha detto, questa viene elaborata e se è una frase del tipo "Questa è la cucina" oppure " Siamo in cucina" il sistema setta la label del nodo più vicino al punto dove si trova attualmente a "cucina".

A questo punto ci troviamo in una situazione simile a quella di Figura 4.1.2.

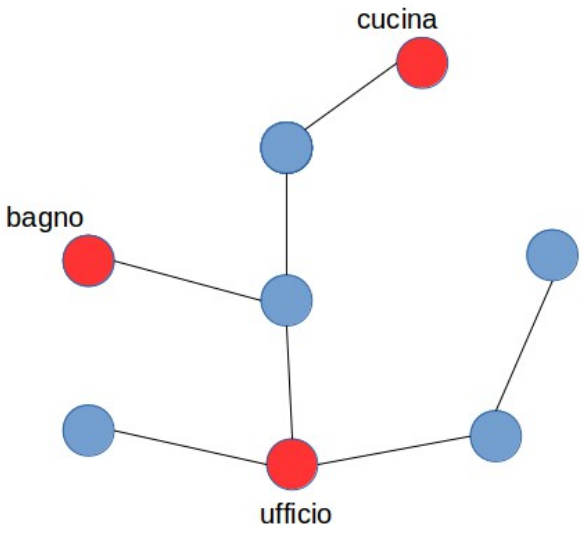

Figura 4.1.2 Possibile mappa topologica generata dal sistema. Si noti che gli archi sono tutti equivalenti in quanto approssimativamente tutti rappresentanti una distanza di 0.5 metri

Si può adesso procedere con la parte di navigazione.
Dopo aver esplorato la mappa si può dare al robot un comando del tipo "Vai in cucina". Così facendo il sistema interpreterà il messaggio e cercherà tra i nodi presenti attualmente nel grafo quello etichettato con "cucina" designandolo come destinazione. Successivamente troverà il nodo più vicino alla posizione del robot e lo sceglierà come nodo di partenza.
A questo punto partirà un algoritmo DFS(Depth First Search) il quale troverà il path tra il nodo di partenza e quello di destinazione restituendo la lista dei nodi attraversati tra i due. Questo path, essendo gli archi approssimativamente tutti uguali, sarà anche lo shortest path.

Una volta ottenuta la lista dei nodi da attraversare il robot comincia a pubblicare messaggi con all'interno le coordinate di ogni nodo sul topic che permette la navigazione: /diago/move_base_simple/goal. Il sistema gestisce anche l'arrivo dei messaggi di fine dei goal di move_base attraverso il topic /diago/move_base/result per coordinare l'invio dei goal successivi.

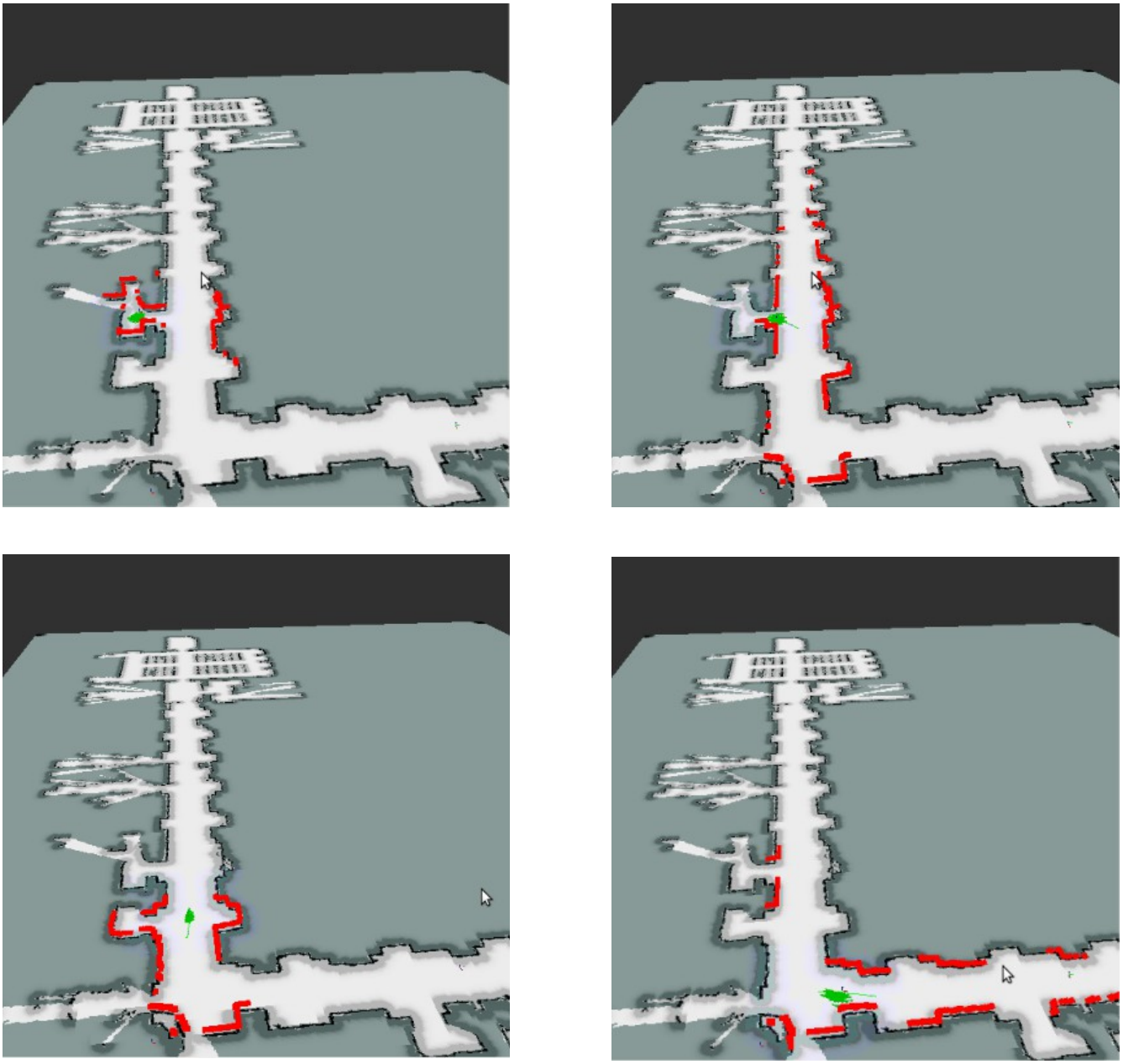

Figura 4.1.3 Queste 4 immagine evidenziano come avviene il processo di navigazione.
Nella 1 immagine il robot è dentro una stanza, la nuvola di punti rappresenta le possibili posizioni stimate dal localizzatore. Nelle immagini successive si possono vedere le linee verdi che rappresentano i goal inviati al robot. Ogni goal lo fa muovere verso la posizione di un nodo, in questo modo riesce ad arrivare alla destinazione attraversando tutti i i nodi che li collegano

Infine c'è la parte dedicata all'interazione vocale con l'utente: il gestore AIML.
Si è creato un file .aiml in grado di gestire una piccola conversazione, come salutare, chiedere e ricordare il nome. Il sistema è anche in grado di rispondere correttamente alle frasi del tipo "Questa è la cucina", " Siamo in cucina", "Vai in cucina" con frasi del tipo "OK [NOME], siamo in cucina" oppure "OK [NOME] , vado in cucina".

```
<category>
    <pattern> SIAMO IN *</pattern>
    <template>
        <think>
            <set name="posto"><star/></set>
        </think>
        [SAY] OK <get name="user"/> Siamo in <get name="posto"/>
    </template>
</category>
```

Figura 4.1.4 Estratto del file .aiml utilizzato.

I file .aiml sono formati da un insieme di tag i più importanti dei quali sono:

- category: Rappresenta l'unità di conoscenza
- pattern: E' il riferimento con cui il sistema confronta ciò che ha ricevuto dalla app di riconoscimento vocale.
- template: Contiene ciò che viene eseguito quando il contenuto di <pattern> corrisponde con quello ricevuto dalla app

Ci sono poi i tag <set> e <get> che servono rispettivamente per registrare e richiamare variabili, le quali vengono salvate con il nome contenuto in name="..." e possono essere richiamate in un qualunque punto del file. In questo caso la variabile "user" contiene il nome dell'utente e la variabile "posto" l'indicazione data dall'utente stesso.

Di seguito è rappresentato il grafo dell'interazione dei singoli nodi ROS nel sistema complessivo, ottenuto con il tool rqt_graph.

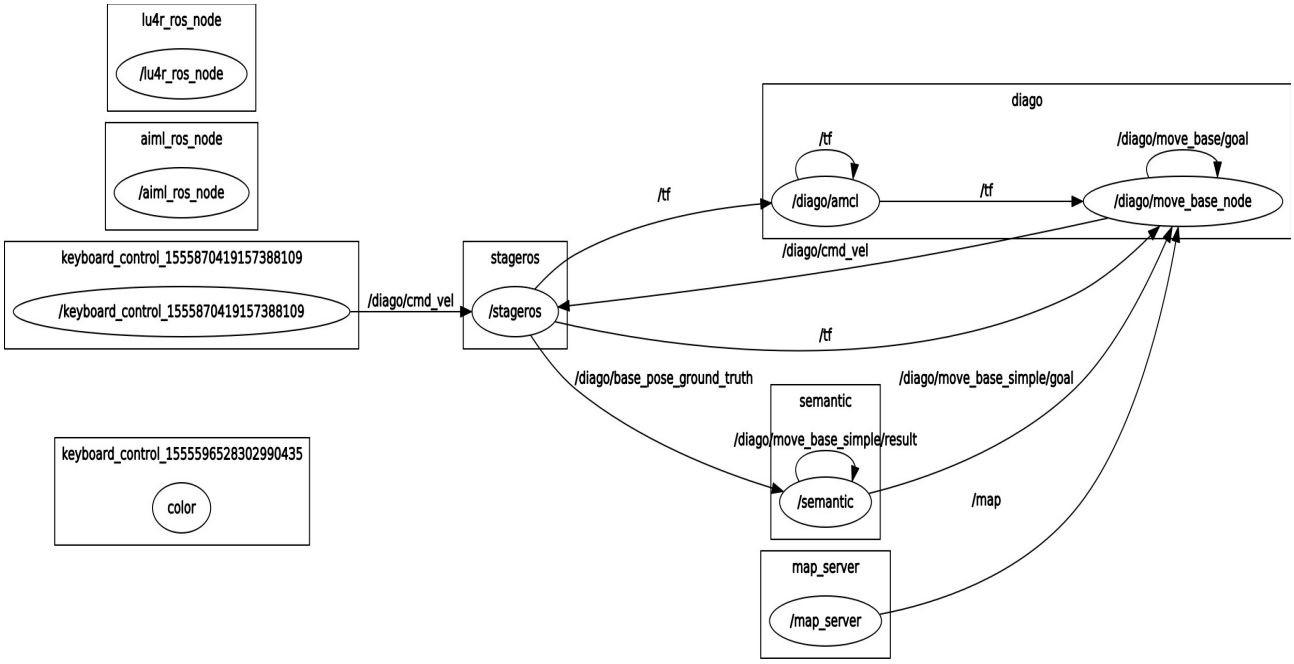

Figura 4.1.5 Grafo dell'interazione dei nodi

4.2 Simulazione

I test sono stati eseguiti attraverso il sistema di simulazione fornito da ROS. Sono state quindi usate delle mappe create in precedenza con GMapping le quali sono state inserite in Stage per simulare l'ambiente reale, si sono usate più mappe fornite dall'Università. Si è cominciato quindi esplorando l'ambiente simulato comandando il robot da tastiera e si è visto come il sistema riusciva a creare un grafo dell'area esplorata in qualunque tipo di mappa e in modo del tutto parallelo alla navigazione guidata. Per la localizzazione, come spiegato precedentemente, è stato usato il localizzatore AMCL mentre il pacchetto move_base è stato utilizzato per la navigazione autonoma. Entrambe questi sistemi si sono rivelati estremamente affidabili e precisi nell'esecuzione, il robot è stato quindi in grado di orientarsi perfettamente in tutti i tipi di mappe utilizzati senza riscontrare problemi. Rviz è stato utilizzato per una più completa visione dell'ambiente in quanto fornisce maggiori informazioni come ad esempio è in grado di visualizzare i goal che vengono dati a move_base.

In particolare si è riscontrato come il robot riesca ad orientarsi in ogni tipo di mappa creando la Mappa Semantica nello stesso momento in cui avviene l'esplorazione, in questo modo

l'utente ha un interazione immediata con il robot, senza nessuna differenza sostanziale tra esplorazione e navigazione, i due stati possono essere scambiati in qualsiasi momento: Quando l'utente dà un comando del tipo "Siamo in cucina" il robot salva la sua posizione che può esse richiamata in qualsiasi momento con comandi del tipo "Vai in cucina". Questa simultaneità tra esplorazione e navigazione potrebbe essere molto utile in applicazioni reali come robot domestici consentendo di rendere molto più facile il compito dell'utente di "insegnare" al robot la struttura della mappa.
Una problematica di questa implementazione è però che il robot ripercorre solamente la strada segnata dai nodi posizionati in coordinate già esplorate, questo può essere poco conveniente in un primo momento ma esplorando più approfonditamente la mappa il grafo diventerà più fitto e connesso fino a ricoprire l'intero ambiente permettendo così una più agile mobilità del robot.

Si è poi proseguito a testare il sistema in un ambiente reale. Per fare questo si è utilizzato il robot Pioneer, sviluppato nei laboratori della Sapienza.
Il robot Pioneer è formato da una base, con 4 ruote mobili e da un sensore laser Hokuyo, utilizzato per orientarsi nell'ambiente.

Inizialmente si è proceduto collegando il computer al robot tramite USB e utilizzando il pacchetto srrg_pioneer contenente il nodo pioneer_robot che permette di interfacciarsi con il robot. Un joystick è stato collegato per comandare il robot attraverso la pubblicazione di messaggi nel topic /cmd_vel al quale il nodo del Pioneer è sottoscritto. Questo nodo si occupa di interpretare questi messaggi e di imporre al robot una velocità relativa al comando ricevuto. Si è avuto cura in questa fase di settare correttamente i topic che dialogavano tra robot e computer, cioè quelli relativi ai comandi di velocità del robot, alla posizione stimata dal localizzatore e quelli del sensore laser, per permettere al sistema di funzionare correttamente.
Si sono infine avviati i nodi relativi al sistema LU4R.

A questo punto, dopo aver completato la configurazione iniziale del sistema, si è potuto testare il pacchetto semantic all'interno del robot Pioneer.
Per prima cosa è stata creata una Mappa Metrica dell'ambiente creando dapprima una bag e ricavando da questa, attraverso il tool Gmapping, un file contenente la mappa. Questa è stata poi passata al localizzatore AMCL permettendogli in questo modo di stimare la posizione del robot all'interno dell'ambiente, utilizzando i sensori laser del Pioneer. Dopo questa procedura si è stato in possesso di tutte le informazioni necessarie per eseguire il nodo semantic.
Il robot ha quindi esplorato l'ambiente circostante per creare il grafo e ha registrato all'interno dei nodi la loro posizione grazie alle informazioni fornite dal localizzatore. Questo grafo è stato poi riempito con le informazioni ricevute tramite il sistema di riconoscimento vocale creando così la Mappa Semantica dell'ambiente, esattamente come avveniva all'interno del simulatore Stage.

La parte relativa alla navigazione autonoma è stata eseguita con il pacchetto move_base, questo ci ha permesso di muovere il robot nella mappa facendogli seguire i nodi creati precedentemente e di raggiungere la destinazione desiderata.

Si è potuto evidenziare quindi come ROS sia uno strumento estremamente utile a chi sviluppa robot in quanto ci ha permesso di creare un ambiente virtuale del tutto simile all'effettivo ambiente reale. Abbiamo sviluppato il sistema inizialmente all'interno di un simulatore per portarlo poi sul robot Pioneer, per fare questo è bastato collegare il robot al computer e settare i relativi topic, non sono state necessarie modifiche nei singoli nodi. Il sistema nel suo complesso è risultato del tutto simile a quello simulato ed è stato in grado di effettuare tutte le procedure indicate nel paragrafo 4.1.

Capitolo 5

Conclusioni

Nello svolgimento di questa Tesi, sviluppata nell'ambito della Laurea in Ingegneria Informatica dell'Università la Sapienza, si è affrontato uno degli argomenti più studiati della robotica odierna: l'esplorazione dei ambienti e la costruzione di Mappe Semantiche.
Si è inizialmente spiegato come il sistema ROS sia di fondamentale importanza per lo sviluppo di robot e qual' è la sua struttura. Si è visto nel dettaglio come è strutturato e quali sono stati i tool disponibili in ROS che sono stati usati nello svolgimento di questo progetto. Successivamente si è specificato cosa sia una Mappa Semantica e di cosa abbiamo bisogno per realizzarla, descrivendo quindi Mappe Metriche, i vari tipi di strutture per costruire Mappe Semantiche e i metodi usati per riconoscere gli oggetti presenti nella mappa e la sua struttura. Infine è stato spiegato nel dettaglio come si è proceduto per la realizzazione del progetto e gli esperimenti effettuati. In questi esperimenti il robot ha dimostrato di essere in grado di creare una Mappa Semantica dell'ambiente circostante qualunque esso sia, basandosi sulla Mappa Metrica ad esso associato.

Questa Tesi pone le basi per lo sviluppo di Mappe Semantiche ma può essere ampliata notevolmente in sviluppi futuri. Una modifica che potrebbe essere apportata al sistema sarebbe quella di affiancare al già presente sistema di riconoscimento vocale su cui si basa attualmente, un sistema di Object detection, cioè di riconoscimento visivo, del tutto simile a quello mostrato nella sezione 3.3. Questo sistema permetterebbe una interazione con l'ambiente ancora migliore rendendo il sistema indipendente anche dalla presenza dell'utente in quanto sarebbe in grado di riconoscere ed etichettare oggetti e stanze della mappa in modo del tutto autonomo. Un'ulteriore modifica sarebbe quello di staccare il sistema dalla dipendenza del grafo rendendolo in grado di muoversi liberamente all'interno dell'intera mappa e trovare percorsi all'interno di essa senza alcuna restrizione.

I possibili sviluppi di tecnologie simili a questa sono vasti e ancora ampiamente da scoprire. La ricerca in questo ambito è sempre più attiva e prolifica, Google sta sviluppando i suoi prototipi di macchine autonome così come anche la General Motors. Affianco a queste ci sono sviluppi industriali e domestici che seguono il trend in forte espansione. È quindi un dato di fatto che lo sviluppo tecnologie di questo tipo, basate sullo sviluppo di sistemi in grado di rendere i robot autonomi e di svolgere compiti sempre più complessi, sia una delle grandi sfide della robotica moderna.

Bibliografia

[1] Andrea Tartaglia, “Autoblog”, [Online]. Avaible: https://www.autoblog.it/post/965623/guida-autonoma-waymo-google-trasforma-auto-normali-in-driveless-car

[2] Wikipedia, “Autovettura Autonoma”, [Online]. Avaible: https://it.wikipedia.org/wiki/Autovettura_autonoma.

[3] “Hurolife”, [Online]. Avaible: https://www.hurolife.it/robot-mobili-autonomi-e-a-guida-autonoma-la-differenza-tra-amr-e-agv/)

[4] Mobile Industrial Robots, “AGV vs. AMR” [Online]. Avaible: https://www.mobile-industrial-robots.com/en/resources/whitepapers/agv-vs-amr-whats-the-difference/

[5] Wikipedia, ”Robot Operating System” [Online]. Avaible: https://it.wikipedia.org/wiki/Robot_Operating_System

[6]“ROS.org”, [Online].: https://www.ros.org/history/

[7] W. Garage, “ROS platform”, April 2013. [Online]. Available: http://www.willowgarage.com/pages/ software/ros-platform.

[8] “ROS.org”, [Online]. Available: http://wiki.ros.org/rosbuild/ROS/Concepts.

[9] “Player Project”, [Online]. Avaible: http://playerstage.sourceforge.net/

[10] “ROS.org”, [Online]. Avaible: http://wiki.ros.org/stage_ros

[11] Avaible: http://wiki.ros.org/roslaunch

[12] Avaible: http://wiki.ros.org/amcl

[13] R. Basili, E. Bastianelli, G. Castellucci, D. Croce, D. Nardi e A. Vanzo, “LU4R Project adaptive spoken Language Understanding For Robots”, [Online]. Available: http://sag.art.uniroma2.it/lu4r.html.

[14] E. Bastianelli, D. D. Bloisi, R. Capobianco, F. Cossu, G. Gemignani, L. Iocchi, D. Nardi, On-line Semantic Mapping,Sapienza University of Rome, Italy.

[15] Rizwan Aslam Butt , Syed M Usman Ali, Semantic Mapping and Motion Planning with Turtlebot Roomba , Karachi, Pakistan 2013.

[16] Xavier Gallart Del Burgo, "Semantic Mapping in ROS", Master Thesis. Stockolm, Sweden 2013.

[17] P. Jensfelt, Approaches to Mobile Robot Localization in Indoor Environments, Stockholm, Sweden: PhD thesis, 2001.

[18] Nico Blodow, Lucian Cosmin Goron, Zoltan-Csaba Marton, Dejan Pangercic, "Autonomous Semantic Mapping for Robots Performing Everyday Manipulation Tasks in Kitchen Environments", Munchen.

[19] Joshua S. Lum ," UTILIZING ROBOT OPERATING SYSTEM (ROS) IN ROBOT VISION AND CONTROL ", California 2015.

[20] Berkeley, "FrameNet",[Online]. Avaible: https://framenet.icsi.berkeley.edu/fndrupal/

[21] MARRtino, [Online]. Avaible: https://www.marrtino.org/home .

Capitolo 1

Introduzione

Questa tesi, sviluppata nell'ambito del Laboratorio di Intelligenza Artificiale e Grafica Interattiva del corso di Laurea in Ingegneria Informatica, ha lo scopo di illustrare come sia possibile per un robot orientarsi e muoversi in un ambiente a lui inizialmente sconosciuto e come estrapolare, all'interno di questo ambiente, delle informazioni utili alla sua comprensione, come potrebbero essere la disposizione nello spazio delle varie stanze e degli oggetti presenti, che permettono di costruire la cosiddetta mappa semantica dell'ambiente.

Si vuole sottolineare quindi, come lo sviluppo di soluzioni simili, in ambito commerciale ed industriale, sia di estrema rilevanza al giorno d'oggi in quanto permetterebbe di semplificare molto aspetti estremamente rilevanti per la nostra società, da quelli legati ad un ambiente casalingo a quelli delle grande industrie. Si pensi ad esempio ai robot che sono in grado di aiutare persone anziane e chiunque abbia difficoltà. Questi tipi di robot sono già molto usati e poggiano proprio su questa tecnologia per mappare ad esempio le stanze della casa ed essere così molto più utili e facili da utilizzare per l'utente.

Un'altra applicazione che si basa sull'utilizzo di mappe semantiche e sta conoscendo un incredibile avanzamento tecnologico e commerciale negli ultimi anni è quello dei veicoli autonomi.

Google, attraverso la sua azienda Waymo, sta infatti investendo molte risorse nello sviluppo di automobili in grado di muoversi in qualunque situazione di traffico e in qualunque ambiente in modo completamente autonomo, escludendo quindi ogni interazione necessaria da parte dell'utente[1]. Questi veicoli sono in grado di esplorare ed analizzare l'ambiente circostante con tecniche come radar, GPS e visione artificiale. A queste tecnologie si affiancano sistemi di controllo molto avanzati, basati sull'uso di sensori presenti nel veicolo. Questi sistemi permettono di interpretare le informazioni ricevute per individuare i percorsi migliori in cui muoversi e anche la presenza di ostacoli. La tecnologia usata da questi veicoli per la localizzazione è proprio la SLAM (Simultaneous Localization And Mapping) che verrà approfondita nel paragrafo 3.1, i sistemi più avanzati però utilizzano una sua variante (SLAM con DATMO) che riesce a mappare e localizzare anche oggetti in movimento.[2]

Infine un'ultima applicazione potrebbe essere quella dell'uso interno di robot industriali. Questi robot vengono utilizzati, appunto, all'interno di grandi aziende per semplificare lavori

che sarebbero estremamente pesanti ed usuranti per i dipendenti, come ad esempio il trasporto di materiale pesante.[3]

I robot ti questo tipo fino ad ora utilizzati sono stati del tipo AGV (Veicolo a Guida Automatica), i quali però avevano bisogno di una struttura fisica fissa per lavorare, come dei fili di acciaio lungo i quali muoversi, oppure sensori nel pavimento per permettere l'orientamento nello spazio, limitando così notevolmente le applicazioni di questa tecnologia.

Si sta quindi facendo molta ricerca su robot di tipo AMR (Robot Mobili Autonomi) che invece implementano sistemi che li rendono indipendenti da strutture esterne, potendo essere così utilizzati in qualunque situazione con estrema flessibilità.[4]

E' proprio su questi aspetti che si fonda questa tesi, sulla possibilità di creare delle mappe semantiche che possano servire allo sviluppo di sistemi simili a quelli descritti e sulla loro navigazione autonoma, entrami ambiti che avranno nell'immediato futuro un enorme espansione.

Nel Capitolo 2 verrà presentato ROS (Robot Operating System), un ambiente di sviluppo software che rende molto più facile lo sviluppo di robot di quanto non fosse in precedenza, fornendo una grande varietà di sistemi e tool per la robotica. Verranno anche presentati nel dettaglio i vari strumenti messi a disposizione dal sistema che sono stati utilizzati per lo sviluppo di questa tesi. E' il sistema sul quale è stata costruita questa tesi.

Nel Capitolo 3 verrà analizzato cose vuol dire e come si può realizzare una mappa semantica, partendo dalla mappa metrica per poi analizzare i vari metodi di creazione e sviluppo di una mappa semantica.

Nel Capitolo 4 invece, si avrà modo di spiegare nel dettaglio quale è stato l'approccio utilizzato nella presente tesi per affrontare questo genere di problemi, quindi come sono stati utilizzati gli strumenti di ROS all'interno del progetto e quali sono state le sperimentazioni a cui è stato sottoposto il sistema creato.

Nell'ultimo Capitolo vengono tratte le conclusioni sul progetto, ricapitolando cosa è stato analizzato e sottolineando le potenzialità dell'argomento, fornendo dei possibili ampliamenti a questo genere di sistemi che potrebbero essere sviluppati in futuro.

Capitolo 2

ROS (Robot Operating System)

ROS (Robot Operating System) è un insieme di framework per lo sviluppo e la programmazione di robot basato su Linux e distribuito con licenza open-source (BSD). I linguaggi attualmente supportati sono C/C++, Python e Lisp, mentre Java e Lua sono in via di sviluppo.[5]

Nasce nel 2007 sulla base di alcune ricerche della Stanford University sulla robotica e sull'intelligenza artificiale, come Stanford AI Robot (STAIR) e il Personal Robots (PR) program, che vennero ampliate dalla compagnia Willow Garage portando così alla creazione di ROS.[6]
Lo scopo principale della Willow Garage era quello di creare un sistema che permettesse a chi sviluppava robot di poter risparmiare molto tempo, evitando di dover ripartire dalla progettazione di base per ogni nuovo modello. Il sistema ROS permette infatti di avere una solida base sulla quale sviluppare il proprio robot, formata da un grande numero di tool diversi che possono essere messi in connessione tra loro, per poter essere poi utilizzati nel proprio progetto, il che rende il sistema estremamente flessibile e condivisibile. Infatti questo concetto di modularità è ciò che più di tutto ha permesso a ROS di guadagnare popolarità ed essere oggi usato in moltissime Università e centri di ricerca per lo sviluppo della robotica. Inoltre, essendo completamente open-source, ogni gruppo di ricerca può avere la sua repository ROS e mantenere il completo controllo su di essa, creando così un folto ecosistema di ricercatori, ad oggi decine di migliaia, che scambiano idee e progetti, dai più semplici a grandi sistemi industriali, permettendo così un massiccio avanzamento della robotica.[7]

ROS fornisce tutti gli strumenti di un sistema operativo come astrazione dell'hardware, comunicazione tra processi attraverso invio di messaggi, gestione delle applicazioni, librerie e strumenti per la scrittura, compilazione ed esecuzione del codice e controllo dei device di basso livello.

2.1 Concetti Fondamentali in ROS

ROS è diviso in 3 livelli concettuali i quali gestiscono rispettivamente: l'organizzazione dei file su disco, la rete peer-to-peer dei nodi e infine lo scambio di software tra i gruppi di sviluppatori.
Di seguito vengono descritti nel dettaglio.[8]

File system Level:

I pacchetti sono il cuore del file system di ROS e possono contenere un eseguibile (nodo), una libreria di ROS, un dataset o anche file di configurazione e per questo rappresentano l'unità più piccola del sistema ROS. Il loro scopo è quello di rendere la struttura del sistema più modulare possibile, potendo essere riutilizzati per diversi progetti e rilasciati per la comunità degli sviluppatori.
Ogni package ha un package manifest, cioè un file .xml che contiene i metadata del pacchetto come ad esempio nome, versione, licenza ma anche le dipendenze da altri pacchetti.

Ci sono poi i Packages Resources Names che sono usati per semplificare il riferimento ai tipi di dato sul disco. Sono formati semplicemente dal nome del package dove si trova la resource seguito dal nome della resource stessa, ad esempio il Package Resource Name del tipo di messaggio (resource) String del package "std_msgs" sarà "std_msgs/String".

Gli Stack sono collezioni di package che collaborano per un unica funzionalità, come può essere ad esempio il "navigation stack", che è l'insieme di package che si occupa della navigazione del robot. Anche gli stack hanno il loro manifesto.

Infine ci sono i Service types e Message types che definiscono la struttura rispettivamente dei Servizi e dei Messaggi.

Computational Graph Level:

Questo livello rappresenta la rete peer-to-peer dei processi ROS che si scambiano informazioni.
I concetti principali di questo livello sono implementati nello stack ros_comm e sono:

- Nodi: I nodi sono i processi ROS che eseguono operazioni che quindi, partendo dai dati ricevuti dai vari topic e server, sono in grado di eseguire determinate azioni. Data la forte modularità del sistema ROS, ogni nodo si occupa di una specifica operazione del sistema generale che può comprendere molti nodi: ad esempio in un robot un nodo

può gestire la navigazione, un nodo la visualizzazione e un altro ancora elaborare i dati del sensore laser.

- Master: Il Master è il centro del Computational Level ed è il responsabile della gestione dei nodi e delle le comunicazioni tra di loro attraverso un sistema di messaggi e servizi. Una parte importante del Master è il Parameter Server che permette di salvare i dati che devono essere condivisi in uno spazio accessibile a tutti i nodi.
  Il Master si comporta fondamentalmente come un DNS della rete dei nodi: dopo aver ricevuto le informazioni da ogni nodo, le fornisce agli altri permettendo così di creare connessioni all'interno della rete. E' fondamentale avviare il nodo Master prima di ogni altro nodo in ROS. Il comando che permette di eseguirlo è semplicemente

```
$ roscore
```

- Messaggi: I Nodi comunicano tra di loro attraverso un sistema di scambio di messaggi. I messaggio sono formati da semplici strutture dati che possono essere formati da tipi primitivi e/o da array di questi che possono essere abbinate a piacere.

```
Header header            # timestamp in the header is the acquisition time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured around
                         # the positive Z axis (counterclockwise, if Z is up)
                         # with zero angle being forward along the x axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
                         # is moving, this will be used in interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].  If your
                         # device does not provide intensities, please leave
                         # the array empty.
```

Figura 2.1.1 Esempio di file .msg, in particolare /std_msgs/LaserScan

- Topics: I Messaggi sono distribuiti ai nodi attraverso un sistema di publishers/subscribers in cui ogni nodo pubblica un determinato tipo di messaggi, ad esempio i messaggi sulla posizione del robot nella mappa. A questo topic si

sottoscriveranno poi tutti i nodi interessati a questa informazione i quali riceveranno tutti i messaggi pubblicati sul topic. Possiamo immaginare i topic quindi come un bus di comunicazione in cui possono viaggiare solo messaggi dello stesso tipo. In generale ogni nodo può pubblicare/sottoscriversi a uno o più topic contemporaneamente e ogni topic può avere uno o più nodi che si sottoscrivono/ pubblicano. I publisher/subscribers non possono sapere quali altri nodi sono connessi. I topic hanno quindi un paradigma di comunicazione molti-a-uno o uno-a-molti.

- Servizi: I Servizi vengono usati per la comunicazione tra nodi quando sono richieste dalle applicazioni che richiedono una richiesta e una risposta. Sono definiti come una coppia di messaggi: uno per la richiesta e uno per la risposta. Un nodo può offrire un servizio che altri nodi usano semplicemente mandando un messaggio e aspettando la risposta. Hanno quindi un paradigma di comunicazione uno-a-uno.

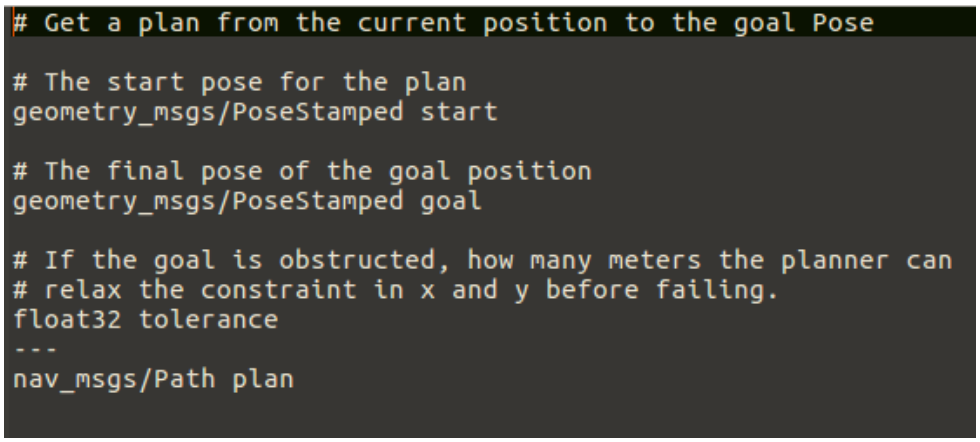

Figura 2.1.2 Esempio di file .srv. I due messaggi di richiesta e risposta sono divisi dalla riga "---"

- Bags: Le bag sono un formato di storage e riproduzione dei messaggi di ROS. Nella fase di registrazione della bag verranno quindi salvati tutti i messaggi inviati fra i vari nodi, come possono essere quelli relativi alla navigazione, al laser o alle telecamere. I dati salvati potranno essere riprodotti in qualsiasi momento. Questo velocizza molto il processo di sviluppo e test degli algoritmi, eliminando così di fatto il bisogno di fare continue prove sul robot fisico, ma permettendo semplicemente di appoggiarsi al sistema delle bag.

Il sitema dei nomi ha una grande importanza nel Computational Graph Level. Ogni resource (Messaggio, Servizio, Nodo) ha il suo nome univoco. Hanno una struttura fortemente

gerarchica, molto simile alla gestione dei file in un Sistema Operativo. In generale le risorse possono essere create e accedute solamente all'interno dello stesso namespaces. Le connessioni invece possono essere fatte tra namespace distinti.

Un' esempio di connessioni della rete del Computation Graph Level:
Possiamo avere un nodo che gestisce il Laser del robot e pubblica su /diago/laser messaggi di tipo sensor_msgs/LaserScan. Possiamo creare un nuovo nodo che si sottoscrive a questo topic a cui arrivano tutti i messaggi publicati su quel topic, se un nuovo nodo si sottoscrive a questo topic, ad entrambi arriveranno esattamente gli stessi messaggi.

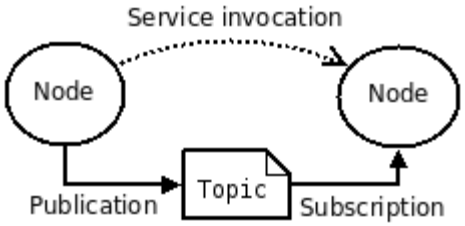

Figura 2.1.3 Comunicazione tra due nodi ROS. Viene messa in evidenza la differenza che c'è tra topic e service

Community Level:

E' la parte di ROS che si occupa dello scambio di informazioni e progetti tra le varie comunità di sviluppatori.

- Distribuzioni: Sono collezioni di stack che rendono più facile l'installazione e l'aggiornamento del sistema ROS

- Repositories: Sono collezioni di codice che vengono pubblicate dai vari componenti della comunità i quali rilasciano parti di software utilizzabili da chiunque.

- ROS Wiki: E' la fonte di informazioni principale e forum di ROS, nel quale ognuno può contribuire con nuovi articoli, tutorial o correzioni.

Figura 2.1.4 Alcune delle ultime distribuzioni rilasciate

2.2 Strumenti Utilizzati

Il sistema ROS, come già detto, mette a disposizione degli sviluppatori un grande numero di packages e tool per lo sviluppo di robot, questi strumenti agevolano notevolmente il lavoro dei ricercatori.
Si possono trovare pacchetti per la navigazione, per la visualizzazione, per il mapping etc.
In questa sezione andremo ad analizzare gli strumenti più importanti che sono stati usati all'interno di questo progetto per costruire la mappa semantica.

Stage

Stage è un simulatore di uno spazio bidimensionale che include anche molti modelli di sensori come sonar, laser e camere così da poter essere il più possibile coerente con il mondo reale.[9]

Stage prende in input un file ".world" che descrive un "mondo". Questo file comprende tutte le informazioni di cui il simulatore ha bisogno per descrivere l'ambiente, come l'immagine della mappa, gli ostacoli presenti e la posizione dei vari robot.

Il nodo stageros è sottoscritto al topic robot/cmd_vel, in modo da poter comandare il robot attraverso joystick o tastiera, inviandogli messaggi che contengo informazioni sulla velocità lineare e angolare da impartire al robot. [10]
Pubblica invece nei seguenti topic:

- odom : Fornisce l'odometria del robot nella mappa, questa dipende dal file .world passato per argomento
- base_scan : Dati dello scanner laser
- base_pose_ground_truth: Posizione del robot assoluta, indipendente dal file .world.
- image : Immagini della camera
- depth: Immagini della camera di profondità
- camera_info: Fornisce dati sulla calibrazione della camera

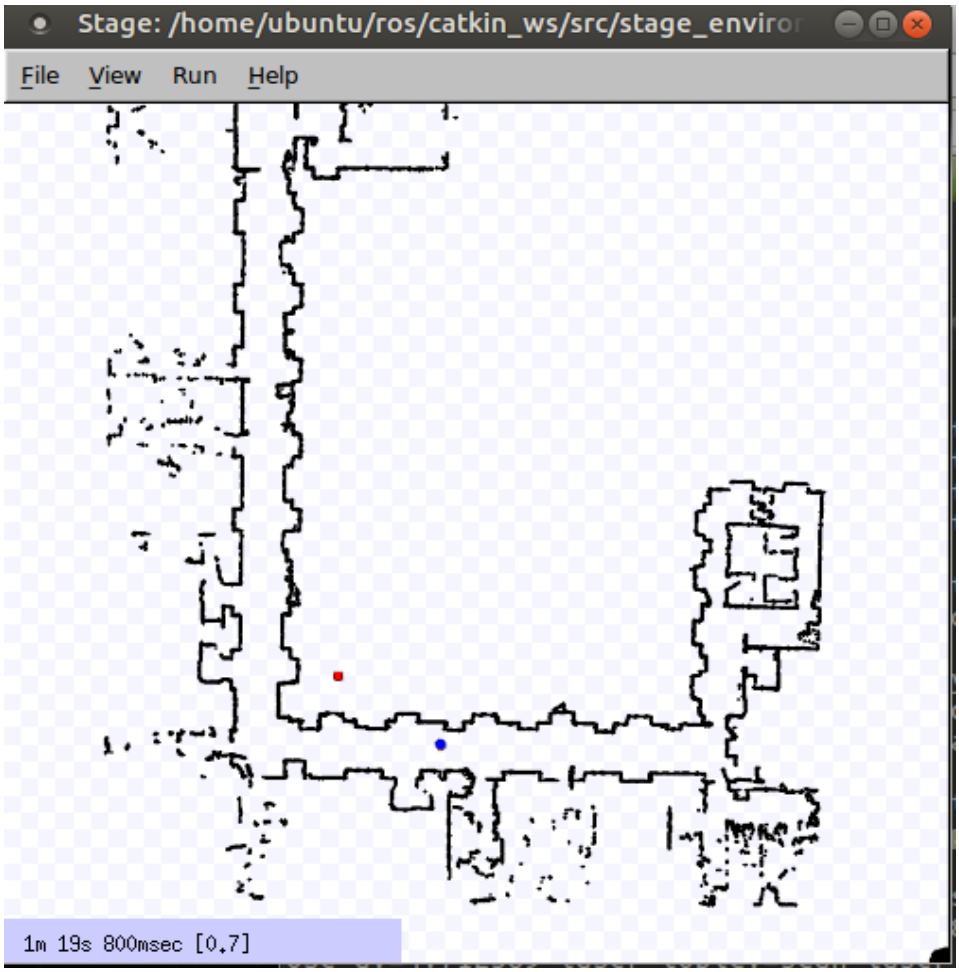

Figura 2.2.1 Simulazione in Stage. Il cerchio blu rappresenta il robot comandato dall'utente. Il quadrato rosso è invece un ostacolo che è possibile spostare con il mouse.

Roslaunch

Roslaunch è un tool che permette di eseguire più nodi ROS contemporaneamente, sia localmente che da remoto, consentendo anche di settare i relativi parametri. Prende come argomento uno o più file XML con estensione .launch. Questi file contengono i dati sui nodi da lanciare e sui rispettivi parametri. [11]
Questo tool è estremamente comodo poiché, in sistemi complessi, è spesso richiesta l'esecuzione di un gran numero di nodi che sarebbe estremamente difficile da gestire altrimenti.

Il comando per eseguire un file .launch è:

```
$ roslaunch package_name file.launch
```

```
<launch>

  <arg name="world_file" default="AUTOGEN_DISB1_diago.world" />
  <arg name="base_frame" default="base_frame" />
  <arg name="laser_topic" default="scan" />
  <arg name="laser_frame" default="laser_frame" />

  <master auto="start"/>
  <param name="/use_sim_time" value="true"/>

  <node pkg="stage_environments" type="stageros_experimental" name="stageros" args="-u $(find stage_environments)/maps/$(arg world_file)"
respawn="false" required="true" output="screen">
    <param name="base_watchdog_timeout" value="0.2"/>
    <param name="base_frame" value="$(arg base_frame)"/>
    <param name="laser_topic" value="$(arg laser_topic)"/>
    <param name="laser_frame" value="$(arg laser_frame)"/>
  </node>

</launch>
```

Figura 2.2.2 Esempio di file .launch

Come si può vedere dalla Figura 2.2.2 il dati contenuti nel file sono compresi nel tag :

<launch> … </launch>.

Per creare un nodo invece si usa:

<node pkg = "..." type = "…" name = "..."/>

- pkg: Package in cui si trova il nodo che si vuole eseguire
- type: Nome del eseguibile del nodo.
- name: Permette di sovrascrivere il nome del nodo

Ci sono poi una serie di parametri e argomenti che possono essere passati nella creazione del nodo.

AMCL

AMCL è un localizzatore che ci permette di stimare la posizione del robot in una mappa bidimensionale. Prende in input i dati del sensore laser, una mappa e i messaggi della trasformata con cui calcola , attraverso il metodo Monte Carlo per la localizzazione, una stima della posizione del robot nell'ambiente.[12]

Move Base

Il package move_base è una parte fondamentale del navigation stack interno a ROS. Ci permette di dare in input delle coordinate verso le quali vogliamo far muovere il robot, il quale vi si dirigerà autonomamente. Questo è eseguito tramite un sistema di Motion Plannig che riesce ad elaborare il percorso migliore per spostarsi all'interno della mappa, riuscendo anche ad evitare eventuali ostacoli presenti lungo il tragitto.

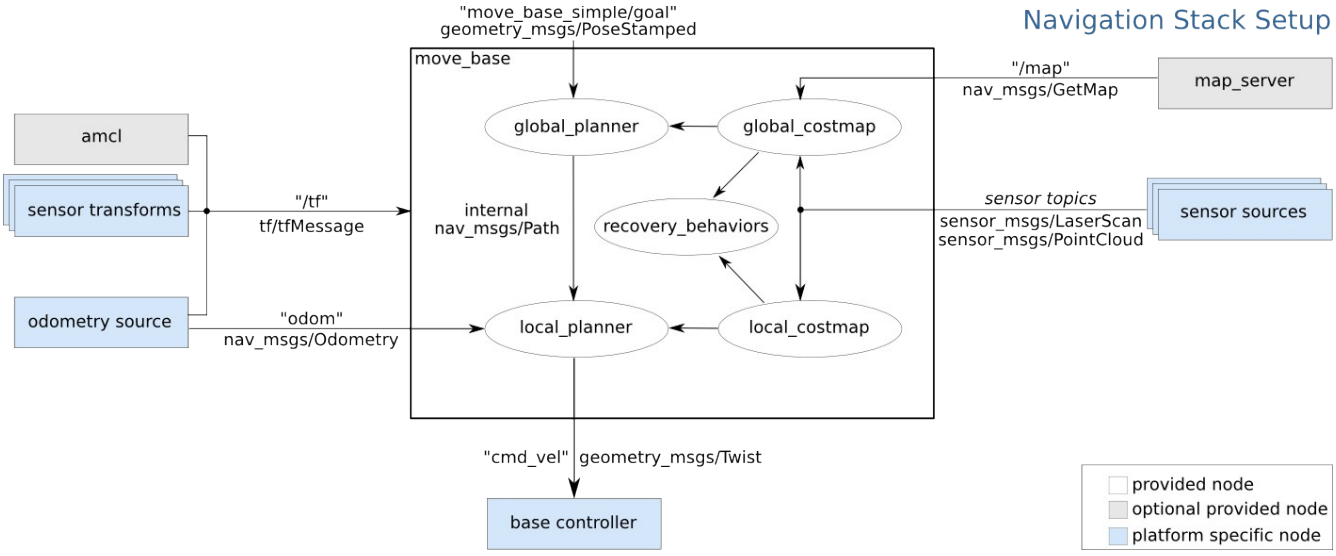

Figura 2.2.3 Grafico del nodo move_base e della sua interazione con il navigation stack.

Si può comunicare con il nodo move_base attraverso un SimpleActionClient, questo ci dà la possibilità di tenere sotto controllo lo stato del goal, permettendoci di interromperlo in qualsiasi momento. Pubblicando nel topic move_base/goal si può indicare una nuovo goal, questo potrà poi essere cancellato attraverso il topic move_base/cancel. Quando invece il robot arriva a destinazione move_base pubblica nel topic move_base/result l'avvenuta fine del goal.

Se non ci interessa tracciare l'andamento del goal, è possibile comunicare con messaggi attraverso il topic move_base_simple/goal.

A questo topic vanno mandati messaggi del tipo std_msgs/PoseStamped i quali contengono al loro interno le coordinate del punto nel quale voglio far dirigere il robot.

RVIZ

Rviz, acronimo di ROS Visualization, è un visualizzatore 3D che è in grado di mostrare e calcolare dati provenienti dai sensori come ad esempio il laser o immagini di telecamere e tutte le altre informazioni inerenti il sistema di simulazione, come possono essere mappe, robot e oggetti.

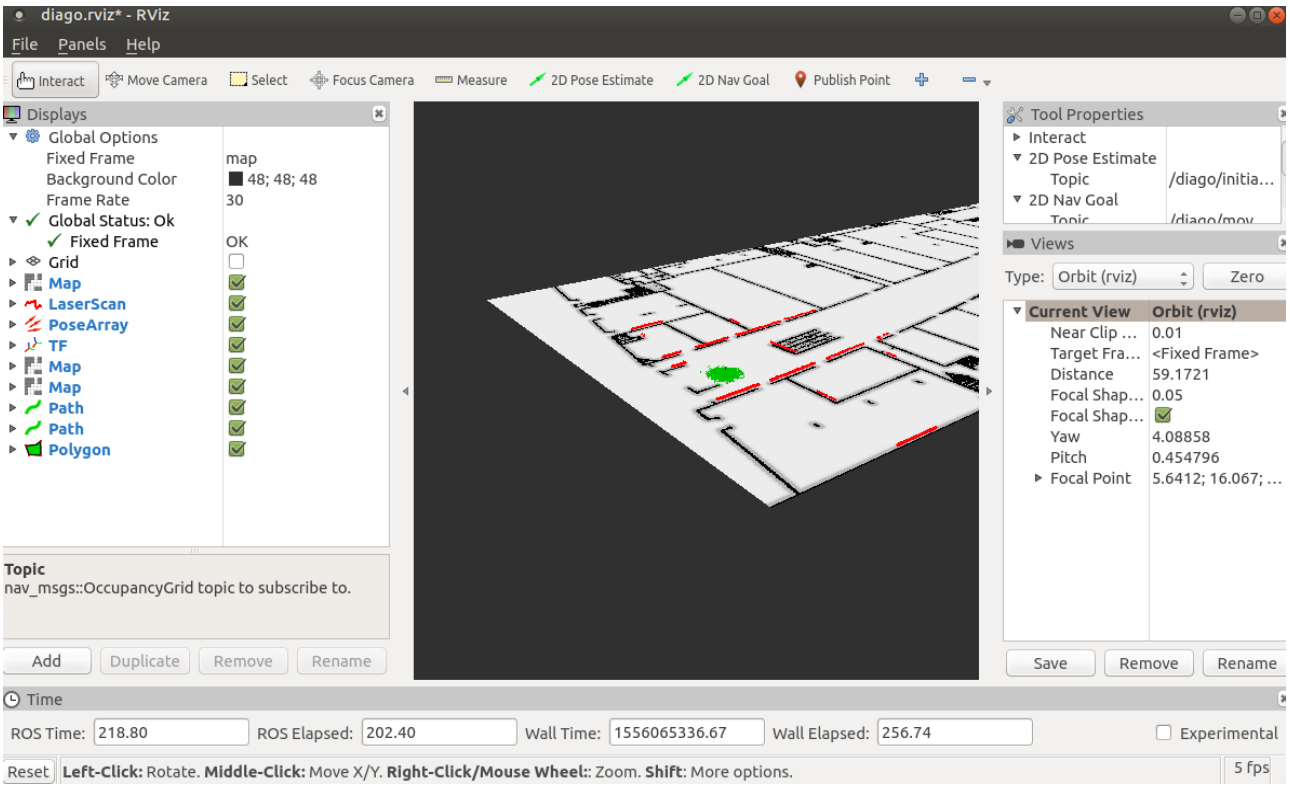

Figura 2.2.4 Interfaccia di Rviz

Come si può vedere nella Figura 2.2.4, nella colonna di sinistra abbiamo una serie di proprietà del simulatore, tra cui il LaserScan, che rappresenta appunto le proprietà del laser. Posso quindi sottoscrivermi al topic dedicato allo scambio di messaggi del sensore laser e grazie a questo comunicare i dati a Rviz.
Abbiamo poi TF che tiene traccia dello spostamento delle varie componenti fisiche del robot. Map invece è il gestore, appunto, della mappa il quale si sottoscrive al topic dedicato /map.

Rviz può essere usato insieme al navigation stack per muovere il robot all'interno di una mappa in modo autonomo.
Per fare questo è importante prima di tutto utilizzare un localizzatore, come AMCL, per stimare la posizione del robot nella mappa.
Per la navigazione posso usare il package move_base che mi permette di navigare autonomamente nella mappa, in particolare il topic in cui pubblica è move_base_simple/goal.
Per settare un goal basta cliccare sul pulsante 2D Nav Goal e poi sul punto desiderato sulla mappa.

LU4R

LU4R (*adaptive spoken Language Understanding chain For Robots* tool) è un tool per il riconoscimento e l'interpretazione del linguaggio parlato. E' scritto completamente in Java ed è basato su una architettura Client/Server che lo rende indipendente dal robot specifico che lo utilizzerà: il Client è il robot utilizzatore mentre il Server è LU4R.
Il sistema LU4R riceve in input una trascrizione del comando e produce una interpretazione che è consistente con il sistema, utilizzando una mappa semantica.[13]

La parte fondamentale di LU4R è il sistema SLU (Spoken Language Understanding), il suo compito è quello di tradurre le espressioni dell'utente in "semantic frames", cioè in frammenti di frase che esprimono specifici concetti semantici di interesse per il sistema. Per fare ciò LU4R si appoggia al sistema FrameNet [20] il quale fornisce le basi linguistiche e cognitive per l'interpretazione. Ad esempio nella frase

"*take the book on the table*"

"take" è un frame che rappresenta una azione, in particolare quelle di prendere,
"the book on the table" è invece il "THEME" cioè l'oggetto che riceve l'azione.
Questi frame servono quindi come connessione tra il linguaggio parlato ed il sistema che gestisce il robot.

Il sistema SLU si divide il 4 parti fondamentali:

- Analisi morfo-sintattica: È l'analisi che viene realizzata su ogni espressione in input, fornisce informazioni morfologiche e sintattiche.
- Re-ranking: Vengono ordinate le varie ipotesi di trascrizione in base alla veridicità.
- Action Detection: Vengono analizzati i vari frame generati e fra questi viene individuato quello che rappresenta l'azione da compiere.
- Argument Labeling: Assegna ad ogni frame la giusta etichetta come ad esempio THEME.

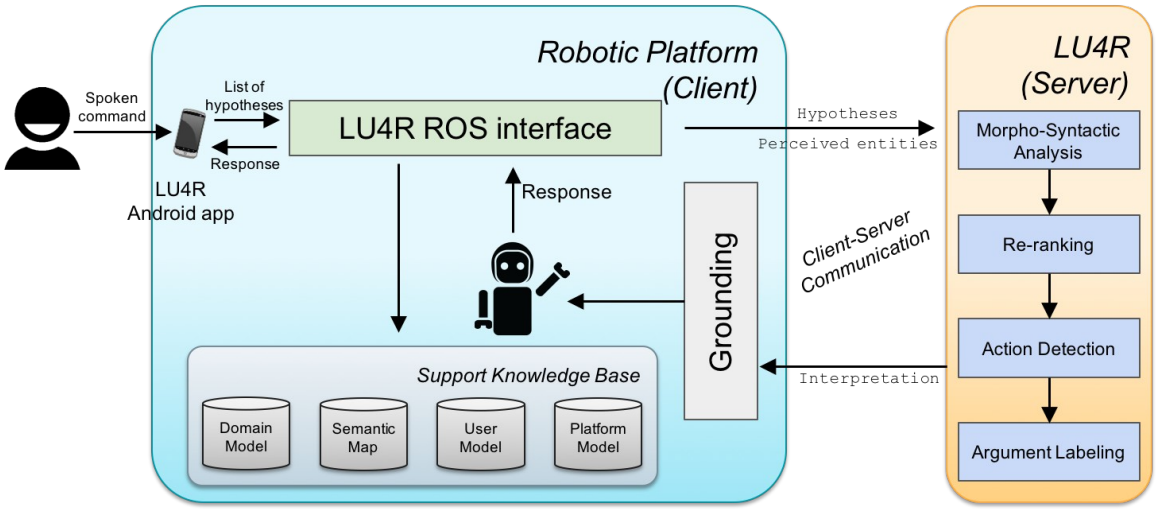

Figura 2.2.5 Rappresentazione architettura Client/Server di LU4R

Il sistema LU4R nel suo complesso si forma di 3 parti: il Server LU4R, la LU4R Android App e la LU4R ROS Interface.

- Il Server LU4R è il cuore del sistema ed è il responsabile dell'interpretazione degli input che gli arrivano dalla app LU4R. E' la parte che contiene il sistema SLU.

- La LU4R Android App si occupa fondamentalmente della trascrizione delle espressioni vocali, che vengono rilevate dal dispositivo attraverso un sistema ASR (Automatic Speech Recognition). Questo sistema genera una o più ipotesi di trascrizione che verranno poi inviate al Server LU4R attraverso connessioni a protocollo TCP. È disponibile anche un joystick che permette di comandare il robot.

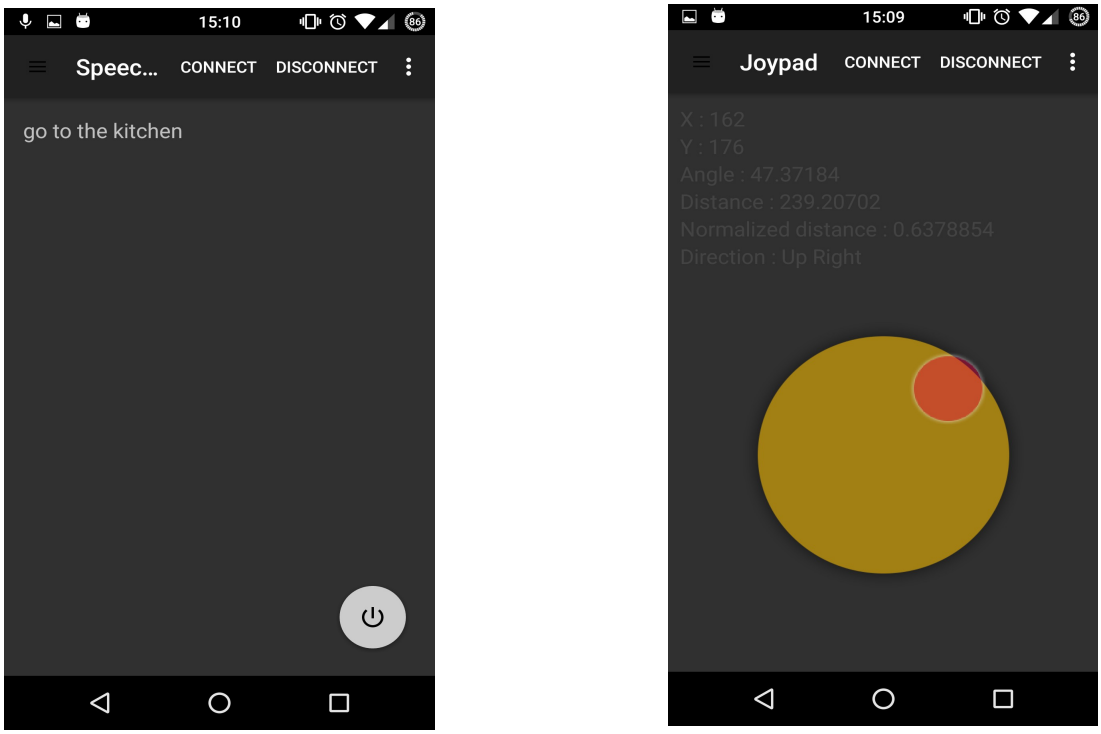

Figura 2.2.6 L'interfaccia dell'app LU4R: nel caso del riconoscimento vocale a sinistra e quello del joystick per la navigazione a destra

- LU4R ROS Interface è un insieme di nodi che permette l'implementazione di LU4R in ROS attraverso il sistema di publisher/subscriber e si divide a sua volta in 4 componenti.

1. android_interface: È il server TCP al quale si connette la app Android. Si occupa di estrarre le informazioni necessarie e di gestire tutte le ipotesi di trascrizione che gli arrivano dalla app, le quali vengono poi pubblicate dal nodo sul topic /speech_hypotesis. Il sistema è anche in grado di distinguere tra queste quella più verosimile, che viene pubblicata come stringa nel topic /best_speech_hypothesis. Si sottoscrive a /lu4r_response, contenente la risposta del Server LU4R in termini di interpretazione semantica dei frame e a /dialogue_mager_response, che contiene invece la risposta generata dal gestore AIML.

2. lu4r_ros: È il ponte tra il sistema LU4R e ROS, comunica attraverso HTTP con un server LU4R attivo

3. aiml_ros: È il responsabile dell'interpretazione della espressione naturale e dell'elaborazione della risposta attraverso un file AIML (Artificial Intelligence Markup Language), il quale è in grado di instaurare un dialogo con l'utente.

4. framenet_ros_msgs:È il ponte con il sistema FrameNet

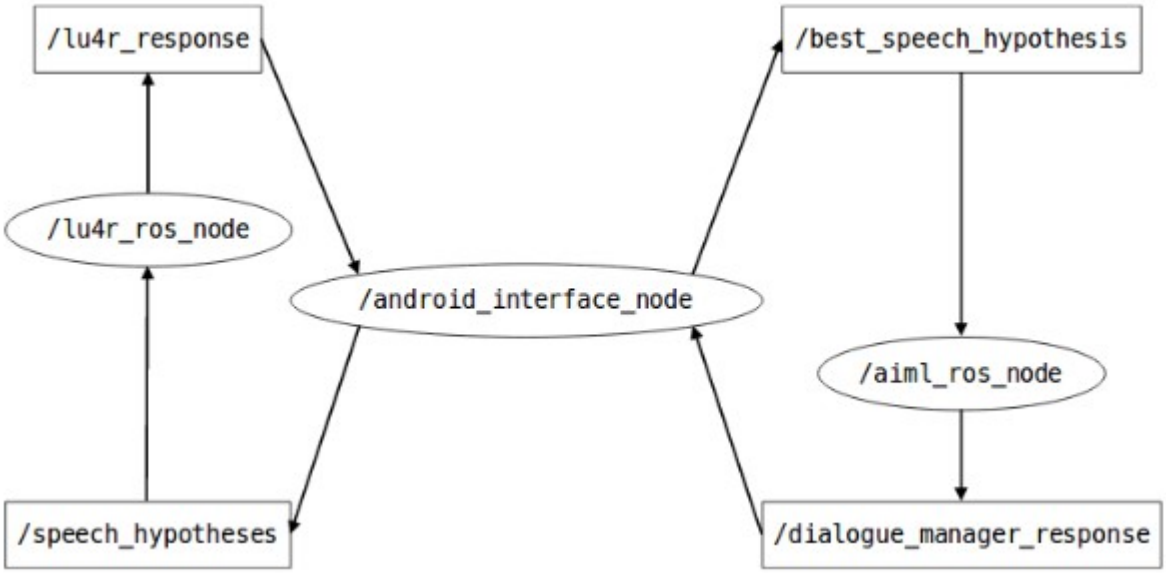

Figura 2.2.7 Grafo dell'interazione dei nodi all'interno di LU4R

La figura 2.2.7 spiega molto bene ciò che succede all'interno di LU4R. Il nodo android_interface è il centro del sistema. Riceve i messaggi dalla app di riconoscimento vocale e li inoltra ai due nodi lu4r_ros e aiml_ros. Questi nodi gestiscono i messaggi con le procedure indicate precedentemente e mandano i risultati di nuovo al nodo android_interface.

Capitolo 3

Mappe Semantiche

Mappa semantica è un termine utilizzato per indicare il processo che permette ad un robot di arricchire la mappa utilizzata per la navigazione con informazioni sull'ambiente in cui si trova come, ad esempio, dove si trova la cucina, l'ufficio o un oggetto. Questo ci permette di far muovere il robot in completa autonomia attraverso i vari punti di interesse della mappa utilizzando un sistema di navigazione. Questo aspetto è di grande interesse pratico, basti pensare al caso di robot domestici che, avendo una mappa semantica della casa, possono svolgere compiti molto complessi ed essere molto più utili e facili da usare per gli utenti.

Per capire come realizzare una mappa semantica dobbiamo innanzitutto sapere come fa il robot ad orientarsi e muoversi nello spazio, dobbiamo quindi capire che cos'è e come funziona una mappa metrica, poi dobbiamo capire come aggiungere a questa mappa delle strutture che ci rendano possibile arricchirla di informazioni aggiuntive sull'ambiente circostante, costruire cioè lo scheletro della mappa semantica, infine dobbiamo capire quali sono i metodi per inserire le informazioni all'interno di queste strutture.

3.1 Mappa Metrica

La mappa metrica è la rappresentazione dell'ambiente fisico in cui il robot si trova. Questa mappa viene ricavata dalle informazioni date da sensori, come laser o camere, le quali vengono elaborate attraverso vari metodi, ad esempio il metodo SLAM (Simultaneous Localization and Mapping). Questo metodo, come dice il nome, è in grado di creare una mappa metrica dello spazio man mano che viene esplorato attraverso i sensori laser e, allo stesso tempo, riesce a localizzare il robot nell'ambiente basandosi su degli oggetti fissi nel sistema come possono essere, ad esempio, dei muri.[15]
Nel sistema ROS un tool in grado di generare mappe metriche a partire da dati laser è GMapping, il quale usa proprio il metodo SLAM.

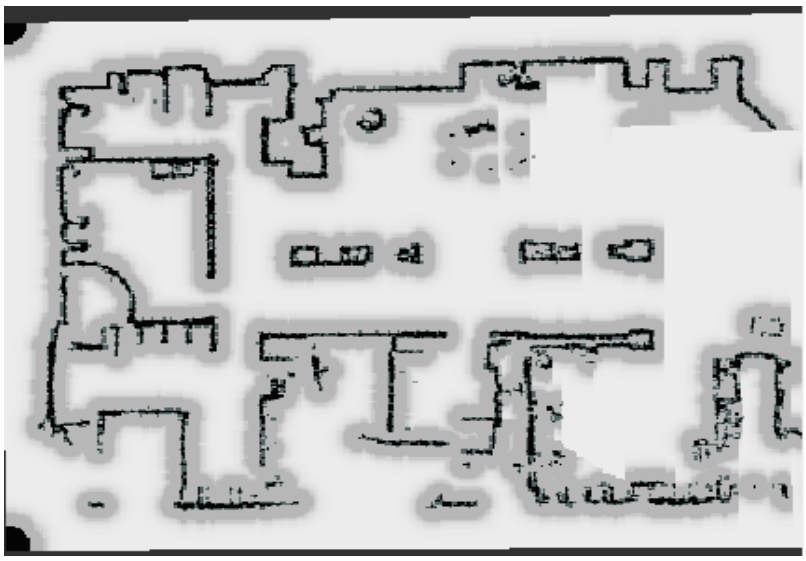

Figura 3.1.1 Mappa metrica creata con GMapping e visualizzata con RViz

3.2 Rappresentazione

Ci sono diversi metodi per realizzare una mappa semantica: la rappresentazione geometrica, la rappresentazione con griglia oppure quella attraverso la mappa topologica.

Nella rappresentazione geometrica vengono usate delle primitive geometriche per rappresentare e delimitare i luoghi di interesse della mappa.
Nella rappresentazione con griglia si divide la mappa in un insieme di celle di dimensioni uguali. Queste celle vengono etichettate una ad una, basandosi sul loro significato all'interno della mappa semantica. [14]

Sulla rappresentazione topologica ci fermeremo più approfonditamente in quanto è quella scelta per la realizzazione della mappa semantica in questo progetto.

Questa rappresentazione prova a gestire il problema utilizzando i grafi. Ogni nodo rappresenta un informazione in qualche modo significativa per il sistema, ad esempio può rappresentare una stanza, un oggetto che vogliamo inserire nella nostra mappa o semplicemente un punto di passaggio del robot. Gli archi invece, rappresentano le connessioni esistenti tra i vari nodi, ci dicono quindi in che modo sono connessi fra loro gli oggetti dell'ambiente e come possono interagire tra loro. Si possono anche aggiungere informazioni agli archi, ad esempio dandogli un peso relativo alla distanza tra un nodo ed un altro a lui connesso, in modo tale da poter creare degli algoritmi che sappiano scegliere il percorso migliore per far muovere il robot da un nodo all'altro.

Formalizzando possiamo dire che la nostra mappa topologica è una coppia $T = (N , E)$ dove $N = \{n_1 , \ldots , n_n\}$ è un inseme di nodi $E = \{e_1 , \ldots , e_m\} \subseteq N \times N$ un inseme di archi. Un nodo è definito come $n_i = (pos, label)$ con $pos = (x, y)$ e label un identificatore unico che definisce l'oggetto/luogo specifico, questo definisce quindi univocamente la sua posizione nella mappa metrica e il suo significato nella mappa semantica. Un arco $e_i = (src,dst)$ descrive il collegamento tra due nodi tale che src è la partenza e dst l'arrivo. I grafi possono essere sia orientati che non.[15]

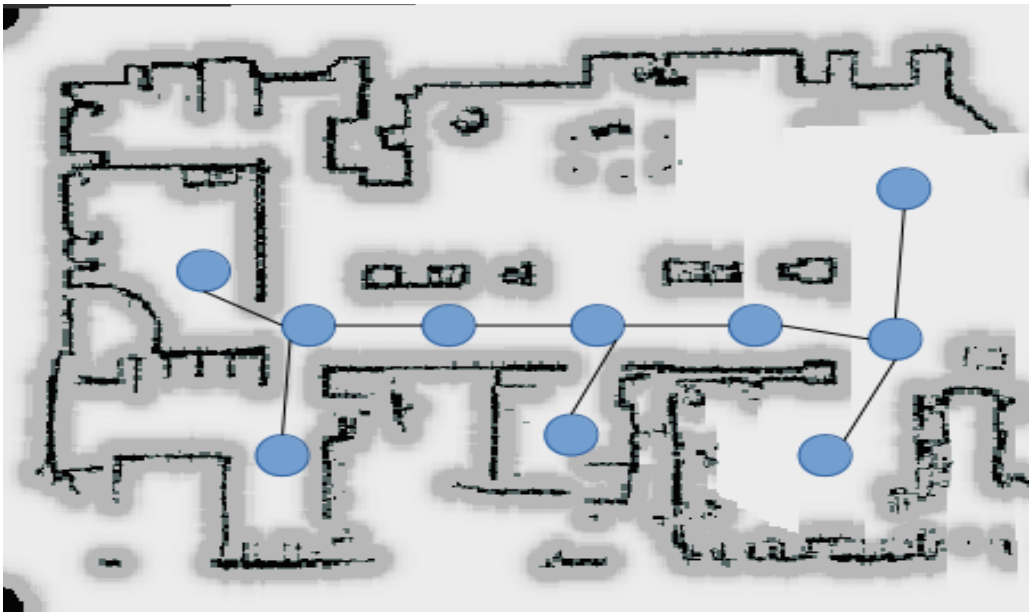

Figura 3.2.1 Rappresentazione di mappa topologica insieme alla rispettiva mappa metrica

3.3 Identificazione di oggetti

A questo punto dobbiamo creare un metodo che ci permetta di riconoscere quali sono effettivamente i punti di interesse per la mappa semantica e quali no.

Il metodo sicuramente più immediato, è quello di creare prima una mappa dell'ambiente e aggiungere in un secondo momento i punti di interesse, ma questo è evidentemente poco utile ai fini pratici.

Un metodo più interessante è quello della Object Detection, cioè il riconoscimento autonomo da parte del robot, attraverso sensori come laser o camere, di oggetti di particolare rilievo. Dal riconoscimento di determinati oggetti possiamo ottenere molte informazioni utili sulla semantica dell'ambiente, ad esempio, individuare un microonde ci può far concludere con buona approssimazione che ci troviamo in cucina.
Una libreria che ci permette di implementare questo genere di algoritmi è OpenCV.
Abbiamo poi bisogno anche di un sistema per capire cosa è una stanza e quando il robot vi entra. Un sistema (proposto da Jensfelt[16]) usa il laser per capire quando il robot passa attraverso una porta. Se il robot ha attraversato una porta vuol dire che ha cambiato stanza.

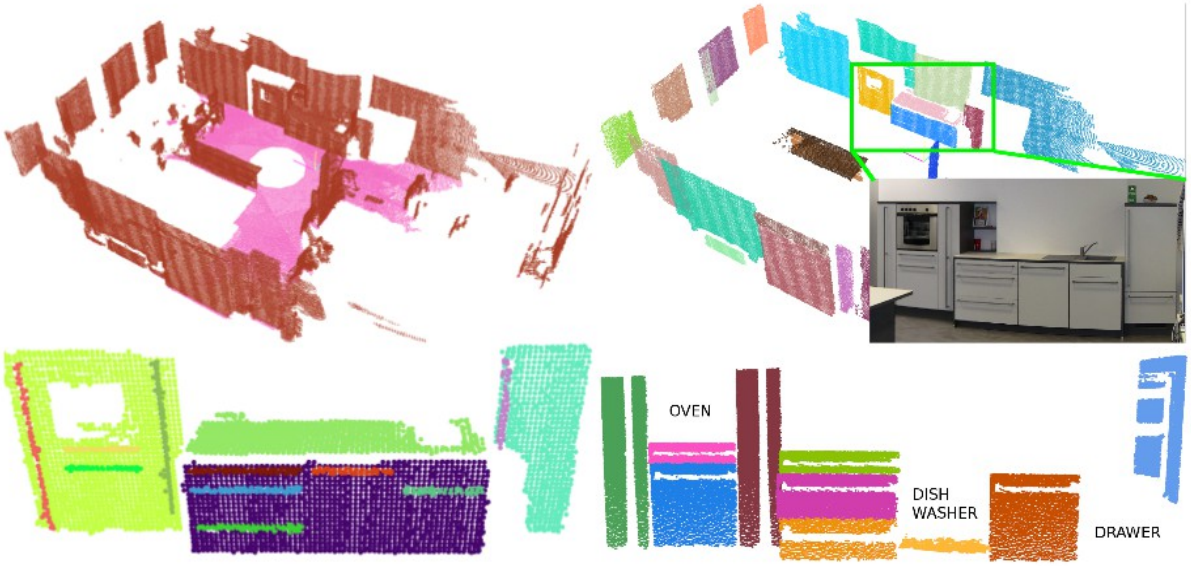

Figura 3.3.1 Esempio di Object Detection ([18])

Infine un altro metodo è quello dell'interazione diretta tra utente e robot, è il metodo scelto per questo progetto.
L'utente ha quindi la possibilità di creare la mappa semantica interagendo vocalmente con il robot. Ad esempio dicendo "Questa è la cucina" il robot, attraverso un sistema di interpretazione del linguaggio come LU4R, elabora l'informazione. In questo caso viene estratta "cucina" come informazione di rilievo ai fini della mappa semantica del sistema e il robot lo aggiunge alla mappa.

Capitolo 4

Costruzione di una Mappa Semantica

In questo capitolo verrà descritto nello specifico il metodo utilizzato per la costruzione della mappa semantica in questo progetto. Si partirà dalla struttura scelta per la rappresentazione, descrivendo poi in che modo si sono messi in connessione i vari strumenti di ROS per il riconoscimento vocale, la localizzazione e la navigazione del robot.
Verranno in fine descritti gli esperimenti condotti con il robot nei simulatori interni a ROS e con il robot Pioneer.

4.1 Realizzazione

Per questo progetto si è creato un unico nodo ROS dal nome semantic nel package semantic che gestisce tutta la computazione.
Si è voluta implementare una realizzazione di mappa semantica attraverso un grafo con lista di adiacenza non orientato. Questo grafo viene creato "on-line" al momento dell'esplorazione dell'ambiente da parte del robot.

```
struct graph_prop {
  int n_vertices;
  int n_edges;
}graph_prop;

struct graph{
    linked_list * nodes;
  struct graph_prop* properties;
};
```

```
struct graph_node_prop{
  const char* label;
  int is_something;
  float x,y;
};

struct graph_node{
    void * value;
    linked_list * out_edges;
    STATUS state;
};
```

Figura 4.1.1 Strutture usate per la rappresentazione dei grafi. Come si può vedere la struttura grafo mantiene una lista di tutti i nodi presenti. A sua volta ogni singolo nodo contiene una lista di tutti i nodi a cui è collegato.

Il grafo registra, in particolare, per ogni nodo la sua posizione (x,y), una stringa che identifica il nodo ed un valore is_something che segnala l'importanza o meno del nodo nella mappa semantica.

Data quindi la struttura sul quale è stato basato il grafo, si procede a descrivere dapprima come avviene la costruzione della mappa semantica e successivamente il suo utilizzo attraverso il sistema di navigazione autonoma.

Costruzione

La prima parte dell'attività del nodo è l'esplorazione e la relativa costruzione della mappa semantica. Il robot viene quindi comandato attraverso joystick/tastiera ed è costantemente aggiornato riguardo la sua posizione nella mappa dal topic /diago/amcl_pose del localizzatore AMCL.

Comincia quindi la creazione del grafo. Un nuovo nodo è creato non appena il sistema rileva che il robot è a più di 0,5 metri di distanza dal nodo più vicino e a questo viene collegato con un arco il nuovo nodo. Si viene così a formare un grafo che segue il robot nella sua esplorazione e che è in grado di gestire incroci e deviazioni. Si creerà a questo punto una struttura del tutto simile a quella rappresentata in Figura 3.2.1.

```
PoseCallback(x,y){

  _X = x
  _Y = y
  grah_node nearest = G.get_nearest_node(_X, _Y)

  if( dist(_X,_Y,nearest.x, nearest.y) < 0.5){
      graph_node new = G.new_node(_X,_Y)
      G.add_edge(nearest,new)
  }
}
```

Figura 4.1.2 Pseudocodice della funzione che crea il grafo.

La funzione in Figura 4.1.2 si occupa proprio di gestire ciò che è stato appena descritto. Viene richiamata ogni volta che il nodo rileva un nuovo messaggio nel topic del localizzatore.
I parametri (x,y) della funzione rappresentano il contenuto del messaggio arrivato dal localizzatore e vengono registrati nelle variabili globali _X e _Y, che registrano appunto la posizione attuale del robot. G è il grafo. La funzione get_nearest_node(x,y) è incaricata di trovare il nodo più vicino. Per fare ciò controlla semplicemente nella lista dei nodi contenuti nella struttura graph (Figura 4.1.1) quali di questi abbia la distanza minore da _X e _Y.

A questo punto si può cominciare ad immettere le informazioni della mappa semantica.
Il nodo è collegato al sistema LU4R attraverso il topic /best_speech_hypothesis, grazie al quale riceve la stringa dell'interpretazione di ciò che l'utente ha detto, questa viene elaborata e se è una frase del tipo "Questa è la cucina" oppure " Siamo in cucina", il sistema setta la label del nodo più vicino al punto dove si trova attualmente a "cucina".

Il codice in Figura 4.1.3 gestisce proprio questa parte del sistema. La variabile msg contiene la stringa ricevuta dal topic /best_speech_hypotesis. La funzione msg.last() invece serve ad estrarre dalla stringa ricevuta la parte che ci interessa immettere come label.

```
if(msg.in("siamo") OR msg.in("quest")){
    graph_node nearest = G.get_nearest_node(_X,_Y)
    nearest.set_label(msg.last());
}
```

Figura 4.1.3 Immissione dei dati nella mappa semantica

A questo punto ci troviamo in una situazione simile a quella di Figura 4.1.4.

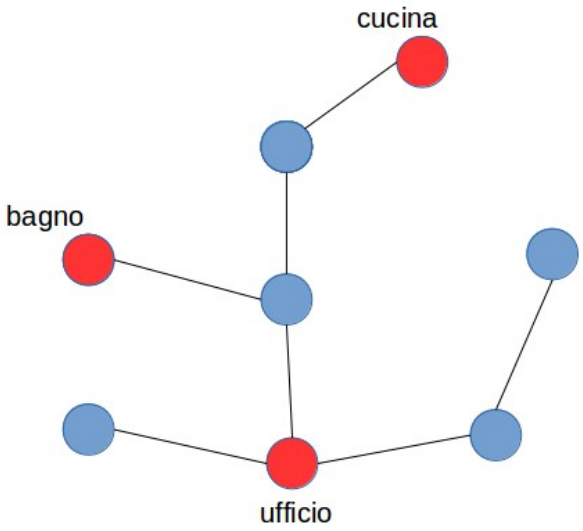

Figura 4.1.4 Possibile mappa topologica generata dal sistema. Si noti che gli archi sono tutti equivalenti in quanto approssimativamente tutti rappresentanti una distanza di 0.5 metri

Utilizzo

Dopo aver esplorato l'ambiente e costruito la mappa semantica che lo rappresenta, possiamo cominciare ad utilizzare il sistema di navigazione autonoma.

```
if(msg.in("vai")){
    graph_node destination = G.find_node_label(msg.last)
    graph_node source = G.get_nearest_node(_X,_Y)
    List node_list = G.DFS(source,destination)

    for all n in node_list{
      go_to(n.X,n.Y)
    }
}
```

Figura 4.1.5 Pseudocodice della funzione di gestione della navigazione autonoma

La funzione in Figura 4.1.5 viene richiamata ogni volta che viene ricevuto un messaggio nel topic /best_speech_hypotesis e il suo argomento msg contiene la trascrizione del comando vocale dell'utente. Possiamo quindi dare al robot un comando del tipo “Vai in cucina”. Così facendo il sistema interpreterà il messaggio e cercherà, tra i nodi presenti attualmente nel grafo, quello etichettato con “cucina”, designandolo come destinazione. Per fare questo si utilizza la funzione find_node_label(label) la quale cerca all'interno del grafo un nodo con etichetta uguale a quella passata per argomento. Troverà poi il nodo più vicino alla posizione del robot e lo sceglierà come nodo di partenza.
A questo punto partirà un algoritmo DFS(Depth First Search) il quale troverà il path tra il nodo di partenza e quello di destinazione, restituendo la lista dei nodi attraversati tra i due.

```
Algorithm pathDFS(G, v, z)
  setLabel(v, VISITED)
  S.push(v)
  if v = z
    return S.elements()
  for all e ∈ G.incidentEdges(v)
    if getLabel(e) = UNEXPLORED
      w ← opposite(v,e)
      if getLabel(w) = UNEXPLORED
        setLabel(e, DISCOVERY)
        S.push(e)
        pathDFS(G, w, z)
        S.pop(e)
      else
        setLabel(e, BACK)
  S.pop(v)
```

Figura 4.1.6 Pseudocodice dell'algoritmo DFS. S rappresenta la lista da ritornare.

Una volta ottenuta la lista dei nodi da attraversare, il robot comincia a pubblicare messaggi con all’interno le coordinate di ognuno di questi nodi sul topic che permette la navigazione: /diago/move_base_simple/goal. Questo viene fatto tramite la funzione go_to(x,y). Il sistema gestisce anche l’arrivo dei messaggi di fine dei goal di move_base attraverso il topic /diago/move_base/result per coordinare l’invio dei goal successivi.

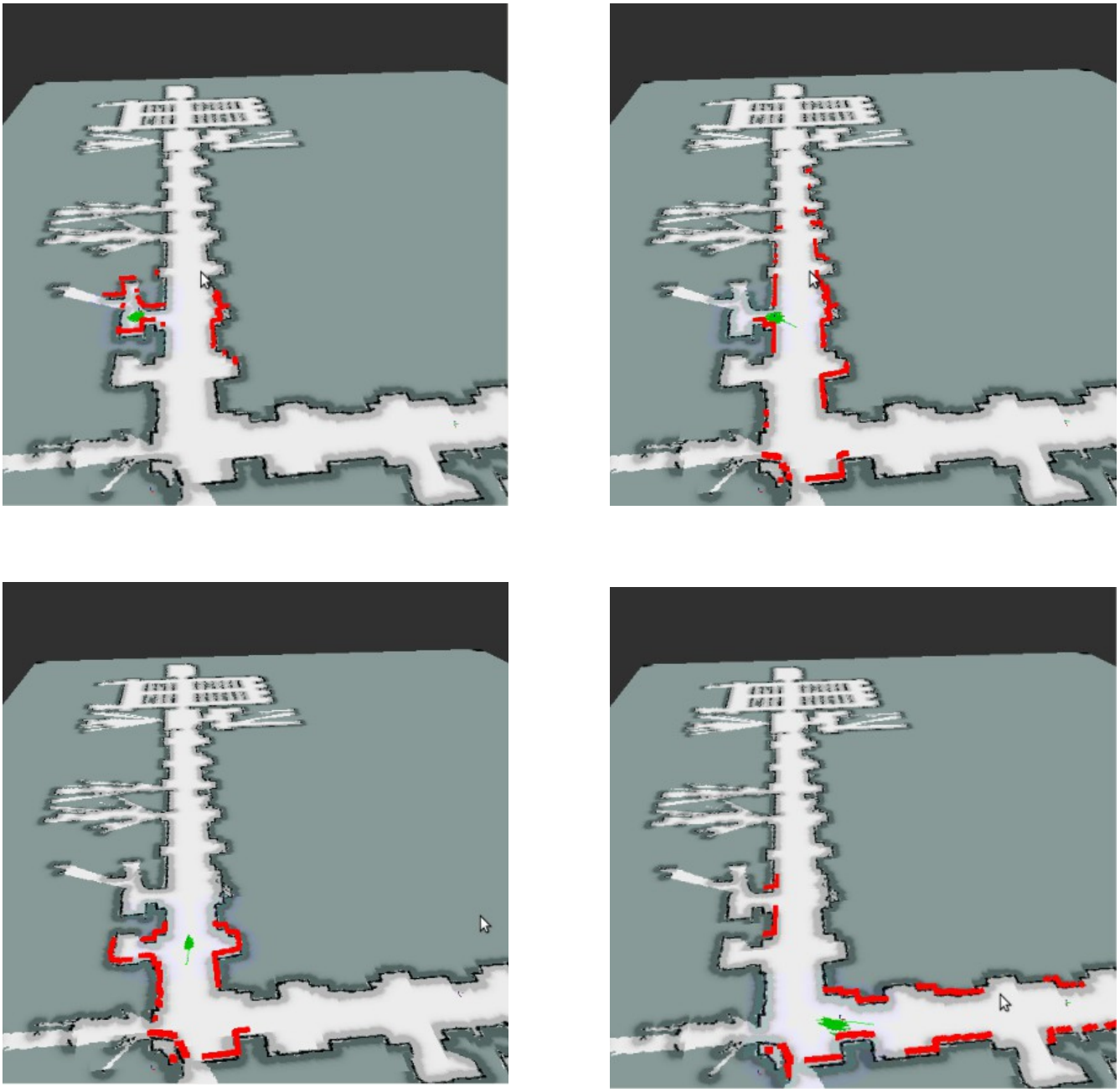

Figura 4.1.7 Queste 4 immagini evidenziano come avviene il processo di navigazione.

Come si può vedere nella Figura 4.1.7, nella prima immagine il robot è dentro una stanza, la nuvola di punti rappresenta le possibili posizioni stimate dal localizzatore. Nelle immagini successive si possono vedere le linee verdi che rappresentano i goal inviati al robot. Ogni goal lo fa muovere verso la posizione di un nodo, in questo modo riesce ad arrivare alla destinazione attraversando tutti i i nodi che li collegano.

AIML

Infine è stata realizzata la parte dedicata all'interazione vocale con l'utente: il gestore AIML. Si è creato un file .aiml in grado di gestire una piccola conversazione, come salutare, chiedere e ricordare il nome. Il sistema è anche in grado di rispondere correttamente alle frasi del tipo "Questa è la cucina", " Siamo in cucina", "Vai in cucina" con frasi del tipo "OK [NOME], siamo in cucina" oppure "OK [NOME] , vado in cucina".

```
<category>
    <pattern> SIAMO IN *</pattern>
    <template>
        <think>
            <set name="posto"><star/></set>
        </think>
        [SAY] OK <get name="user"/> Siamo in <get name="posto"/>
    </template>
</category>
```

Figura 4.1.8 Estratto del file .aiml utilizzato.

I file .aiml sono formati da un insieme di tag i più importanti dei quali sono:

- category: Rappresenta l'unità di conoscenza.
- pattern: E' il riferimento con cui il sistema confronta ciò che ha ricevuto dalla app di riconoscimento vocale.
- template: Contiene ciò che viene eseguito quando il contenuto di <pattern> corrisponde con quello ricevuto dalla app.

Ci sono poi i tag <set> e <get> che servono rispettivamente per registrare e richiamare variabili, le quali vengono salvate con il nome contenuto in name="..." e possono essere richiamate in un qualunque punto del file. In questo caso la variabile "user" contiene il nome dell'utente e la variabile "posto" l'indicazione data dall'utente stesso.

Di seguito è rappresentato il grafo dell'interazione dei singoli nodi ROS nel sistema complessivo, ottenuto con il tool rqt_graph.

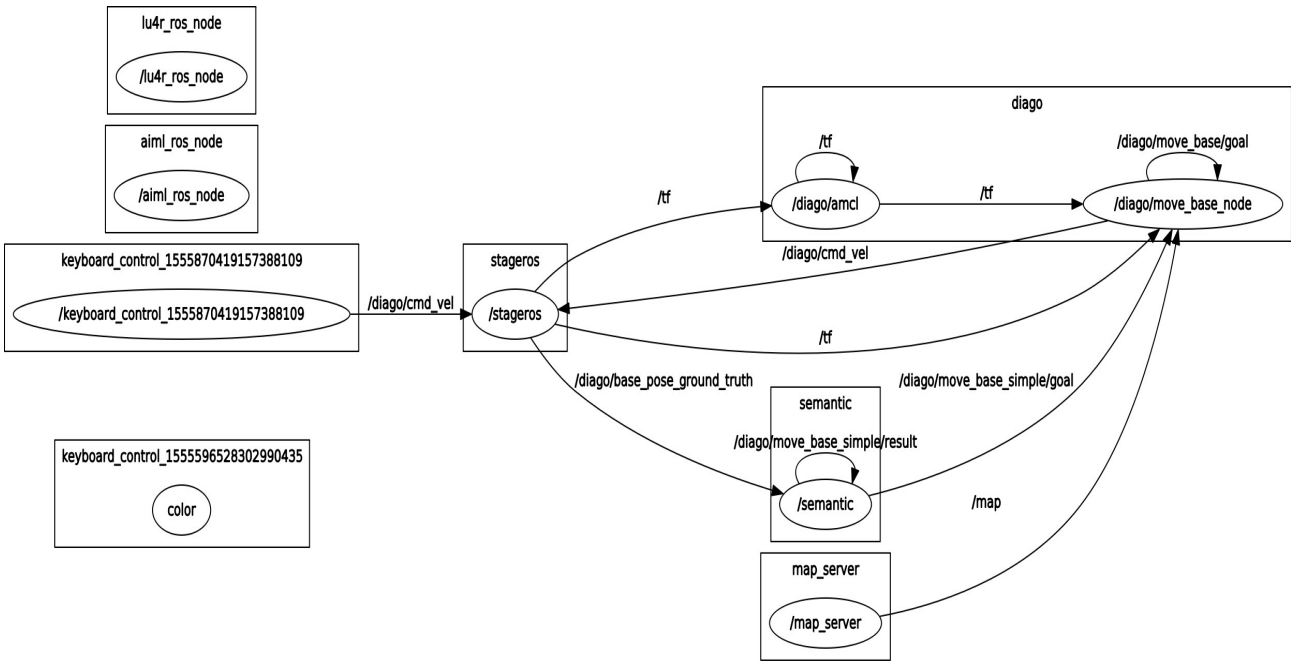

Figura 4.1.9 Grafo dell'interazione dei nodi

4.2 Esperimenti

Sono stati quindi svolti diversi esperimenti sul sistema progettato. Inizialmente è stato testato sul sistema di simulazione interno a ROS, successivamente invece si è sperimentato il sistema su un robot fisico, in particolare sul robot Pioneer.
Di seguito vengono descritti nello specifico le procedure utilizzate.

Esperimenti in simulazione

I test sono dapprima stati eseguiti attraverso il sistema di simulazione fornito da ROS. Sono state quindi usate delle mappe create in precedenza con GMapping, le quali sono state inserite in Stage per simulare l'ambiente reale, si sono usate più mappe fornite dall'Università. Si è cominciato quindi esplorando l'ambiente simulato comandando il robot da tastiera e si è visto come il sistema riusciva a creare un grafo dell'area esplorata in qualunque tipo di mappa e in modo del tutto parallelo alla navigazione guidata. Per la localizzazione, come spiegato precedentemente, è stato usato il localizzatore AMCL, mentre il pacchetto move_base è stato utilizzato per la navigazione autonoma. Entrambe questi sistemi si sono rivelati estremamente affidabili e precisi nell'esecuzione. Il robot è stato quindi in grado di orientarsi perfettamente in tutti i tipi di mappe utilizzati senza riscontrare problemi. Rviz è stato utilizzato per una più completa visione dell'ambiente in quanto fornisce maggiori

informazioni, come ad esempio è in grado di visualizzare i goal che vengono inviati a move_base.

In particolare si è riscontrato come il robot riesca ad orientarsi in ogni tipo di mappa creando la mappa semantica nello stesso momento in cui avviene l'esplorazione, in questo modo l'utente ha un interazione immediata con il robot, senza nessuna differenza sostanziale tra esplorazione e navigazione, i due stati possono essere scambiati in qualsiasi momento: quando l'utente dà un comando del tipo "Siamo in cucina", il robot salva la sua posizione, che può esse richiamata in qualsiasi momento con comandi del tipo "Vai in cucina". Questa simultaneità tra esplorazione e navigazione potrebbe essere molto utile in applicazioni reali, come robot domestici, consentendo di rendere molto più facile il compito dell'utente di "insegnare" al robot la struttura della mappa.
Una problematica di questa implementazione è però che il robot ripercorre solamente la strada segnata dai nodi posizionati in coordinate già esplorate, questo può essere poco conveniente in un primo momento ma, esplorando più approfonditamente la mappa, il grafo diventerà più fitto e connesso, fino a ricoprire l'intero ambiente, permettendo così una più agile mobilità del robot.

Esperimenti sul robot

Si è poi proseguito a testare il sistema in un ambiente reale. Per fare questo si è utilizzato il robot Pioneer, sviluppato nei laboratori della Sapienza.
Il robot Pioneer è formato da una base, con 4 ruote mobili e da un sensore laser Hokuyo, utilizzato per orientarsi nell'ambiente.

Inizialmente si è proceduto collegando il computer al robot tramite USB e utilizzando il pacchetto srrg_pioneer, contenente il nodo pioneer_robot che permette di interfacciarsi con il robot. Un joystick è stato collegato per comandare il robot attraverso la pubblicazione di messaggi nel topic /cmd_vel al quale il nodo del Pioneer è sottoscritto. Questo nodo si occupa di interpretare questi messaggi e di imporre al robot una velocità relativa al comando ricevuto. Si è avuto cura in questa fase di settare correttamente i topic che dialogavano tra robot e computer, cioè quelli relativi ai comandi di velocità del robot, alla posizione stimata dal localizzatore e quelli del sensore laser, per permettere al sistema di funzionare correttamente.
Si sono infine avviati i nodi relativi al sistema LU4R.

A questo punto, dopo aver completato la configurazione iniziale del sistema, si è potuto testare il pacchetto semantic all'interno del robot Pioneer.

Per prima cosa è stata creata una mappa metrica dell'ambiente, creando dapprima una bag e ricavando da questa, attraverso il tool Gmapping, un file contenente la mappa. Questa è stata poi passata al localizzatore AMCL, permettendogli in questo modo di stimare la posizione del robot all'interno dell'ambiente, utilizzando i sensori laser del Pioneer. Dopo questa procedura si è stato in possesso di tutte le informazioni necessarie per eseguire il nodo semantic.
Il robot ha quindi esplorato l'ambiente circostante per creare il grafo e ha registrato all'interno dei nodi la loro posizione, grazie alle informazioni fornite dal localizzatore. Questo grafo è stato poi riempito con le informazioni ricevute tramite il sistema di riconoscimento vocale, creando così la mappa semantica dell'ambiente, esattamente come avveniva all'interno del simulatore Stage.
La parte relativa alla navigazione autonoma è stata eseguita con il pacchetto move_base, questo ci ha permesso di muovere il robot nella mappa, facendogli seguire i nodi creati precedentemente e di raggiungere la destinazione desiderata.

Si è potuto evidenziare quindi come ROS sia uno strumento estremamente utile a chi sviluppa robot, in quanto ci ha permesso di creare un ambiente virtuale del tutto simile all'effettivo ambiente reale. Abbiamo sviluppato il sistema inizialmente all'interno di un simulatore per portarlo poi sul robot Pioneer, per fare questo è bastato collegare il robot al computer e settare i relativi topic, non sono state necessarie modifiche nei singoli nodi. Il sistema nel suo complesso è risultato del tutto simile a quello simulato ed è stato in grado di effettuare tutte le procedure indicate nel paragrafo 4.1.

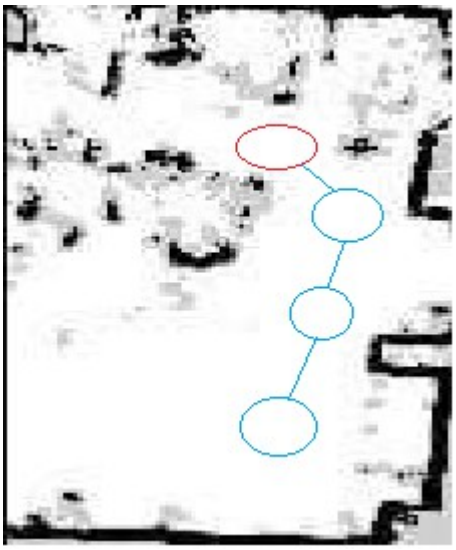

Figura 4.2.1 Mappa semantica del test eseguito nel laboratorio RoCoCo della Sapienza. I nodi il blu sono i nodi utilizzati come passaggio mentre il nodo in rosso il nodo significativo nella mappa semantica, nel nostro caso la cucina.

Capitolo 5

Conclusioni

Nello svolgimento di questa Tesi, sviluppata nell'ambito della Laurea in Ingegneria Informatica dell'Università la Sapienza, si è affrontato uno degli argomenti più studiati della robotica odierna: l'esplorazione dei ambienti e la costruzione di mappe semantiche.
Si è inizialmente spiegato come il sistema ROS sia di fondamentale importanza per lo sviluppo di robot e qual' è la sua struttura. Si è visto nel dettaglio come è strutturato e quali sono stati i tool disponibili in ROS che sono stati usati nello svolgimento di questo progetto. Successivamente si è specificato cosa sia una mappa semantica e di cosa abbiamo bisogno per realizzarla, descrivendo quindi mappe metriche, i vari tipi di strutture per costruire mappe semantiche e i metodi usati per riconoscere gli oggetti presenti nella mappa e la sua struttura. Infine è stato spiegato nel dettaglio come si è proceduto per la realizzazione del progetto e gli esperimenti effettuati. In questi esperimenti il robot ha dimostrato di essere in grado di creare una mappa semantica dell'ambiente circostante, qualunque esso sia, basandosi sulla mappa metrica ad esso associato.

Questa tesi pone le basi per lo sviluppo di mappe semantiche ma può essere ampliata notevolmente in sviluppi futuri. Una modifica che potrebbe essere apportata al sistema sarebbe quella di affiancare al già presente sistema di riconoscimento vocale su cui si basa attualmente, un sistema di Object detection, cioè di riconoscimento visivo, del tutto simile a quello mostrato nella sezione 3.3. Questo sistema permetterebbe una interazione con l'ambiente ancora migliore, rendendo il sistema indipendente anche dalla presenza dell'utente in quanto sarebbe in grado di riconoscere ed etichettare oggetti e stanze della mappa in modo del tutto autonomo. Un'ulteriore modifica sarebbe quello di staccare il sistema dalla dipendenza del grafo, rendendolo in grado di muoversi liberamente all'interno dell'intera mappa e trovare percorsi all'interno di essa senza alcuna restrizione.

I possibili sviluppi di tecnologie simili a questa sono vasti e ancora ampiamente da scoprire. La ricerca in questo ambito è sempre più attiva e prolifica, Google sta sviluppando i suoi prototipi di macchine autonome così come anche la General Motors. Affianco a questo ci sono sviluppi industriali e domestici che seguono il trend in forte espansione. È quindi un dato di fatto che lo sviluppo tecnologie di questo tipo, basate sullo sviluppo di sistemi in grado di rendere i robot autonomi e di svolgere compiti sempre più complessi, sia una delle grandi sfide della robotica moderna.

Bibliografia

[1] Andrea Tartaglia, “Autoblog”, [Online]. Avaible: https://www.autoblog.it/post/965623/guida-autonoma-waymo-google-trasforma-auto-normali-in-driveless-car

[2] Wikipedia, “Autovettura Autonoma”, [Online]. Avaible: https://it.wikipedia.org/wiki/Autovettura_autonoma.

[3] “Hurolife”, [Online]. Avaible: https://www.hurolife.it/robot-mobili-autonomi-e-a-guida-autonoma-la-differenza-tra-amr-e-agv/)

[4] Mobile Industrial Robots, “AGV vs. AMR” [Online]. Avaible: https://www.mobile-industrial-robots.com/en/resources/whitepapers/agv-vs-amr-whats-the-difference/

[5] Wikipedia, ”Robot Operating System” [Online]. Avaible: https://it.wikipedia.org/wiki/Robot_Operating_System

[6]“ROS.org”, [Online].: https://www.ros.org/history/

[7] W. Garage, “ROS platform”, April 2013. [Online]. Available: http://www.willowgarage.com/pages/ software/ros-platform.

[8] “ROS.org”, [Online]. Available: http://wiki.ros.org/rosbuild/ROS/Concepts.

[9] “Player Project”, [Online]. Avaible: http://playerstage.sourceforge.net/

[10] “ROS.org”, [Online]. Avaible: http://wiki.ros.org/stage_ros

[11] Avaible: http://wiki.ros.org/roslaunch

[12] Avaible: http://wiki.ros.org/amcl

[13] R. Basili, E. Bastianelli, G. Castellucci, D. Croce, D. Nardi e A. Vanzo, “LU4R Project adaptive spoken Language Understanding For Robots”, [Online]. Available: http://sag.art.uniroma2.it/lu4r.html.

[14] E. Bastianelli, D. D. Bloisi, R. Capobianco, F. Cossu, G. Gemignani, L. Iocchi, D. Nardi, On-line Semantic Mapping, Sapienza University of Rome, Italy.

[15] Rizwan Aslam Butt , Syed M Usman Ali, Semantic Mapping and Motion Planning with Turtlebot Roomba , Karachi, Pakistan 2013.

[16] Xavier Gallart Del Burgo, "Semantic Mapping in ROS", Master Thesis. Stockolm, Sweden 2013.

[17] P. Jensfelt, Approaches to Mobile Robot Localization in Indoor Environments, Stockholm, Sweden: PhD thesis, 2001.

[18] Nico Blodow, Lucian Cosmin Goron, Zoltan-Csaba Marton, Dejan Pangercic, "Autonomous Semantic Mapping for Robots Performing Everyday Manipulation Tasks in Kitchen Environments", Munchen.

[19] Joshua S. Lum ," UTILIZING ROBOT OPERATING SYSTEM (ROS) IN ROBOT VISION AND CONTROL ", California 2015.

[20] Berkeley, "FrameNet",[Online]. Avaible: https://framenet.icsi.berkeley.edu/fndrupal/