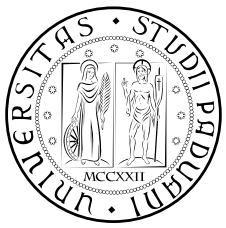

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI TECNICA E GESTIONE DEI SISTEMI INDUSTRIALI
CORSO DI LAUREA TRIENNALE IN INGEGNERIA MECCATRONICA

TESI DI LAUREA TRIENNALE

ROS2: studio di strumenti open-source per l'analisi e visualizzazione di dati eterogenei in ambito robotico

Relatrice: Monica Reggiani

Correlatore: Mattia Guidolin

Laureanda: Chiara Fanello
1216904-IMC

ANNO ACCADEMICO: 2022-2023

SOMMARIO

Negli ultimi anni, la diffusione di robot collaborativi ha portato gli sviluppatori ad avere bisogno di un software fortemente modulare che permetta di progettare sistemi complessi. Due studenti dell'università di Standford si misero a creare un software che potesse aiutare per tali studi.

Creato su duplice linguaggio (Python e C++), ROS è diventato uno dei software più usati in ambito industriale grazie alla sua facilità e velocità di sviluppo, inoltre essendo open-source diventa utilizzabile sia in ambito commerciale che non.

Con lo sviluppo di ROS2, molti dei difetti di ROS sono stati corretti principalmente per quanto riguarda l'elaborazione in real-time e la sicurezza nelle comunicazioni.

ROS2 dispone di molti strumenti open-source per l'analisi dei dati dei progetti. Da la possibilità di avviare, controllare e monitorare un robot reale o una sua simulazione tramite comandi testuali. Permette di visualizzare la comunicazione tra i vari nodi oltre che salvare dati di esecuzione per poterli rieseguire separando l'implementazione hardware da quella software.

È inoltre possibile utilizzare una simulazione 2 o 3D che permette di rappresentare graficamente l'andamento dei parametri di interesse o di servirsi di un'interfaccia grafica che consente di visualizzare il proprio robot e i parametri dei vari sensori.

In questa tesi verranno analizzate le principali differenze tra le varie versioni di ROS e i principali comandi utilizzabili per poter sfruttare a pieno le potenzialità di ROS, sia utilizzando istruzioni a linea di comando che ambienti grafici.

INDICE

1 Introduzione a ROS 1
- 1.1 Evoluzione di ROS 1
- 1.2 ROS, perchè usarlo? 4
- 1.3 Nomenclatura 5
- 1.4 ROS1 VS ROS2 6
  - 1.4.1 Sicurezza nel protocollo rete 7
  - 1.4.2 Comunicazione tra nodi 8
  - 1.4.3 Coesistenza di più nodi su un singolo processo 8
  - 1.4.4 Unificazione del linguaggio per l'implementazione delle librerie 9

2 Command Line Interface di ROS2 11
- 2.1 Principali comandi 12
  - 2.1.1 Comandi per i Node 12
  - 2.1.2 Comandi per i topic 13
  - 2.1.3 Comandi per i service 13
  - 2.1.4 Comandi per le bag 14
  - 2.1.5 Comandi per le action 15

3 Graphic User Interface di ROS2 17
- 3.1 RQt 17
- 3.2 RVIZ2 20

Conclusioni 25

Bibliografia 27

ELENCO DELLE FIGURE

Figura 1.1 Logo di ROS 1
Figura 1.2 Robot PR1, immagine di IEEE Spectrum 2
Figura 1.3 Esempio di simulazione con Gazebo 3
Figura 1.4 Logo della distribuzione Humble 5
Figura 1.5 Componenti base per la comunicazione tra nodi in ROS 7
Figura 1.6 Esempio di comunicazione con terminazione di Ros Master 8
Figura 2.1 Esempio di una CLI di ROS per l'avvio e la gestione di un programma semplice 11
Figura 2.2 Grafo che mostra la comunicazione tra diversi nodi 12
Figura 2.3 Comunicazione tra un node publisher e due node subscriber per mezzo di un topic 13
Figura 2.4 Comunicazione call-end-response per mezzo di un service 14
Figura 3.1 Esempio di collegamenti tra nodi e topic con il comando plugin > Introspection > Node Graph 18
Figura 3.2 Esempio di messaggi di log(1) 19
Figura 3.3 Esempio di grafico del parametro x di Turtlesim 19
Figura 3.4 Esempio di visualizzazione della schermata principale di rviz con la configurazione View Robot 21
Figura 3.5 Esempio di visualizzazione con flag attivi dell'orientamento, nome e posizione del frame 24

ACRONIMI

ROS	Robot Operating System
BSD	Berkeley Software Distribution
GPL	General Public License
RPC	Remote Procedure Call
DDS	Data Distribution Service
CLI	Command Line Interface
API	Application Programming Interface
OSRF	Open Source Robotic Foundation
EOL	End-Of-Life

GUI Graphic User Interface

URDF Unified Robotics Description Format

TF Transform

1 INTRODUZIONE A ROS

ROS (Robot Operating System) [5] è un middleware-operating system per lo sviluppo di programmi per robot.

Fornisce servizi tipici di un vero e proprio sistema operativo quali astrazione dell'hardware, gestione di processi, pacchetti e dispositivi; arricchendolo con elementi tipici del middleware quale l'infrastruttura per la comunicazione tra processi o macchine differenti con strumenti di utilità per lo sviluppo, il debugging e la simulazione. ROS è pensato per operare in simbiosi con sistemi operativi Linux, in particolare le principali distribuzioni su cui viene portato avanti lo sviluppo sono Ubuntu e Debian. La compatibilità di ROS è continuamente in espansione grazie ai numerosissimi pacchetti che ne ampliano la portabilità, spesso sviluppati e supportati dalla community stessa.

1.1 EVOLUZIONE DI ROS

ROS è nato come progetto personale di Keenan Wyrobek ed Eric Berger (studenti della Standford University e creatori dello Standford Personal Robot Program) con lo scopo di creare un software unico per la robotica[8]. I problemi principali della ricerca robotica erano:

- Il troppo tempo speso per reimplementare l'infrastruttura software necessaria per costruire algoritmi robotici complessi (driver per sensori ed attuatori, comunicazione tra programmi diversi nello stesso robot);
- Il troppo poco tempo a disposizione per costruzione di programmi robotici intelligenti basati su tale infrastruttura.

All'interno di ogni organizzazione di sviluppo, driver e sistemi di comunicazione venivano implementati nuovamente per ogni progetto. Per questo, Eric e Keenan idearono nel 2006 un programma chiamato Stanford Personal Robotics Program, con l'obiettivo di costruire un framework che permettesse ai processi di comunicare tra loro, oltre ad alcuni strumenti che aiutassero a creare codice su di esso, risolvendo i due problemi precedenti. All'inizio questo framework doveva essere utilizzato per programmare il Personal Robot (PR, Fig.1.2), un robot che avrebbero realizzato come banco di prova.

Figura 1.1: Logo di ROS

Figura 1.2: Robot PR1, immagine di IEEE Spectrum[8]

L'idea era quella di costruirne dieci per poi fornirli alle università e fare in modo che potessero sviluppare software basato sul loro sistema[12].

Mentre erano a Stanford, Keenan ed Eric ricevettero un finanziamento di 50.000 dollari ma fu sufficiente solo per costruire un robot che dimostrasse la potenzialità del loro progetto. Si resero quindi conto che per costruire un sistema davvero universale e fornire quei robot ai gruppi di ricerca, avrebbero avuto bisogno di ulteriori finanziamenti e per questo iniziarono a rivolgersi agli investitori.

Nel 2008, Keenan ed Eric incontrarono Scott Hassan, investitore e fondatore di Willow Garage, un centro di ricerca specializzato in prodotti robotici. Scott trovò la loro idea così interessante che decise di finanziarla e di avviare con loro un programma di robotica personale all'interno di Willow Garage. Nacque così il Robot Operating System e con esso il robot PR2. In realtà, il progetto ROS divenne così importante che tutti gli altri progetti di Willow Garage furono scartati per concentrarsi solo sullo sviluppo e la diffusione di ROS.

Nel 2009 fu rilasciata la prima distribuzione: ROS Mango Tango, chiamata anche ROS 0.4. La versione 1.0 di questa distribuzione fu lanciata quasi un anno dopo, nel 2010. Da quel momento, il team ROS decise di chiamare le distribuzioni con nomi di tartarughe. Finora, sono state create e rilasciate le seguenti distribuzioni:

- Box Turtle (2010)
- ROS C-Turtle (2010)
- Diamond Back (2011)
- ROS Electric Emys (2011)

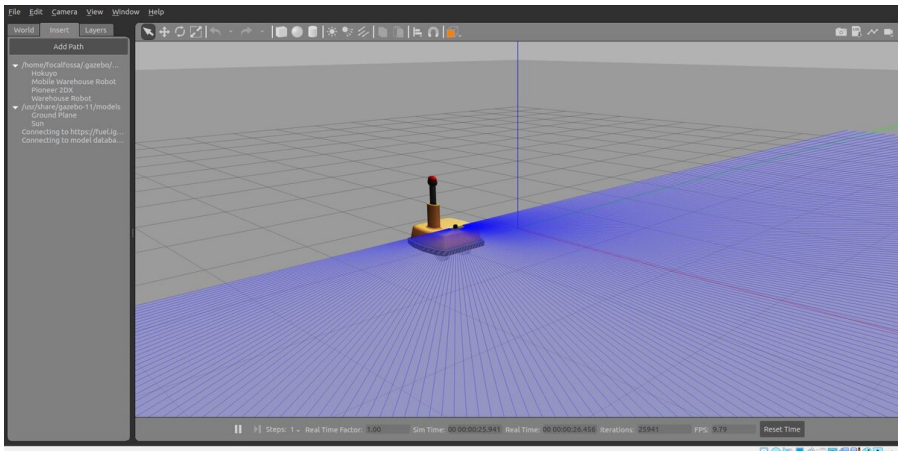

Figura 1.3: Esempio di simulazione con Gazebo[4]

- ROS Fuerte Turtle (2012)
- ROS Groovy Galapagos (2012)

Nel 2010 furono costruiti 11 robot PR2 e messi a disposizione delle università per lo sviluppo di software robotico con ROS come previsto dall'idea originale di Eric e Keenan. La simulazione, però, iniziò a diventare molto importante, nello specifico quella 3D, e per questo motivo il team decise di incorporare Gazebo, il simulatore di robotica 3D del progetto Player/Stage[1], facendone il predefinito per ROS (Fig1.3).

Un altro evento importante che ha aumentato le dimensioni della comunità ROS è stato l'annuncio, nel novembre 2010, da parte di Willow Garage, del rilascio del robot Turtlebot, che andò a sostituire il costoso PR2[11]. Turtlebot è un robot semplice ed economico che permette a chiunque di sperimentare le basi della robotica e di ROS. In breve tempo ha riscosso un grande successo tanto da essere utilizzato ancora oggi, nelle versioni Turtlebot2 e Turtlebot3.

Nel 2013, Willow Garage è stata chiusa e la Open Source Robotics Foundation ha assunto la guida dello sviluppo di ROS. Sotto la nuova gestione legale della OSRF, sono state sviluppate e rilasciate nuove distribuzioni:

- ROS Hydro Medusa (2013)
- ROS Indigo Igloo (2014)
- ROS Jade Turtle (2015)
- ROS Kinetic Kame (2016)
- ROS Lunar Loggerhead (2017)
- ROS Melodic Morenia (2018)
- ROS Noetic Ninjemys (2020)

1 progetto che crea software libero che consente la ricerca nei sistemi di robot e sensori, ultima release nel 2010[9]

Per ogni distribuzione è stata assegnata una data di EOL (End-Of-Life) ad indicare la fine sel supporto ed aggiornamento. Di quelle sopracitate, sono ancora mantenute Melodic (fino a maggio 2023) e Noetic (fino a maggio 2025).

Intorno al 2015, le carenze di ROS per le applicazioni industriali si sono manifestate molto chiaramente. Il singolo punto di failure, la mancanza di sicurezza e l'assenza di supporto in tempo reale erano alcune delle principali problematiche per cui le aziende faticavano a introdurre ROS nei loro prodotti. Tuttavia, era chiaro che, se ROS avesse vovuto diventare lo standard per la robotica, avrebbe dovuto raggiungere il settore industriale con una voce più forte rispetto a quella delle poche aziende pioniere che già lo utilizzavano nei loro prodotti.

Per superare questo punto, l'OSRF si impegnò per sviluppare ROS2, con lo scopo di migliorare le limitazioni di ROS1 e favorire lo sviluppo aziendare. Anche per tale versione sono state rilasciate a partire dal 2015 diverse distribuzioni quali:

- alpha1 - alpha5 (2015)
- beta1 (2016)
- beta2 e beta3 (2017)
- Ardent Apalone (2017)
- Bouncy Bolson (2018)
- Crystal Clemmys (2018)
- Dashing Diademata (2019)
- Eloquent Elusor (2019)
- Foxy Fitzroy (2020)
- Galactic Geochelone (2021)
- Humble Hawksbill (2022) (fig. 1.4)
- Iron Irwini (maggio 2023)

Di queste distribuzioni sono ancora aggiornate Galactic (EOL maggio 2023) e Humble (EOL maggio 2027).

1.2 ROS, PERCHÈ USARLO?

ROS ha come obiettivo quello di differenziarsi dai principali framework pensati per la robotica a seguito della sua facilità e rapidità di sviluppo. Questo è reso possibile dal suo design fortemente modulare che si presta bene al riutilizzo del codice. I pacchetti che compongono il software sono riutilizzabili e possono essere condivisi in altri progetti, a patto di rispettarne le dipendenze.

Figura 1.4: Logo della distribuzione Humble

ROS è pensato anche per essere utilizzato all'interno di sistemi distribuiti. Numerosi software ai giorni nostri sono basati sulla cooperazione di processi che possono anche non risiedere sulla stessa macchina: alcuni robot hanno integrati più computer che controllano sensori e attuatori diversi, uno stesso task può essere raggiunto con la coordinazione di più robot che devono agire insieme per risolvere il problema. È semplice notare la necessità di poter comunicare tra processi, questa comunicazione asincrona viene permessa dal middleware.

ROS è distribuito secondo i termini della licenza BSD[2], che consente lo sviluppo di progetti sia commerciali che non. Consentendo comunicazioni inter-processo, i sistemi costruiti attorno a ROS possono utilizzare svariate licenze per i vari componenti: i singoli moduli possono incorporare software protetti da varie licenze, dalla GPL[3] alla BSD, fino a quelle proprietarie.

1.3 NOMENCLATURA

Prima di andare avanti, è bene fissare qualche concetto base di ROS per comprendere appieno le sue funzionalità[6]:

- *Node (Nodo)*: un nodo è un processo che compie una qualsiasi attività all'interno del sistema ROS. Tali sistemi comprendono numerosi nodi che possono essere interpretati come moduli software, ognuno dei quali incaricato di gestire un aspetto comportamentale del robot (parte decisionale, movimento, azionamento dei motori, ...). Ogni nodo in esecuzione dispone di un nome univoco che lo identifica nel resto del sistema.

2 Famiglia di licenze software libere-permissive, impongono restrizioni minime sull'uso e distribuzione del software coperto

3 Licenza copyleft, ogni versione di ciascun programma deve rimanere libera e consultabile

- *Message (Messaggio)*: un messaggio è una semplice struttura di dati, composta da campi tipizzati. Sono supportati i tipi primitivi standard (interi, virgola mobile, booleani, ecc.) e gli array di tipi primitivi. I messaggi possono includere strutture e array arbitrariamente annidati e vengono utilizzati per la comunicazione tra i vari nodi.

- *Topic*: i topic costituiscono il mezzo di comunicazione asincrono, unidirezionale, per lo scambio di messaggi tra nodi, secondo una semantica di tipo publish/subscribe. Ci possono essere più publisher e subscriber concorrenti per un singolo topic, e uno specifico nodo può pubblicare e/o sottoscriversi a più topic. Ogni topic è fortemente tipizzato dal tipo di messaggio che viene pubblicato, e i nodi possono ricevere solo messaggi il cui tipo sia compatibile.

- *Service (Servizio)*: il servizio rappresenta il mezzo di comunicazione sincrono secondo una semantica di tipo request / reply, implementando in ROS funzionalità di RPC. Andrà cosi a definirsi nella rete nodi che svolgeranno funzione di service provider e nodi client.

- *Action*: l'action consente a un nodo (il client) di inviare una richiesta a un altro nodo (il service provider) per eseguire un'azione specifica; durante l'esecuzione dell'azione, il server invierà costantemente un feedback al client con dettagli sull'avanzamento dell'azione in corso. Una volta completata l'azione, il server invierà al client un risultato con l'esito finale dell'azione.

- *Bag*: la bag consente di registrare dei dati pubblicati su dei topic nel sistema accumulandoli e salvandoli su un database, dando la possibilità di riprodurli nuovamente per riottenere i risultati dei test.

- *Package (Pacchetto)*: in ROS il software viene organizzato in package, che costituiscono dei moduli. Ogni package può contenere un nodo ROS, un file di configurazione, un dataset, una libreria, software di terze parti. Lo scopo dei packages è ottenere una maniera efficace per il riuso del codice, venendo strutturati in modo tale da risultare funzionali ma non troppo pesanti e difficili da utilizzare.

In Fig.1.5 è possibile visualizzare le relazioni tra i vari componenti per la comunicazione tra nodi.

1.4 ROS1 VS ROS2

Come introdotto in precedenza, ROS1 è stato progettato e rilasciato con la necessità di velocizzare la ricerca sulla robotica. Con il tempo ha guadagnato molta popolarità ed è così divenuto la principale piattaforma per la sperimentazione. ROS1 non è mai stato pensato come software ad uso industriale e quindi aspetti come sicurezza, topologia

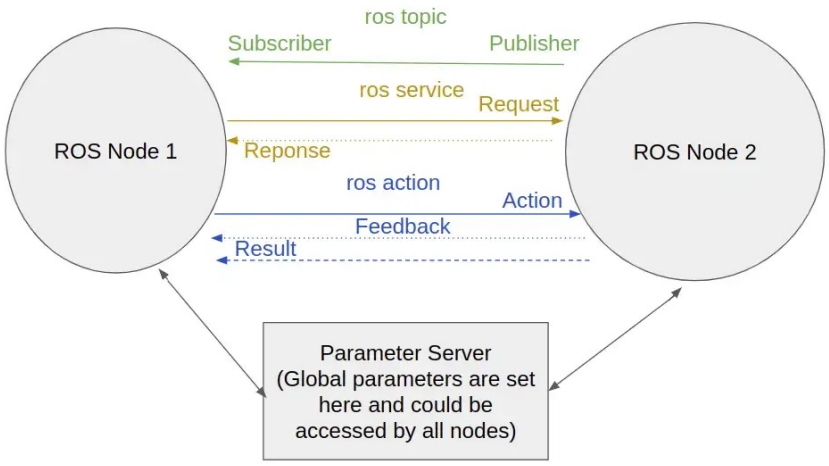

Figura 1.5: Componenti base per la comunicazione tra nodi in ROS[3]

di rete e tempo di attività del sistema non erano stati considerati essenziali. Con l'espansione di ROS anche nell'ambito industriale, molte delle principali mancanze sono diventate sempre più evidenti. Da questo è arrivato il bisogno di ricostruirlo dalla base tenendo conto delle seguenti esigenze[3]:

- Sicurezza: deve essere sicuro, con un'adeguata crittografia dove necessario.
- Sistemi embedded: ROS2 deve essere in grado di funzionare su sistemi embedded.
- Reti diversificate: deve essere in grado di funzionare e comunicare attraverso vaste reti di robot, dalla LAN ai collegamenti multisatellitari.
- Elaborazione in tempo reale: deve essere in grado di eseguire calcoli in tempo reale.
- Prontezza del prodotto: deve essere conforme agli standard industriali e di sicurezza pertinenti, in modo da essere pronto per il mercato.

Esaminando quindi le principali differenze tra ROS2 e ROS1 si può notare come ROS2 abbia risolto i principali difetti di ROS1.

1.4.1 *Sicurezza nel protocollo rete*

I progettisti di ROS2 hanno deciso di modificare il protocollo di rete passando al DDS[2] (Data Distribution Service) per tutte le comunicazioni che avvengono internamente a ROS2. Fornisce le garanzie di sicurezza e l'affidabilità necessarie per mantenere una buona comunicazione in aree con connessioni deboli o con grandi probabilità di perdite di segnale (collegamenti wifi o satellitari). Questo è in contrasto con il protocollo personalizzato TCP sviluppato in ROS1, che

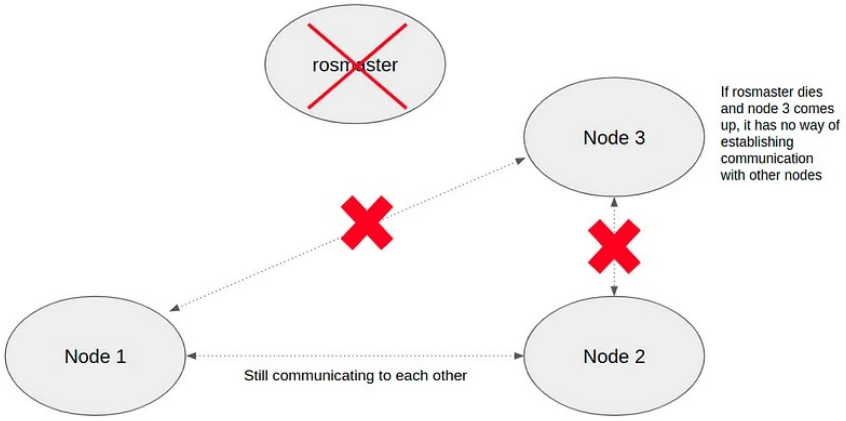

Figura 1.6: Esempio di comunicazione con terminazione di Ros Master

manca di molte delle garanzie di sicurezza e affidabilità fornite da DDS.

1.4.2 *Comunicazione tra nodi*

In ROS1, esiste un nodo chiamato ROS Master, che fornisce servizi di denominazione e registrazione per i nodi, inoltre funge anche da mediatore per stabilire le connessioni tra i nodi. Se ROS master termina, i nodi connessi ad esso continueranno a comunicare tra loro; tuttavia, se un nuovo nodo si presenta e tenta di comunicare con un nodo già connesso, non avrà modo di farlo perché ROS master, che dovrebbe mediare la connessione, è stato interrotto. Questo può portare a nodi orfani nello stack, che non hanno modo di comunicare con altri nodi, come per esempio in Fig.1.6. In ROS2, il ROS master è stato eliminato e ciò è stato possibile perché ROS2 utilizza DDS, che permette ai nodi di comunicare tra loro in modo peer-to-peer senza la necessità di un mediatore. Questo aumenta la tolleranza ai guasti del sistema, in quanto ora non c'è un singolo punto di guasto.

1.4.3 *Coesistenza di più nodi su un singolo processo*

In ROS1, i nodi ROS sono stati originariamente sviluppati per essere eseguiti su un singolo processo. In seguito sono stati introdotti i nodelets di ROS1, che consentono l'esecuzione di più nodi sullo stesso processo (ogni nodo è compilato come libreria condivisa e caricato in fase di esecuzione da un processo contenitore). L'esecuzione dei nodi su un singolo processo è utile se si eseguono operazioni che richiedono molta memoria (che risulta essere condivisa) o una comunicazione tra nodi veloce. I nodi di ROS2 funzionano intrinsecamente in modo simile ai nodelet di ROS1, consentendo ai nodi di essere eseguiti sul processo, riducendo così la latenza della comunicazione tra i nodi. In più, il ciclo di vita dei nodi in ROS2 può essere gestito; ad esempio, è possibile modellare un nodo come una macchina a stati e può essere gestito da un sistema esterno in base allo stato attivo dello stesso.

1.4.4 *Unificazione del linguaggio per l'implementazione delle librerie*

In ROS1, ciascuna delle librerie client (come roscpp o rospy) è completamente scritta nei propri linguaggi, con la conseguenza della gestione e manutenzione di due librerie parallele. Questo comporta, ad esempio, che l'implementazione sottostante di un publisher in rospy è diversa da quella di roscpp, introducendo variazioni nelle prestazioni a seconda del linguaggio utilizzato. In ROS2, tutte le librerie client condividono ora un'implementazione comune scritta in C chiamata rcl, la quale si colloca tra le librerie client e le interfacce DDS necessarie per la comunicazione, facilitando anche la capacità degli sviluppatori di creare nuove librerie client per nuovi linguaggi.

CONCLUSIONI ROS2 è un middleware operating-system sviluppato per sopperire ad alcune mancanze di ROS1 quali sicurezza, unificazione del linguggio, creazione di comunicazione tra nodi senza aver bisogno di un intermediario. E' stato creato con lo scopo di facilitare e velocizzare lo sviluppo di nuovi robot collaborativi, pertanto si è scelto un design fortemente modulare che permettesse il riutilizzo del codice in diversi progetti.

Nei prossimi capitoli si andranno ad analizzare i principali strumenti che ROS2 fornisce per lo studio e l'analisi di dati in ambito robotico, inizialmente in formato testuale mediante la CLI tools e infine con una interfaccia grafica per mezzo di RQt e RViz2.

2

COMMAND LINE INTERFACE DI ROS2

Abbreviato come CLI, la Command Line Interface è un'interfaccia utente testuale che viene utilizzata per eseguire programmi, gestire file e interagire con il sistema. Il comando viene digitato su una riga specifica seguendo un prompt visivo dal computer e il sistema esegue il comando inserito visualizzando a schermo l'output. Attraverso questa interazione di input e output, l'utente è in grado di emettere una sequenza di comandi, che vengono eseguiti dal sistema o dal programma.

Parlando nello specifico di ROS, la *CLI tools* è un insieme di comandi per avviare, ispezionare, controllare e monitorare un robot[10]. Si ispira molto alla filosofia Unix, che è un insieme di norme culturali e approcci filosofici allo sviluppo software minimalista e modulare. Si basa sull'esperienza dei principali sviluppatori del sistema operativo Unix che furono importanti nel portare i concetti di modularità e riusabilità nella pratica dell'ingegneria del software, generando un movimento di "strumenti software". Tale filosofia enfatizza la creazione di codice semplice, compatto, chiaro, modulare ed estensibile, il quale può essere facilmente mantenuto e riproposto da sviluppatori diversi dai suoi creatori.

Si interfaccia direttamente con le API di ROS2 che sono semplicemente un'implementazione dei modelli di alto livello.

La CLI ha una sintassi similare alla maggior parte dei linguaggi, ogni comando inizia con *ros2* seguito dalla sintassi vera e propria del comando. Essendo ispirato a sistemi Unix, si ha la possibilità di completare il comando tramite il tasto *TAB* e di avere la documentazione del comando tramite l'aggiunta di *--help* alla fine dello stesso.

Il comando principale è *ros2 run <package-name> <executable-name> <args>* che ha lo scopo di eseguire un nodo di uno specifico pacchetto.

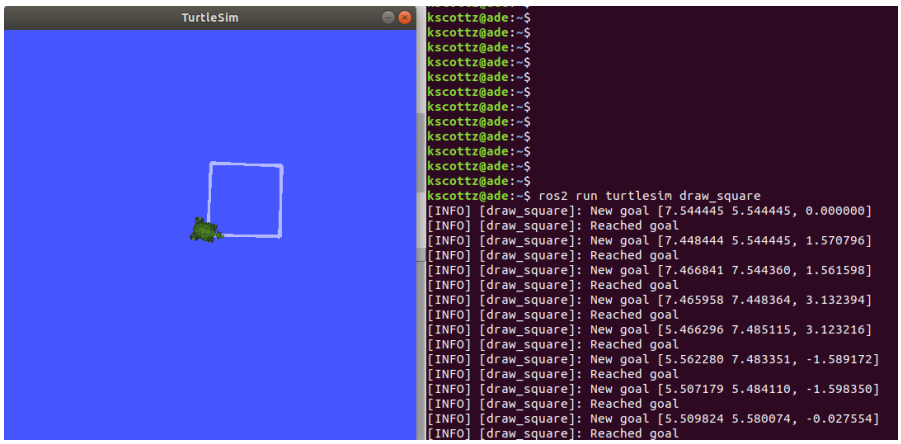

Figura 2.1: Esempio di una CLI di ROS per l'avvio e la gestione di un programma semplice

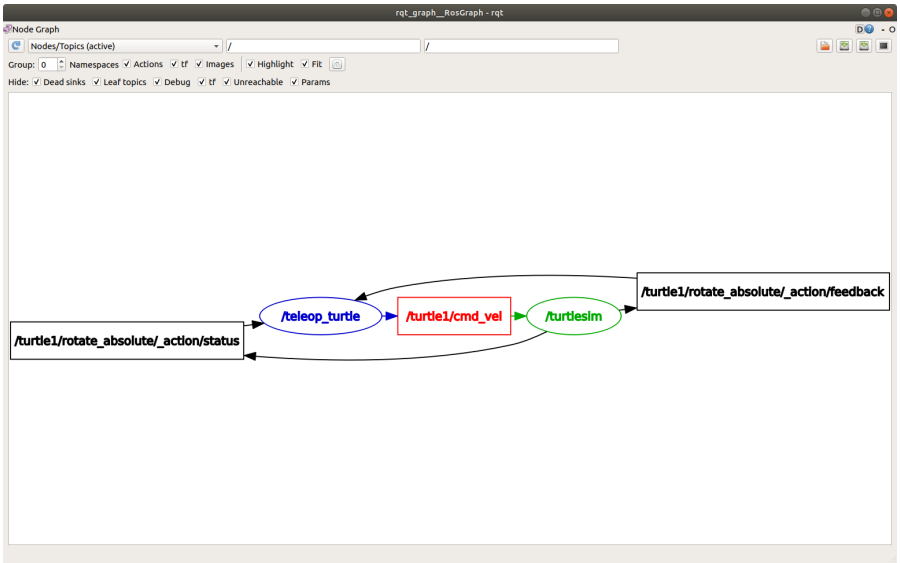

Figura 2.2: Grafo che mostra la comunicazione tra diversi nodi

La vera potenza di ROS non sta nel fatto che può eseguire un programma, ma che può eseguire molti programmi contemporaneamente, che comunicano tra loro per controllare un robot, o più robot, che lavorano insieme.

2.1 PRINCIPALI COMANDI

Di seguito verranno illustrati i principali comandi per il controllo e la gestione dei sistemi in ROS tramite la CLI.

2.1.1 *Comandi per i Node*

Come accennato in precedenza, ogni nodo in ROS2 deve essere responsabile di un singolo scopo (ad esempio, un nodo per controllare i motori delle ruote, un nodo per controllare un telemetro laser, ecc.) Ogni nodo può inviare e ricevere dati ad altri nodi tramite topic, service, action o parameter. Molti nodi che lavorano contemporaneamente formano un sistema robotico completo. In ROS2, un singolo eseguibile può contenere uno o più nodi. Il grafo ROS è una rete di elementi che elaborano dati in simultanea e comprende tutti gli eseguibili e le connessioni tra di essi, come per esempio in Fig.2.2. I comandi disponibili per i nodi sono:

ros2 node list Questa istruzione mostra i nomi di tutti i nodi in esecuzione. Risulta essere particolarmente utile quando si vuole interagire con un nodo e si vuole cercare il nome o quando si ha un sistema con molti nodi in esecuzione e occorre tenerne traccia.

ros2 node info <node-name> Tale comando restituisce un elenco di publisher, subscriber, service e action che interagiscono con il nodo <node-name>.

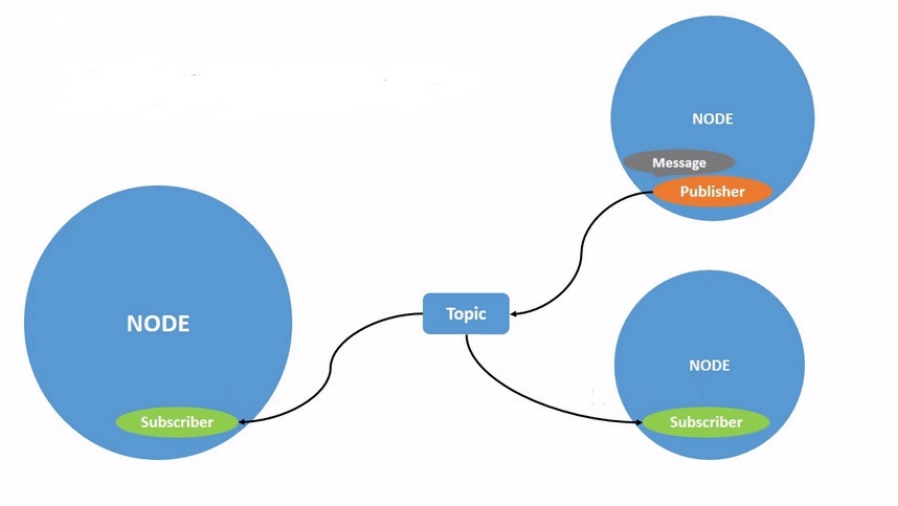

Figura 2.3: Comunicazione tra un node publisher e due node subscriber per mezzo di un topic

2.1.2 *Comandi per i topic*

ROS2 suddivide i sistemi complessi in molti nodi modulari. I topic sono elementi fondamentali del grafo che permettono la creazione di connessioni per lo scambio di messaggi tra i nodi. Un nodo può pubblicare e ricevere dati su un numero qualsiasi di topic, come si può comprendere dalla Fig.2.3, dove un publisher comunica con due subscriber tramite un topic. I topic sono uno dei modi principali per spostare i dati tra i nodi e quindi tra le diverse parti del sistema. Le principali istruzioni disponibili per i topic sono:

ros2 topic list Questa istruzione restituisce un elenco di tutti i topic attualmente attivi nel sistema. E' possibile aggiungere *-t* al comando cosicchè restituisca lo stesso elenco di argomenti con il tipo di topic aggiunto tra parentesi. Tali attributi sono il modo in cui i nodi sanno che stanno parlando delle stesse informazioni quando si spostano su un diverso topic.

ros2 interface show <msg-type> Dopo aver identificato il nome di un messaggio per uno specifico pacchetto, è possibile conoscerne i dettagli, in particolare la struttura dei dati che il messaggio si aspetta tramite tale comando.

ros2 topic pub <topic-name> <msg-type> <args> Avendo identificato la struttura del messaggio, è possibile pubblicare i dati su uno specifico topic con questa istruzione. L'argomento *<args>* è il dato effettivo che si passerà al topic.

2.1.3 *Comandi per i service*

I service sono un altro metodo di comunicazione per i nodi del grafo ROS. Essi si basano su un modello di *call-end-response*, come da Fig.2.4,

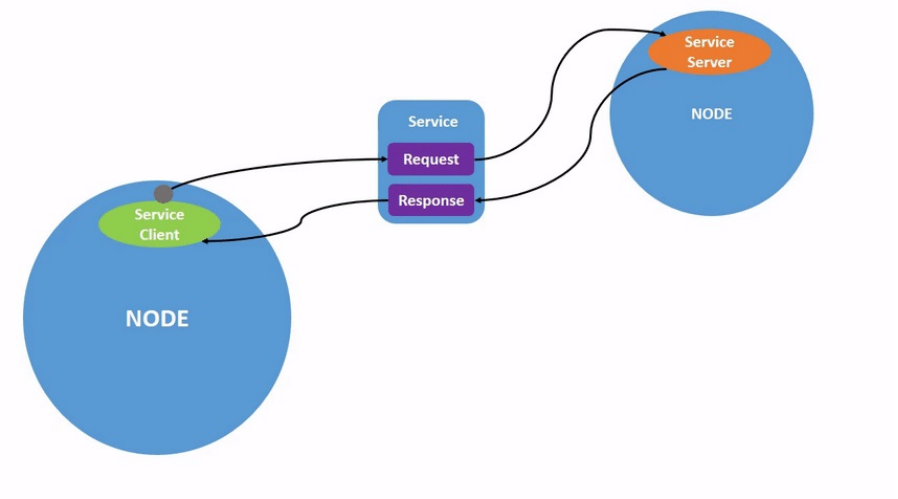

Figura 2.4: Comunicazione call-end-response per mezzo di un service

a differenza del modello publisher-subscriber dei topic.

Mentre quest'ultimi consentono ai nodi di sottoscrivere flussi di dati e di ottenere aggiornamenti continui, i service forniscono dati solo quando vengono chiamati specificamente da un client. Le principali istruzioni disponibili per i service sono:

ros2 service list Tale comando restituisce una lista di tutti i service attivi nel sistema.

ros2 service type <service-name> I service hanno tipi che descrivono come sono strutturati i dati di richiesta e risposta, che sono definiti in modo similare rispetto ai tipi dei topic, con la differenza che i tipi dei service hanno due parti: un messaggio per la richiesta e un altro per la risposta. Tramite questa istruzione posso conoscere il tipo dello stesso.

ros2 service find <type-name> Tale comando è usato per trovare tutti i service per un tipo specifico.

ros2 interface show <type-name> Questa istruzione è usata per conoscere la struttura degli argomenti in ingresso su un service.

ros2 service call <service-name> <service-type> <args> Con tale comando è possibile richiamare un determinato service.

2.1.4 *Comandi per le bag*

Ipotizziamo di utilizzare un robot e di sviluppare un algoritmo con lo scopo di rilevare ed evitare gli ostacoli. Lavorando con sistemi reali alcune condizioni esterne possono essere diverse ad ogni test, facendo diventare difficile il confronto di algoritmi diversi. Con le bag di ROS2, è possibile far funzionare il robot, registrare un campione (bag) di ciò che sta accadendo e poi riutilizzare questo campione più volte.

Inoltre, è possibile riprodurre una bag ROS2 in modo da poter saltare l'avvio di alcune parti dell'applicazione del robot oppure per separare l'implementazione hardware da quella software. Le principali istruzioni disponibili per le bag sono:

ros2 bag record <topic-name> Questa istruzione è utile per registrare i dati pubblicati su un topic. Nel caso si vogliano raccogliere dati da più topic contemporaneamente, si può usare il comando stesso con in aggiunta il parametro *-o subset* e l'elenco dei topic interessati.

ros2 bag info <bag-name> Con tale istruzione si può vedere il dettaglio nella bag in esecuzione.

ros2 bag play <bag-name> Questo comando crea un publisher per ogni topic registrato.

2.1.5 *Comandi per le action*

Le action sono uno dei tipi di comunicazione di ROS 2 e sono destinate a compiti di lunga durata. Sono composte da tre parti: un obiettivo, un feedback e un risultato.

Le action si basano su topic e service. La funzionalità è simile a quella dei service, tranne per il fatto che le azioni sono annullabili durante l'esecuzione. Inoltre, forniscono un feedback costante, a differenza dei service che restituiscono una sola risposta.

Le action utilizzano un modello client-server, simile al modello publisher-subscriber. Un nodo "action client" invia un obiettivo a un nodo "action server" che lo riconosce e restituisce un flusso di feedback e un risultato.

ros2 action list Tale istruzione mostra una lista di tutte le action.

ros2 interface show <action-name> Questo comando è usato per conoscere la struttura delle action.

ros2 action send-goal <action-name> <action-type> <value> Tale istruzione restituisce il risultato dell'action.

CONCLUSIONE ROS2 fornisce un'insieme di comandi atti ad avviare e monitorare un robot. Risulta essere molto utile per il controllo delle connessioni e comunicazioni tra nodi in modesti progetti, essendo solamente un interfaccia testuale risulta complesso l'utilizzo per sistemi di grandi dimensioni.

3

GRAPHIC USER INTERFACE DI ROS2

La Graphic User Interface (GUI), è un'interfaccia attraverso la quale un utente interagisce con delle macchine attraverso l'uso di icone, menu e altri indicatori o rappresentazioni grafiche. Le GUI visualizzano graficamente le informazioni e i relativi controlli utente, a differenza delle interfacce basate su testo (e.g. CLI), in cui dati e comandi sono rigorosamente in formato testo. Le GUI sono state create per essere abbastanza intuitive in modo da essere utilizzate anche da personale relativamente non qualificato senza conoscenza pregressa di alcun linguaggio di programmazione. Diventate lo standard nella programmazione di applicazioni software, il loro design è sempre più incentrato sull'utente che potrà lavorare su progetti complessi con più semplicità.

Nello specifico di ROS, la GUI è un sistema che consente agli utenti di interagire e addentrarsi all'interno dell'ambiente ROS in modo visivo ed efficace fornendo strumenti pre-esistenti con lo scopo di incoraggiare a sviluppare e contribuire alla comunity. La GUI ROS è progettata come un'architettura che consente agli utenti di implementare rapidamente plug-in grafici basati su Qt[1].

3.1 RQT

RQt è un framework per interfacce grafiche che implementa vari strumenti e interfacce sottoforma di plugin e che consente di visualizzare, analizzare e interagire con i dati e i componenti del sistema robotico in modo visuale. Gli strumenti possono ancora essere eseguiti con un metodo tradizionale indipendente, ma RQt rende più facile la gestione di tutte le finestre in un'unica schermata[6].

I plugin di tale interfaccia sono classificati in diverse categorie, tra cui *Topics*, *Services*, *Actions*, *Logging*, *Robot Tools*, *Visualization* e molti altri. Ciascuna categoria fornisce una serie di modelli aggiuntivi specifici che consentono di interagire con i dati e i componenti del sistema robotico in modo specifico.

Ad esempio, la categoria Topics fornisce plugin per la visualizzazione dei dati sui topic ROS2, come ad esempio *rqt-topic*, che consente di visualizzare i messaggi trasmessi su un topic specifico in tempo reale. La categoria Services fornisce componenti aggiuntivi per l'interazione con i servizi ROS2, come ad esempio *rqt-service-caller*, che consente di chiamare un servizio ROS2 e di visualizzarne la risposta.

La categoria Robot Tools fornisce plugin specifici per la manipolazione dei dati dei robot, come ad esempio *rqt-joint-trajectory-controller*, che consente di creare e gestire i profili di traiettoria per i controllori

1 framework applicativo multipiattaforma ampiamente utilizzato per lo sviluppo di software applicativo con un'interfaccia utente grafica

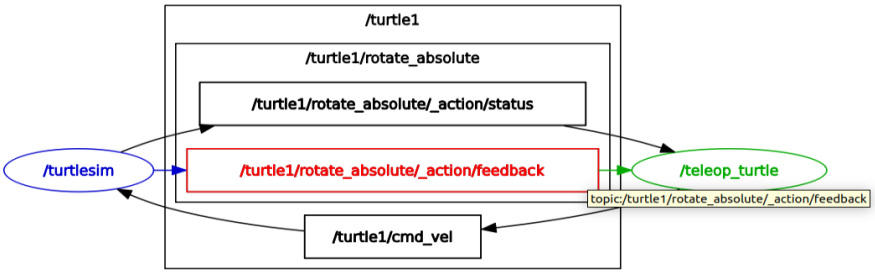

Figura 3.1: Esempio di collegamenti tra nodi e topic con il comando plugin > Introspection > Node Graph

di traiettoria dei giunti del robot. La categoria Visualization fornisce modelli aggiuntivi per la visualizzazione dei dati in tempo reale, come ad esempio *rqt-plot*, che consente di visualizzare i dati numerici in forma di grafico.

Inoltre, rqt fornisce anche un ambiente di sviluppo integrato (IDE) per la creazione di plugin personalizzati. Gli sviluppatori possono utilizzare il framework di Qt per creare i propri modelli personalizzati e facilmente integrabili nell'interfaccia grafica.

Si può eseguire qualunque strumento tramite il comando *rqt*, il quale consente anche di scegliere ed eseguire qualsiasi plug-in disponibile sul proprio sistema in finestre autonome.

RQt consiste in due metapacchetti:

- *rqt*: moduli per l'infrastruttura principale
- *rqt-common-plugins*: strumenti di debug comunemente usati

Rispetto ad una creazione della GUI da zero, RQt ha il vantaggio di avere procedure comuni standardizzate (e.g. ripristino stati precedenti), tra le più comuni:

plugins > introspection > node graph È uno dei comandi principali per visualizzare i nodi e gli argomenti che cambiano, nonchè le connessioni tra loro (e.g. Fig.3.1).

plugins > logging > console Tale sequenza viene utilizzata per visualizzare i messaggi di log del programma in esecuzione oppure caricare file precedentemente salvati per andare ad esaminare la cronologia dei messaggi. In Fig.3.2 si può notare come vengano visualizzati i dettagli dei messaggi di log [1] e tale interfaccia dà la possibilità di filtrare i messaggi tramite [2].

plugins > logging > bag Tramite tale comando è possibile visualizzare i dati registrati in un file bag. Utilizzando gli appositi pulsanti posti nel pannello grafico è possibile far riprodurre la bag, registrarla, salvarla oppure caricarla. Tale strumento è molto utile per il debugging e l'analisi dei dati registrati durante il funzionamento di un sistema robotico.

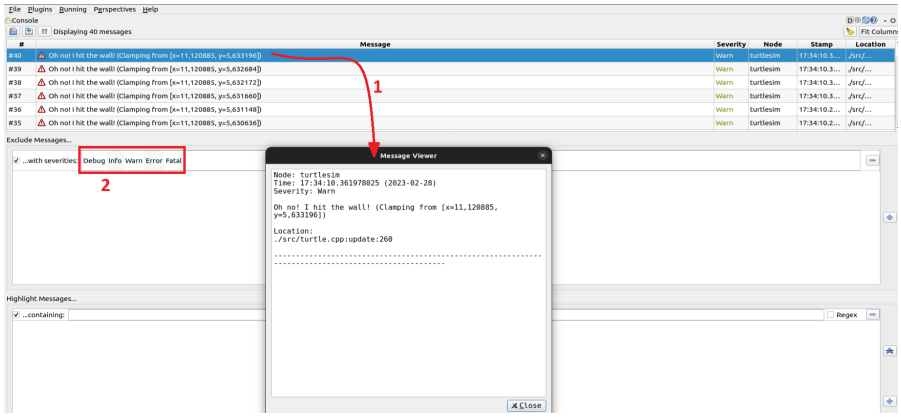

Figura 3.2: Esempio di messaggi di log(1)

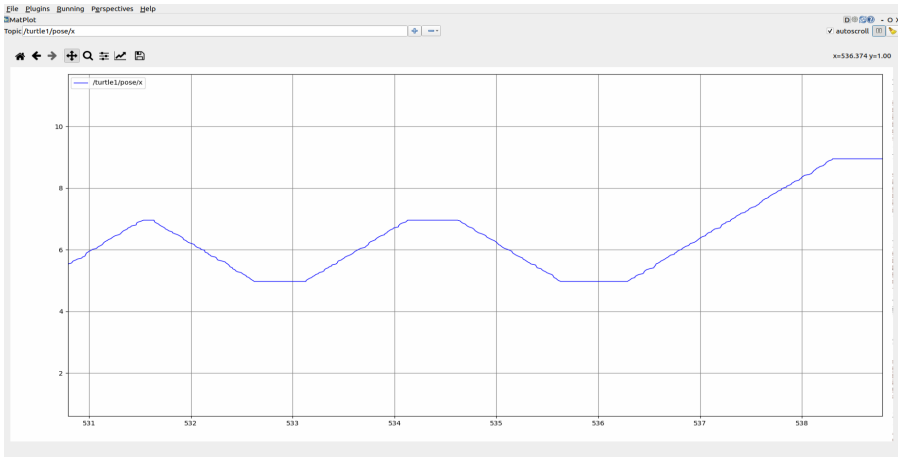

Figura 3.3: Esempio di grafico del parametro x di Turtlesim

plugins > robot tools > joint trajectory controllers Tale sequenza viene utilizzata per visualizzare e gestire i controllori della traiettoria dei giunti di un robot, si rivela molto utile per il controllo dei robot e per l'esecuzione di movimenti precisi e complessi. Tramite gli strumenti forniti si può avviare e interrompere i controllori, modificare le traiettorie dei giunti ed impostare i parametri di controllo.

plugins > visualization > plot Tramite tale comando è possibile tracciare un grafico per un parametro numerico di un nodo con la possibilità di cambiare scala sugli assi cartesiani e di spostarsi manualmente sul grafico (e.g. Fig.3.3). È possibile anche monitorare più parametri contemporaneamente.

plugins > visualization > image view Tale sequenza viene utilizzata per visualizzare le immagini acquisite da una fotocamera o da un altro sensore di un sistema visivo in real-time, fornendo anche alcuni strumenti per la modifica dell'immagine quali per esempio contrasto, luminosità, saturazione, gamma e bilanciamento del bianco.

plugins > visualization > tf tree Tramite tale comando è possibile visualizzare il grafo delle trasformazioni di coordinate del sistema robotico mostrando le relazioni tra i frame di riferimento del sistema

robotico, come ad esempio la posizione e l'orientamento di un sensore rispetto al frame di riferimento del robot.

plugins > topic > message publisher Tale sequenza viene utilizzata per inviare messaggi su un topic specifico semplicemente selezionandolo da interfaccia grafica, selezionando poi il tipo di messaggio che si desidera inviare ed infine inserendo i valori dei campo del messaggio. Viene utilizzato principalmente per testare e debuggare un sistema inviando manualmente i messaggi sui topic.

3.2 RVIZ2

RViz2 è uno strumento di visualizzazione 3D open source e gratuito sviluppato per consentire agli utenti di visualizzare il proprio robot, dati dei sensori, posizione e orientamento nello spazio, percorso di navigazione pianificato, mappe e altro ancora.

Grazie alla sua interfaccia grafica user-friendly, RViz2 consente agli sviluppatori di robotica di creare rapidamente applicazioni di visualizzazione personalizzate. Offre una vasta gamma di plugin e strumenti per aiutare gli sviluppatori a creare visualizzazioni avanzate. Alcuni di questi includono strumenti di selezione e modifica degli oggetti, visualizzazione di mappe e modelli di robot, e per la visualizzazione di percorsi di navigazione. Essendo anche altamente configurabile e personalizzabile, gli sviluppatori possono creare facilmente interfacce personalizzate per l'utente, aggiungere nuove funzionalità e adattare il software alle loro esigenze specifiche. RViz2 supporta anche la visualizzazione di dati da più robot contemporaneamente, il che è particolarmente utile per progetti multi-robot

Con questo strumento è possibile eseguire sessioni RVIZ preconfigurate per i tre tipi di camera ZED[2] quali ZED-M, ZED2 e ZED2i. Le tre sessioni caricano plug-in predefiniti standard per mostrare i dati più utilizzati dall'infrastruttura ZED ROS2[1]. In Fig3.4 è rappresentata una schermata tipica di RViz2, si può notare come nella parte sinistra sono collocati tutti i parametri per la gestione della simulazione.

Esistono diversi tools e plugin disponibili che possono essere utili per semplificare l'utilizzo e aumentare le funzionalità del software. I principali, che verranno descritti in seguito, sono:

- RViz2 Web Tools
- TF2 tools
- Robot Model
- Grid map
- Interactive Markers

2 telecamere stereo di profondità 3D sviluppate da Stereolabs, che consentono la creazione di modelli 3D ad alta precisione degli oggetti e degli ambienti circostanti

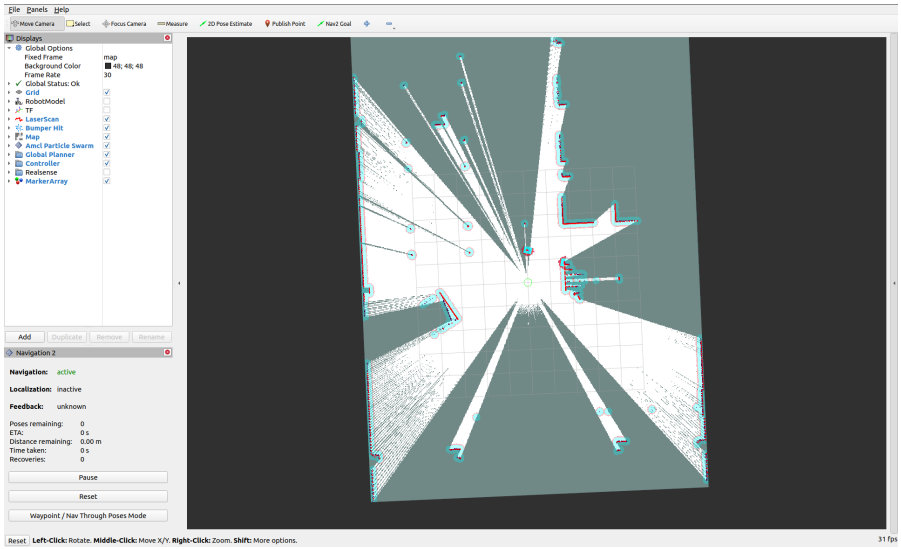

Figura 3.4: Esempio di visualizzazione della schermata principale di rviz con la configurazione View Robot[7]

rviz2 web tools E' un plugin che consente di accedere a RViz2 attraverso un browser web, senza dover installare il software sul proprio computer. Consente agli sviluppatori di accedere ai dati dei sensori del robot da qualsiasi dispositivo connesso alla stessa rete del robot, rendendo più facile e conveniente il debug e il monitoraggio del robot. Offre una vasta gamma di funzionalità, tra cui la visualizzazione dei dati dei sensori in tempo reale, la manipolazione del robot in tempo reale e l'accesso ai dati di stato del robot. Inoltre, consente di interagire con il robot attraverso un'interfaccia utente grafica (GUI), che semplifica il debug e il monitoraggio del robot.

TF2 (TRANSFORMATION LIBRARY) E' un framework di ROS2 per la gestione delle trasformazioni tra i diversi frame di riferimento utilizzati dai sensori e dai componenti del robot. Questo sistema consente di gestire le trasformazioni tra i frame di riferimento in modo efficiente e preciso, fornendo una base solida per la programmazione robotica. E' progettato per semplificare la gestione delle trasformazioni tra i diversi frame di riferimento utilizzati dai sensori e dai componenti del robot, consentendo di convertire le coordinate da un frame di riferimento all'altro in modo efficiente e accurato. In particolare, TF2 consente di eseguire le trasformazioni in modo asincrono, riducendo al minimo la latenza e massimizzando l'efficienza del sistema.

TF2 è composto da diversi componenti, tra cui un albero dei frame di riferimento, un sistema di buffer delle trasformazioni e una libreria di trasformazioni. L'albero dei frame di riferimento è una struttura gerarchica che descrive la relazione tra i diversi frame di riferimento utilizzati dal robot. Il sistema di buffer delle trasformazioni consente di memorizzare le trasformazioni tra i diversi frame di riferimento, mentre la libreria di trasformazioni consente di effettuare le trasformazioni in modo efficiente.

Offre un'interfaccia di programmazione delle applicazioni (API) intuitiva e facile da usare, che consente agli sviluppatori di integrare

facilmente le funzionalità di TF2 nei loro progetti di robotica.

robot model E' un componente che consente di visualizzare un modello del robot. Questo componente utilizza la libreria URDF[3] per caricare e visualizzare tale modello. L'URDF descrive la geometria del robot, la sua cinematica e la sua dinamica, ed è supportato da molti robot e simulatori di robot. Consente di visualizzare il modello del robot in modo interattivo, dando la possibilità di esplorare il modello del robot e di interagire con esso, consente inoltre di visualizzarlo in diversi formati, fornendo una maggiore flessibilità per la visualizzazione del modello del robot. Supporta anche la visualizzazione delle trasformazioni dei giunti del robot, consentendo di vedere in tempo reale come le diverse parti del robot si muovano in relazione l'una all'altra. Inoltre, consente di visualizzare le informazioni sui sensori montati sul robot, come le telecamere.

grid map E' un componente che consente di visualizzare mappe. Queste griglie sono utilizzate per rappresentare l'ambiente circostante del robot in termini di occupazione delle celle della griglia.

Supporta la visualizzazione di diverse informazioni sulla mappa, come ad esempio le informazioni sui sensori utilizzati per costruire la mappa, la risoluzione della griglia e la scala della mappa. Una delle caratteristiche più utili di Grid map è la possibilità di interagire con la mappa. Ad esempio, è possibile modificare manualmente la mappa per segnalare l'occupazione o la libera circolazione di alcune celle, o impostare le regioni di interesse sulla mappa.

interactive marker E' uno dei componenti di RViz2 che consente di creare oggetti interattivi tridimensionali all'interno dell'ambiente grafico. Questi oggetti possono essere utilizzati per controllare un robot o un'interfaccia utente, o per fornire un modo per interagire con un modello 3D. Permette di creare oggetti interattivi in tempo reale, che possono essere modificati e spostati da utenti umani o da algoritmi di controllo automatico. Questi oggetti possono avere diverse forme e funzionalità, come ad esempio frecce per indicare una direzione, pulsanti per attivare una funzione, e anche forme personalizzate, come oggetti CAD. Supporta anche la possibilità di visualizzare e modificare la posizione e l'orientamento dell'oggetto, così come altri attributi, come le dimensioni e le proprietà fisiche dell'oggetto.

Uno degli utilizzi più comuni di Interactive Markers è per il controllo di un robot. Ad esempio, gli utenti possono utilizzare un oggetto interattivo per muovere il braccio di un robot in una posizione specifica o per controllare la velocità di una macchina mobile.

Tramite un pannello posto nell'interfaccia grafica, è possibile personalizzare alcuni parametri della simulazione:

3 una specifica XML per modellare sistemi multicorpo

opzioni globali Per poter visualizzare le informazioni pubblicate dalla telecamera ZED, è necessario configurarle correttamente attraverso i seguenti parametri:

- *Fixed frame*: indica il nome del frame utilizzato come riferimento per tutti gli altri. Si può selezionare ogni frame disponibile nella casella corrispondente.
- *Frame rate*: indica la frequenza massima utilizzata per aggiornare la visione 3D, normalmente 30 o 60 FPS[4].

grid Come descritto in precedenza, permette di visualizzare una griglia normalmente associata al piano del pavimento. Anch'essa ha dei parametri chiave quali:

- *Reference frame*: indica il frame utilizzato come riferimento per le coordinate della griglia (normalmente *fixed-frame*).
- *Plane cell count*: stabilisce la dimensione della griglia in numero di celle.
- *Normal cell count*: identifica il numero di celle nella direzione normale al piano della griglia
- *Cell size*: stabilisce la dimensione di ogni cella della griglia (in metri).
- *Plane*: identifica i due assi orientati del piano della griglia.

robot model Come sopra descritto, consente di visualizzare il modello di robot in base alla sua descrizione dal modello URDF. I parametri fondamentali sono:

- *Visual enabled*: permette di abilitare o disabilitare la visualizzazione 3D del modello.
- *Collision Enabled*: consente di attivare il controllo per la collisione del robot con eventuali ostacoli
- *Links*: espandendo questa voce, è possibile visualizzare la struttura ad albero del modello e relativi collegamenti, posizione e orientamenti nello spazio rispetto al *fixed frame*.

TF2 Come descritto precedentemente, permette di visualizzare la posizione e l'orientamento di tutti i frame che ne compongono la gerarchia (e.g Fig.3.5). Anch'esso ha parametri chiave quali:

- *Show name*: permette di abilitare o disabilitare la visualizzazione 3D del nome dei collegamenti.
- *Show axes*: consente di abilitare o disabilitare la visione 3D degli assi dei telai.

4 Frames Per Second

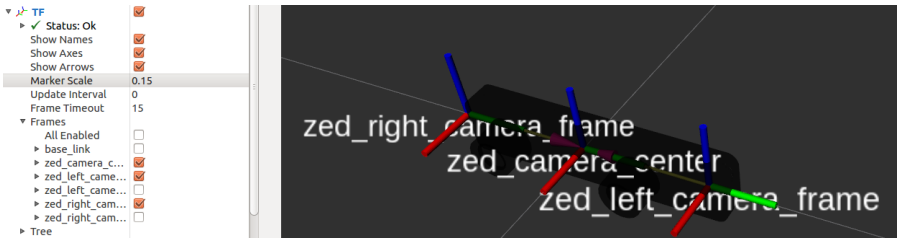

Figura 3.5: Esempio di visualizzazione con flag attivi dell'orientamento, nome e posizione del frame

- *Show arrows*: permette di abilitare o disabilitare la visualizzazione 3D delle frecce che collegano i vari frame.
- *Marker Scale*: viene utilizzato per ridimensionare tutti gli oggetti TF per renderli chiaramente visibili.
- *Update interval*: identifica il tempo di aggiornamento (in secondi).
- *Frames*: espandendo tale voce, è possibile abilitare o disabilitare la visualizzazione di ogni singolo frame consentendo di concentrarsi solo sulle parti fondamentali del proprio progetto.

CONCLUSIONI ROS2 fornisce diverse interfacce grafiche per controllare e comandare il proprio robot. RQt è un'interfaccia grafica utilizzata per visualizzare e interagire con i dati e i componenti del sistema robotico, risulta essere molto utile per sistemi sia di piccole che di grandi dimensioni. RViz2 è uno strumento grafico per la visualizzazione di modelli 3D di robot, dello spazio circostante e dei dati dei sensori, risulta essere particolarmente utile per progetti di grandi dimensioni che coinvolgono anche più robot.

CONCLUSIONI

ROS2 è uno dei principali software open-source atto a sviluppare progetti in ambito robotico. Utilizza strumenti utili per avviare, controllare e monitorare testualmente il robot, fino anche a rappresentare graficamente i parametri del robot ed effetturare una simulazione 3D dello stesso.

Risulta quindi molto utile per lo sviluppo e lo studio di piccoli e grandi progetti, sia industriali che non, per via della sua forte modularità e semplicità.

Ros2 mette a disposizione strumenti per l'analisi e la visualizzazione di dati in ambito robotico quali strumenti testuali (CLI tools) per comprendere le connessioni tra i vari nodi, strumenti grafici (RQt) per visualizzare la rappresentazione grafica dell'architettura e un'interfaccaia grafica (RViz2) per visualizzare un modello 3D del robot nello spazio in real-time.

BIBLIOGRAFIA

[1] *Data display with Rviz2*. URL: https://www.stereolabs.com/docs/ros2/rviz2/.

[2] *DDS foundation*. URL: https://www.dds-foundation.org/.

[3] Omar Elmofty. «ROS2 vs. ROS1— key differences and which one is better?» In: (). URL: https://medium.com/@oelmofty/ros2-how-is-it-better-than-ros1-881632e1979a.

[4] *How to Simulate a Robot Using Gazebo and ROS 2*. URL: https://automaticaddison.com/how-to-simulate-a-robot-using-gazebo-and-ros-2/.

[5] Morgan Quigley. «ROS: an open-source Robot Operating System». In: *ICRA workshop on open source software* 3.3.2 (2009). URL: http://robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf.

[6] *ROS2 Documentation*. URL: https://docs.ros.org/en/humble/index.html.

[7] *Rviz2*. URL: https://turtlebot.github.io/turtlebot4-user-manual/software/rviz.html.

[8] Ricardo Tellez. *A History of ROS (Robot Operating System)*. URL: https://www.theconstructsim.com/history-ros/.

[9] *The Player Project*. URL: https://playerstage.sourceforge.net/.

[10] «The ROS Command Line Interface». In: (). URL: https://osrf.github.io/ros2multirobotbook/ros2_cli.html.

[11] *What is a TurtleBot?* URL: https://www.turtlebot.com/.

[12] Keenan Wyrobek. *The Origin Story of ROS, the Linux of Robotics*. URL: https://spectrum.ieee.org/the-origin-story-of-ros-the-linux-of-robotics.