Politecnico di Torino

DIPARTIMENTO DI AUTOMATICA E INFORMATICA

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Monitoraggio remoto di robot mediante tecnologie Web Based e framework ROS

Relatore

Prof. Ing. Enrico Masala

Candidato

Federico Barone

Anno Accademico 2018-2019

Sommario

La robotica rappresenta uno dei settori disciplinari maggiormente in crescita nel mondo della ricerca tecnologica. In particolare, la robotica di servizio spicca tra i vari ambiti che studiano tale disciplina. Il presente elaborato si pone l'obiettivo di realizzare un'applicazione web che sia in grado di interagire con un robot da remoto. Questa permette sia di controllare i movimenti effettuati dal robot che le sue potenzialità. Di fatto, il robot presenta alcune caratteristiche dalle quali è possibile ricavare molteplici informazioni, tra cui, principalmente, la sua posizione. Inoltre, mediante l'utilizzo di un dispositivo android, è possibile utilizzare opportuni sensori presenti in esso gestendone i dati multimediali acquisiti. L'interfaccia web, nel suo complesso, ha lo scopo di monitorare da remoto l'ambiente circostante al robot, visualizzando i parametri in opportuni grafici bidimensionali. L'utilizzo del framework ROS (Robot Operating System) è alla base della programmazione robotica di servizio. Con l'emergere dell'Internet of Things (IoT), aumenta l'interesse nel fornire interfacce web che consentano agli utenti di monitorare i loro strumenti da remoto. Pertanto, sfruttando talune tecnologie Web Based è stato possibile realizzare uno strumento, composto dall'ambiente di configurazione ROS e dall'applicativo web, atto al monitoraggio remoto del robot. Usufruendo di uno o più dispositivi android posti sul robot, inoltre, è stato possibile monitorare e analizzare differenti parametri acquisiti dai vari sensori. Effettuando varie misurazioni mediante più dispositivi android, è stato possibile confrontare i dati acquisiti e trarre le dovute considerazioni.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

> Il **robot** è un manipolatore multifunzionale riprogrammabile, progettato per muovere materiali, parti, attrezzi o dispositivi specialistici attraverso vari movimenti programmati, per l'esecuzione di diversi compiti [1].

> La robotica è un settore disciplinare che ha per oggetto lo studio e la realizzazione di robot, e le loro applicazioni pratiche nelle attività di produzione industriale e di ricerca scientifica e tecnologica [2].

Questa disciplina analizza il comportamento e le gestualità del corpo umano ed, attraverso dispositivi artificiali (attuatori e sensori), tenta di trasmetterli alle macchine. La sperimentazione ed, ancor di più, il ragionamento sono linfa vitale per questa disciplina. È, inoltre, necessario sottolineare l'importanza della cooperazione della robotica con discipline ad essa affini: la meccanica, l'elettronica, l'informatica, la sensoristica, l'intelligenza artificiale e la matematica.

Si suddividano i campi della robotica in due macrocategorie: la **robotica industriale**, la quale si occupa delle applicazioni dei robot in ambito industriale; la **robotica avanzata**, che si propone come obiettivo quello di realizzare applicazioni che possano essere utili in ambienti ostici (spaziale,

1. Robot Institute of America, 1980. Divisione della School of Computer Science presso la Carnegie Mellon University di Pittsburgh, in Pennsylvania, negli Stati Uniti. Un articolo del giugno 2014 della rivista *Robotica Business Review* la definisce "la migliore struttura di ricerca sulla robotica del mondo" e "un pioniere nella ricerca e nell'educazione della robotica".
2. Cfr. http://www.treccani.it/enciclopedia/robotica

sottomarino, nucleare, militare, etc.). Un settore in crescita della robotica avanzata è la *robotica di servizio*, i cui sviluppi appaiono piuttosto proficui per il progresso tecnologico. Fine ultimo della robotica di servizio è assistere ed interagire con l'essere umano. Un robot, pertanto, deve essere in grado di muoversi abilmente nell'ambiente circostante, di modo che non rechi danni ad oggetti o persone, e sia, altresì, in grado di sfruttare le proprie capacità svolgendo mansioni predefinite: applicazioni domestiche, assistenza medica, intrattenimento, agricoltura, education, monitoring, etc.

Affinché un robot eserciti le sue funzioni correttamente, è necessario che recepisca i comandi ad esso impartiti dalla stazione di controllo. Tali comandi, inoltre, devono essere acquisiti coerentemente con il lavoro stabilito a priori. D'altra parte, per far sì che un robot possa interagire con la stazione remota, bisogna disporre di opportuni algoritmi in grado di rilevare i comandi che vengono inoltrati al dispositivo suddetto e rendere tali ordini utili per finalizzare l'obiettivo preposto. Recenti sviluppi (afferenti agli ultimi decenni) nel campo della robotica ed, in particolar modo, nella sensoristica, hanno consentito lo sviluppo di queste capacità.

Il presente elaborato si propone di realizzare un'interfaccia web che permetta all'utente di poter interagire da remoto con il robot. La matrice profonda di questo studio è rappresentata dal monitoraggio del robot a distanza da una stazione remota, nonché dal controllo dei suoi movimenti e dall'esercizio di ogni sua potenzialità. Si è reso, altresì, opportuno l'utilizzo di particolari sensori messi a disposizione dai dispositivi android e la gestione dei dati multimediali provenienti da essi.

1.1 Stato dell'arte

Il lavoro svolto per l'elaborazione di questa tesi si avvale dell'apporto di studi incentrati sulla gestione remota di droni e rovers da parte dell'utente. Questi, insieme ad ulteriori analisi basate sulla gestione dei dati multimediali, sono i temi principali dell'elaborato. Una panoramica viene presentata qui di seguito, dove sono stati riportati gli stati dell'arte dei principali argomenti caratterizzanti la tesi.

1.1.1 Monitoraggio remoto di droni e rovers

La progettazione e lo sviluppo di software efficienti per dispositivi mobili, quali, ad esempio, robot, richiedono tempo e risultano piuttosto impegnativi.

Avere una conoscenza interdisciplinare, dai driver hardware di basso livello alle astrazioni del software di alto livello, è sicuramente un requisito utile per chi si appresta ad operare nel mondo della robotica. Fortunatamente, l'emergenza di piattaforme middleware di robotica, come Robot Operating System (ROS), Player Project, Open RObot COntrol Software (OROCOS), Yet Another Robot Platform (YARP) ed altri, hanno fornito un passo in più verso un approccio più accessibile per lo sviluppo di software nell'ambito della robotica.

Avere un accesso remoto mediante interfacce e pagine web, consente un'interazione e un controllo di droni e rovers da remoto molto più semplice, specie per gli utenti inesperti. Inoltre, la teleoperazione da remoto permetterebbe di eseguire compiti in condizioni avverse per gli esseri umani, come il trasporto e salvataggio. Questa funzione risulterebbe piuttosto utile anche in particolari ambiti come quello militare e l'enforcement. Con la crescente tendenza all'integrazione delle tecnologie di intelligence, è possibile migliorare l'efficienza e ridurre la manodopera in situazioni quali la manutenzione, compiti di routine di sorveglianza con intervento umano minimo.

La realizzazione di un'interfaccia web che permetta di monitorare il robot, tuttavia, non è basata solamente sul controllo del moto del robot stesso. Infatti, grazie all'utilizzo di opportuni sensori, per esempio una videocamera incorporata oppure dispositivi di controllo di vario tipo (termometro, barometro, giroscopio, accelerometro, etc.), è possibile effettuare opportune misurazioni anche in luoghi difficilmente accessibili all'uomo.

Tuttavia, il monitoraggio da remoto non sfrutta solamente le qualità e le caratteristiche presenti nel robot, ma tiene conto anche di talune tecnologie realizzate proprio per essere applicate al mondo della robotica. Utilizzare lo smartphone come una componente sensoristica, ci consente di sfruttare le qualità presenti in esso, analizzando le tecniche e librerie messe a disposizione dai vari framework.

1.1.2 Gestione dei dati multimediali

La multimedialità è una forma di comunicazione basata sull'interazione di più linguaggi (testi scritti, immagini, video, audio) in uno stesso supporto o contesto informativo. Quando si parla di «contenuti multimediali», in ambito informatico, si intende l'utilizzo di diversi media atti ad inoltrare un'informazione relativa a qualcosa.

Il dato multimediale si presenta sotto forma di differenti formati: testo, audio, immagini e video. La caratteristica condivisa da tutte le tipologie succitate è che la generazione del dato multimediale (nella sua forma finale di natura digitale) si produce dal campionamento di un segnale analogico. Tale fase di rilevazione del dato «data capture» può essere eseguita in tanti modi e con diversi livelli di accuratezza, producendo dati di altrettanta qualità.

La realizzazione dell'interfaccia web per il monitoraggio del robot, come detto in precedenza, non è basata solamente sul controllo del moto del robot. Mediante opportuni sensori installati sul robot stesso ed all'interno dello smartphone, è stato possibile analizzare specifici dati multimediali. Inoltre, attraverso una videocamera opportunamente ubicata all'interno del robot, è stato possibile osservare le immagini trasmesse in tempo reale.

Importante studio è stato dedicato alla trasmissione in streaming dei dati multimediali video, focalizzando l'attenzione sia sull'utilizzo di dati compressi, che sull'uso di quelli grezzi. A questo tema sono state aggiunte ulteriori analisi relative ai sensori d'ambiente, quali dispositivi di controllo di temperatura e pressione, oltre ai valori ricavati dall'illuminamento. Ulteriori studi sono stati condotti sulla sensoristica di rete; in particolar modo, è stata effettuata un'analisi dei dati trasmessi e ricevuti in rete, della potenza del segnale e del livello di qualità del segnale. In definitiva, è stato realizzato un sistema di geolocalizzazione che permette di individuare esattamente la posizione del robot all'interno di una mappa in un determinato lasso di tempo.

1.2 Struttura dell'elaborato

Le linee seguite dal presente elaborato sono le seguenti:

- Nel primo capitolo è presente un celere accenno al lavoro svolto, guardando per sommi capi al mondo della robotica e, soprattutto, ai suoi possibili ambiti d'applicazione. Una veloce panoramica ha messo in evidenza le tematiche fondamentali della tesi: il monitoraggio remoto di droni e rovers e la gestione dei dati multimediali.
- Il secondo capitolo entra nel dettaglio delle tecnologie utilizzate, in particolare è descritto il framework ROS e le varie librerie messe a disposizione. Un'ulteriore analisi indaga i vari tool utilizzati per la simulazione del lavoro svolto.
- Nel terzo capitolo è presentata, con dovizia di particolari, l'architettura dell'applicazione web, di cui sono descritti i singoli componenti. Inoltre,

è mostrata la configurazione dell'ambiente di lavoro, sia manualmente (eseguendo i singoli comandi), sia in maniera automatica (mediante un opportuno script). Infine, è illustrata la realizzazione di un package ROS per l'analisi della rete.

- Il quarto capitolo descrive interamente l'applicazione web realizzata, introducendo la tecnologia Web Based e la struttura dell'interfaccia stessa. Sono state analizzate le varie sezioni dedicate ai sensori messi a disposizione, focalizzando l'attenzione sui topic utilizzati, sui messaggi presenti nel framework ROS e su quelli appositamente realizzati per la sensoristica di rete.

- Nel quinto capitolo vengono descritti, in maniera dettagliata, i passaggi da eseguire affinché sia possibile configurare correttamente il Clearpath Jackal. Inoltre, viene descritta l'usabilità dell'interfaccia web, mostrando come l'utente può interagire con essa. Infine, vengono illustrati i risultati ottenuti dalle varie misurazioni effettuate, utilizzando opportuni grafici bidimensionali e tabelle riassuntive dei valori acquisiti.

- Il sesto capitolo conclude l'elaborato riassumendo gli step svolti per il lavoro di tesi, focalizzando l'attenzione sui risultati analizzati e i possibili sviluppi futuri.

Capitolo 2

Robot Operating System

L'acronimo ROS, ovvero Robot Operating System, individua uno standard de facto per la «prototipazione» e lo sviluppo di software per applicazioni di robotica. Questo presenta diversi livelli di astrazione software alle risorse hardware robotizzate. Ciò consente agli sviluppatori di software di approcciarsi alla programmazione hardware di basso livello in maniera più semplice. Tuttavia, lo sviluppo delle applicazioni lato client richiede una piena comprensione e padronanza di ROS e della robotica, il che non è semplice per i non-ROS utenti e per principianti. Con l'emergere dell'Internet of Things (IoT), inoltre, c'è un crescente interesse nel fornire interfacce web che consentano agli utenti di accedere a Internet senza problemi.

Robot Operating System, pertanto, è un framework open source, contrariamente a quanto si potrebbe dedurre dal nome che gli è stato assegnato. Esso permette di sviluppare software per la robotica. La distribuzione utilizzata per questo progetto è la *Kinetic versione 1.12.14 LTS*.

ROS mette a disposizione degli sviluppatori diverse funzionalità: astrazioni hardware, controllo di dispositivi a basso livello, comunicazione tra nodi e gestione dei pacchetti. Fornisce anche tool e librerie per ottenere, costruire, scrivere ed eseguire codice tra più computer.

2.1 Concetti fondamentali di ROS

Il sistema ROS si basa sul concetto di grafico computazionale, che rappresenta la rete di processi ROS (potenzialmente distribuiti attraverso varie macchine). I concetti fondamentali, su cui si basa l'architettura ROS, sono i seguenti:

- **Master**: questo elemento ha il compito di orchestrare i nodi ad esso connessi gestendone l'esecuzione parallela e la comunicazione tra di loro. Senza la presenza del core, i nodi non potrebbero individuarsi reciprocamente, scambiarsi messaggi o invocare servizi.

- **Nodi**: il processo in ROS viene definito *nodo*, il quale è responsabile dell'esecuzione dei calcoli e dell'elaborazione dei dati raccolti dai sensori. Si tratta di un eseguibile, generato da un insieme di codici sorgente, che fa parte di un package. Un nodo ROS utilizza le librerie dedicate per comunicare sia con gli altri nodi che con il core. Può, altresì, leggere o pubblicare su un topic ed inoltre è in grado di fornire o richiedere servizi.

- **Messaggi**: i nodi comunicano tra di loro mediante i *messaggi*. Un messaggio è una struttura dati che assomma a sé diversi campi (con opportuni tipi). Sono supportati i tipi primitivi: integer, floating point, boolean, etc. I messaggi possono includere arbitrariamente strutture e array.

- **Topic**: i messaggi vengono scambiati mediante il sistema di comunicazione *publish/subscribe*. Un nodo invia un messaggio pubblicandolo su un dato topic. Il topic è un identificatore del contenuto del messaggio. Un nodo che, invece, è interessato ad un certo tipo di dato, si sottoscrive all'appropriato topic. Possono essere utilizzati più publishers e subscribers per un singolo topic, ed un unico nodo può pubblicare o sottoscrivere più topic.

- **Servizi**: il sistema di comunicazione *request/reply* viene realizzato mediante l'utilizzo di *servizi*, i quali vengono definiti da una coppia di strutture di messaggi: uno per la richiesta ed uno per la risposta. A differenza dei topic, che implementano uno scambio asincrono, i servizi vengono offerti da un nodo server, mentre un nodo client lo utilizza sfruttando il paradigma delle chiamate di procedura remote.

- **Bags**: sono dei formati per il salvataggio dei messaggi ROS. Sono un meccanismo importante per la memorizzazione dei dati, come quelli provenienti dai sensori, i quali possono risultare difficili da ricavare ma sono tuttavia necessari per lo sviluppo e il testing degli algoritmi.

2.1.1 Configurazione dell'ambiente ROS

ROS Kinetic è supportato unicamente nelle versioni di Ubuntu 15.10, 16.04 e Debian 8. Per poter usufruire di questo framework, è necessario eseguire alcune istruzioni utili per una corretta installazione.

Configurazione di sources.list Bisogna, anzitutto, configurare il PC di modo che esso sia in grado di scaricare del software a partire da packages.ros.org:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/
    ubuntu $(lsb_release -sc) main" > /etc/apt/sources.
    list.d/ros-latest.list'
```

Configurazione delle chiavi È opportuno settare le chiavi per poter effettuare il collegamento con il server:

```
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-
    keyservers.net:80 --recv-key 421
    C365BD9FF1F717815A3895523BAEEB01FA116
```

Installazione In prima istanza si dovrebbe aggiornare il proprio package presente nel sistema operativo:

```
$ sudo apt-get update
```

In seguito è possibile scegliere tra diverse modalità per effettuare l'installazione del framework. Quella consigliata è l'installazione completa, la quale comprende ROS, rqt, rviz, librerie generiche ROS e simulatori 2D/3D:

```
$ sudo apt-get install ros-kinetic-desktop-full
```

Configurazione dell'ambiente È conveniente, inoltre, eseguire il seguente comando in modo tale che vengano aggiunte automaticamente le variabili d'ambiente ROS alla sessione di bash ogni volta che viene lanciata una nuova shell:

```
$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.
    bashrc
$ source ~/.bashrc
```

Infine, una volta installato correttamente, è possibile eseguire il core di ROS mediante il seguente comando:

```
$ roscore
```

Nella Figura 2.1 è possibile visualizzare le informazioni relative al master avviato mediante il suddetto comando:

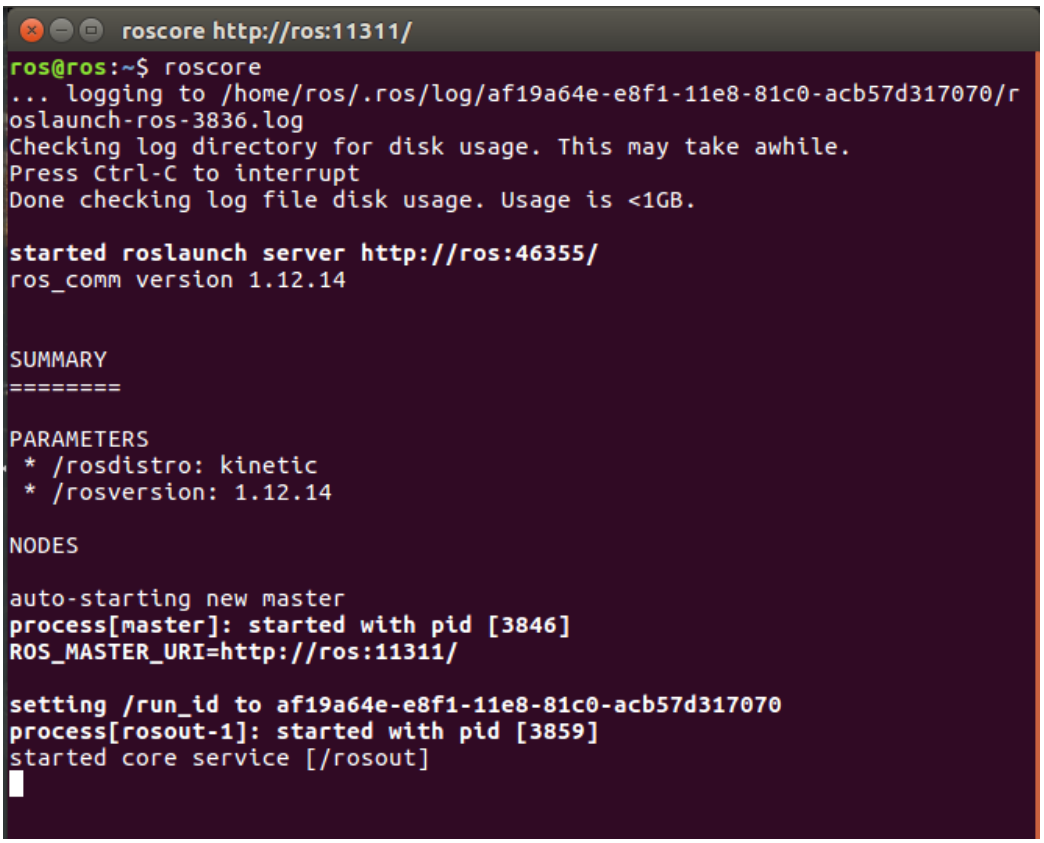


Figura 2.1: Avvio del core ROS mediante il comando *roscore*

È stato avviato il server contenente il core di ROS. La distribuzione indicata è la *kinetic*, mentre la versione è *1.12.14*. Inoltre, il master è identificato da uno specifico *pid*. Si osservi, a dimostrazione, la Figura 2.1.

2.2 Gazebo Multi-Robot Simulator

La simulazione di un robot è uno strumento essenziale per chi lavora nel mondo della robotica. Un simulatore ben progettato consente di testare rapidamente algoritmi, progettare robot, eseguire test di regressione e addestrare i sistemi di intelligenza artificiale utilizzando scenari realistici.

Gazebo offre la possibilità di simulare con scrupolosa efficienza complessi ambienti interni ed esterni. I preziosi strumenti di cui questo simulatore dispone sono: un robusto motore fisico, una grafica di alta qualità e comode interfacce programmatiche e grafiche. Sviluppato da Open-Source Robotics Foundation, si tratta di un tool open source.

2.2.1 Architettura di Gazebo

Gazebo utilizza un'architettura distribuita mediante opportune librerie separate per la simulazione fisica, il rendering, l'interfaccia utente, la comunicazione e la generazione di sensori. Gazebo fornisce, inoltre, un'architettura client/server per effettuare simulazioni:

- un server **gzserver** per la simulazione fisica, il rendering e i sensori;
- un client **gzclient** che fornisce un'interfaccia grafica per la visualizzazione e l'interazione con il simulatore.

Il client e il server comunicano usando le librerie di comunicazione presenti in Gazebo.

Gazebo Master Si tratta, di fatto, di un server che fornisce un lookup dei nomi e la gestione dei topic. Un singolo master può gestire simultaneamente: simulazioni fisiche, generatori di sensori e interfacce grafiche.

Librerie di Comunicazione Le principali librerie di comunicazione utilizzate tra i diversi processi sono:

- **Protobuf**: per la serializzazione dei messaggi;
- **boost::ASIO**: come meccanismo di trasporto.

È supportato il paradigma di comunicazione *publish/subscribe*. Questo meccanismo consente l'introspezione di una simulazione e fornisce un comodo meccanismo per controllare gli aspetti di Gazebo.

Librerie Fisiche Le librerie fisiche forniscono una generica interfaccia per la simulazione tra i componenti, includendo: corpi rigidi, forme di collisione e giunti per rappresentare i vincoli di articolazione. Questa interfaccia è stata integrata mediante l'utilizzo di quattro motori fisici open source:

- **Open Dynamics Engine (ODE)**: libreria per la simulazione dinamica dei corpi rigidi, risulta essere completa ed indipendente dalla piattaforma, mediante l'utilizzo di API C/C++ facili da usare. È utile sia per la simulazione di veicoli, che di oggetti in ambienti di realtà virtuale ed, in ultimo, di creature virtuali. Attualmente utilizzato per strumenti di authoring 3D e strumenti di simulazione.
- **Bullet**: 3D Game Multiphysics Library che fornisce il rilevamento delle collisioni allo stato dell'arte.
- **Simbody**: set di strumenti SimTK che detiene capacità di dinamica multibody generale. Viene fornito come API C++ open source e orientata agli oggetti. Questo set porta a risultati di qualità scientifica/ingegneristica ad alte prestazioni ed in condizioni di precisione controllata.
- **Dynamic Animation and Robotics Toolkit (DART)**: libreria cross-platform che fornisce strutture dati e algoritmi per applicazioni cinematiche e dinamiche in robotica e computer animation. Al contrario di molti noti motori fisici, che percepiscono il simulatore come una scatola nera, DART dà pieno accesso alle grandezze cinematiche e dinamiche interne. Offre, inoltre, la possibilità dell'utilizzo di API per incorporare le classi fornite dall'utente nelle strutture dati DART.

Librerie di Rendering La libreria di rendering maggiormente utilizzata è **OGRE (Object-Oriented Graphics Rendering Engine)**. Essa fornisce una semplice interfaccia per il rendering delle scene 3D, sia per la GUI che per le librerie di sensori. Include: illuminazione, trame e simulazioni varie. È possibile realizzare plugins opportuni per il motore di rendering.

Generazione di Sensori La libreria, che si occupa della generazione dei sensori, implementa tutti i vari tipi di attuatori, si aggiorna in maniera automatica e produce l'output specificato dai sensori inizializzati.

GUI La libreria GUI utilizza Qt [3] per la creazione di widget grafici utili per l'interazione con il simulatore. All'utente è data la possibilità di controllare punto per punto i GUI widget; modificare l'ambiente di simulazione aggiungendo, modificando o rimuovendo i modelli.

2.2.2 TurtleBot

TurtleBot è un kit di robot personali a basso costo con software open source. I creatori del suddetto kit sono Melonee Wise e Tully Foote i quali nel 2010, presso Willow Garage [4], hanno realizzato un robot in grado di orientarsi, vedere in 3D e possedere parecchi sensori per sviluppare applicazioni davvero interessanti.

TurtleBot consta di una base mobile, un sensore di distanza 2D/3D, un computer portatile o SBC (Single Board Computer) e il kit di montaggio di TurtleBot. Il progetto ha disposto un facile assemblaggio con l'utilizzo di prodotti e componenti di largo consumo, i quali possono essere creati con materiali standard. In quanto piattaforma mobile, TurtleBot offre alcune funzionalità delle piattaforme di robotica più grandi, come PR2 [5].

Di seguito illustriamo come è possibile avviare un ambiente virtuale in cui è possibile usufruire di un rover TurtleBot che risponda in maniera del tutto coerente ai comandi a cui è sottoposto.

TurtleBot Simulator in ROS with Gazebo L'ambiente virtuale Gazebo mette a disposizione molteplici librerie che permettono di trasferire nel mondo virtuale vari ambienti di lavoro. Inoltre, è possibile modificare questi ambienti e renderli adatti alle proprie esigenze mediante l'utilizzo di opportuni packages aggiuntivi.

Nel nostro caso, è stato installato un package aggiuntivo oltre a quelli messi già a disposizione dal tool Gazebo denominato *ros-kinetic-turtlebot*.

3. Framework per applicazioni multipiattaforma per la creazione di interfacce utente grafiche (GUI) eseguite su tutte le principali piattaforme desktop e sulla maggior parte delle piattaforme mobili o incorporate.
4. Laboratorio di ricerca fondato da Scott Hassan nel 2006 per lo sviluppo della robotica e l'avanzamento del software di robotica open source.
   Cfr. http://www.willowgarage.com/pages/about-us/history
5. PR2 è stato realizzato per sviluppare e testare applicazioni robotiche e tecnologiche. Combina la mobilità alla navigazione negli ambienti umani e la destrezza per afferrare e manipolare gli oggetti in quegli ambienti.

Questo package mette a disposizione vari componenti utili per la simulazione del nostro lavoro.

Gli step per l'installazione del suddetto package sono i seguenti:

```
$ sudo apt-get update
$ sudo apt-get install ros-kinetic-turtlebot ros-
    kinetic-turtlebot-apps ros-kinetic-turtlebot-
    interactions ros-kinetic-simulator ros-kinetic-
    turtlebot-simulator ros-kinetic-kobuki-ftdi ros-
    kinetic-rocon-remocon ros-kinetic-rocon-qt-library
    ros-kinetic-ar-track-alvar-msgs
$ source /opt/ros/kinetic/setup.bash
```

Una volta installato il package, è possibile aprire il terminale e lanciare il seguente comando per avviare il simulatore TurtleBot:

```
$ roslaunch turtlebot_gazebo turtlebot_world.launch
```

Nella Figura 2.2 viene illustrato l'ambiente di virtualizzazione Gazebo utilizzando il TurtleBot:

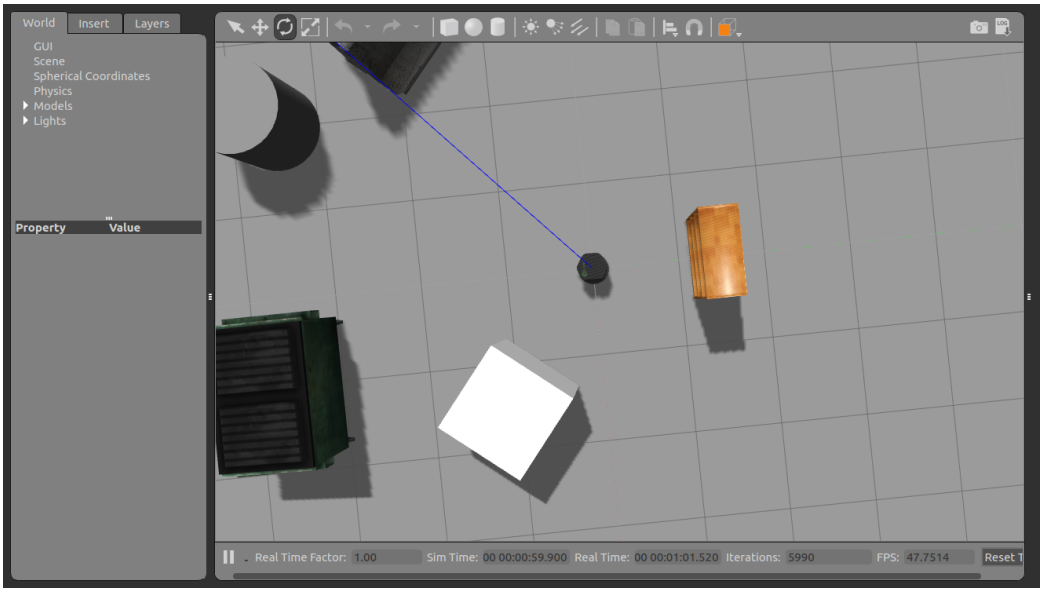


Figura 2.2: TurtleBot Gazebo Simulator

2.2.3 Clearpath Jackal

Clearpath Jackal, così come già presentato in TurtleBot, è una semplice piattaforma di ricerca robotica. Presenta sia un computer di bordo, che alcune funzionalità integrate con il framework ROS: GPS e IMU. Si tratta di un robot plug-and-play compatibile con tanti accessori per robot, in modo tale da poter ampliare la ricerca e lo sviluppo sui dispositivi mobili.

Il simulatore Gazebo presenta una versione del tutto analoga a quella già presentata per il robot TurtleBot, ma utilizzando, in questo caso, il robot Clearpath Jackal. Di seguito sono illustrati i passi per una corretta installazione.

Clearpath Jackal Simulator in ROS with Gazebo Per poter utilizzare il simulatore del robot Clearpath Jackal mediante l'applicazione Gazebo, è opportuno includere il package relativo al simulatore suddetto. In questo caso, il tool Gazebo mette a disposizione il simulatore Jackal denominato *ros-kinetic-jackal*. I comandi utili per una corretta installazione del simulatore sono i seguenti:

```
$ sudo apt-get update
$ sudo apt-get install ros-kinetic-jackal-simulator ros
  -kinetic-jackal-desktop
$ source /opt/ros/kinetic/setup.bash
```

Per avviare il simulatore Clearpath Jackal, bisogna digitare il suddetto comando:

```
$ roslaunch jackal_gazebo jackal_wolrd.launch
```

Nella Figura 2.3 viene illustrato l'ambiente di virtualizzazione Gazebo utilizzando il Clearpath Jackal:

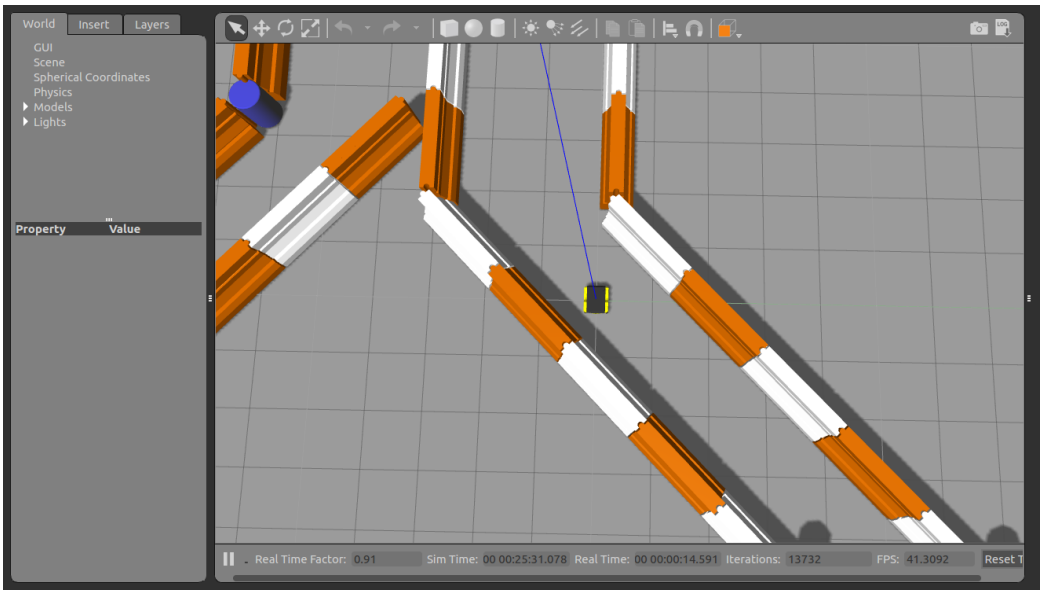


Figura 2.3: Clearpath Jackal Gazebo Simulator

2.3 RQT

RQT è un framework basato sulla tecnologia Qt e consta di tre meta-pacchetti:

- **rqt**: fornisce un widget rqt_gui che consente di avere più widget rqt ancorati in un'unica finestra;
- **rqt_common_plugins**: suite di strumenti back-end usabile in maniera runtime dal robot;
- **rqt_robot_plugins**: strumenti per interagire con i robot a runtime.

Gli strumenti rqt (ROS Qt GUI toolkit) che fanno parte di ROS permettono di rappresentare graficamente: i nodi ROS, i relativi topic, messaggi ed ulteriori informazioni.

2.3.1 rqt_graph

Uno degli strumenti che viene maggiormente impiegato nell'ambito di rqt è **rqt_graph**. Esso consente di visualizzare i nodi e gli argomenti attivi.

Per eseguire il tool sopracitato, è opportuno digitare il seguente comando:

```
$ rqt_graph
```

Un esempio che mostra l'utilizzo del suddetto tool è rappresentato nella Figura 2.4:

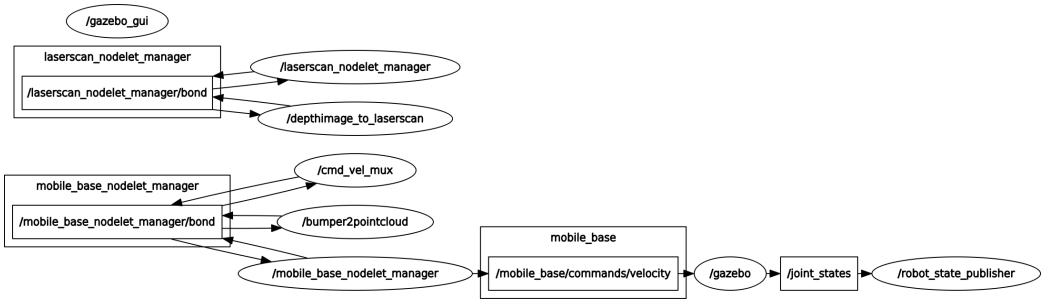


Figura 2.4: TurtleBot Gazebo rqt_graph

Nell'esempio in Figura 2.4 è stato eseguito il comando rqt_graph, dopo aver avviato il simulatore TurtleBot Gazebo. Come si può notare, il grafico mostra quali sono i nodi messi a disposizione dal simulatore stesso e i topic con i quali è possibile interagire.

Un esempio di topic utile per impartire comandi a distanza al robot TurtleBot è */cmd_vel_mux/input/teleop*. Tramite il suddetto topic, si ha la possibilità di settare i parametri presenti nel messaggio *geometry_msgs/Twist*, come mostrato nella Figura 2.5 e, di conseguenza, imprimere il movimento al dispositivo mobile.

```
ros@ros: ~
ros@ros:~$ rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

Figura 2.5: ROS Message *geometry_msgs/Twist*

I parametri presenti nel messaggio *geometry_msgs/Twist* in Figura 2.5 individuano i due possibili movimenti che può effettuare il robot: lineare e

angolare. È possibile, altresì, indicare lungo quale asse il robot debba essere spostato, modificando opportunamente i valori presenti nel messaggio.

2.3.2 rqt_image_view

Un altro strumento messo a disposizione da rqt è il package **rqt_image_view**. Si tratta di un plugin che permette la visualizzazione di immagini.

Per eseguire il tool sopracitato, è opportuno digitare il seguente comando:

```
$ rqt_image_view
```

Le immagini visualizzate dipendono dallo specifico topic che viene selezionato. Un esempio di visualizzazione di un'immagine è riportato nella Figura 2.6:

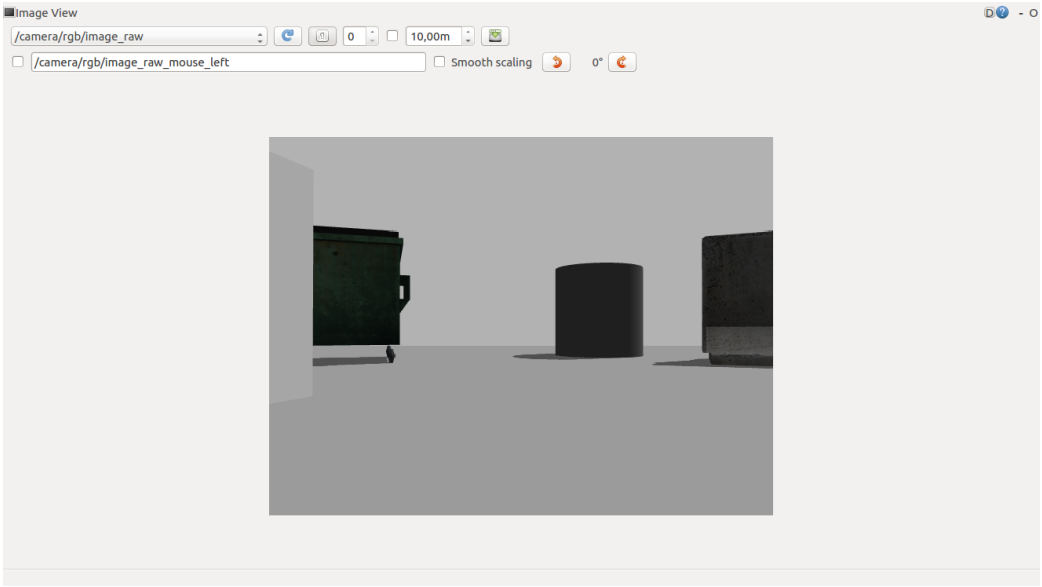


Figura 2.6: TurtleBot Gazebo rqt_image_view

Allorquando è stato selezionato il topic desiderato, è possibile effettuare il refresh di quest'ultimo e visualizzare il nuovo topic. Inoltre, l'immagine può essere ruotata e salvata all'occorrenza.

2.4 ROSBridge

Il framework ROS mette a disposizione uno strato di astrazione intermedio, definito *middleware*, il quale è in grado di fornire una tecnologia utile per lo sviluppo di applicazioni web. Tale strato software è chiamato **ROSBridge**.

ROSBridge è in grado di fornire un semplice accesso programmatico basato sulle Web Socket. Queste sono in grado di interfacciarsi al robot e ai vari algoritmi messi a disposizione da ROS. In particolare, facilita l'utilizzo delle tecnologie web, quali ad esempio JavaScript, allo scopo di ampliare l'utilizzo e l'utilità della tecnologia robotica.

2.4.1 Architettura di ROSBridge

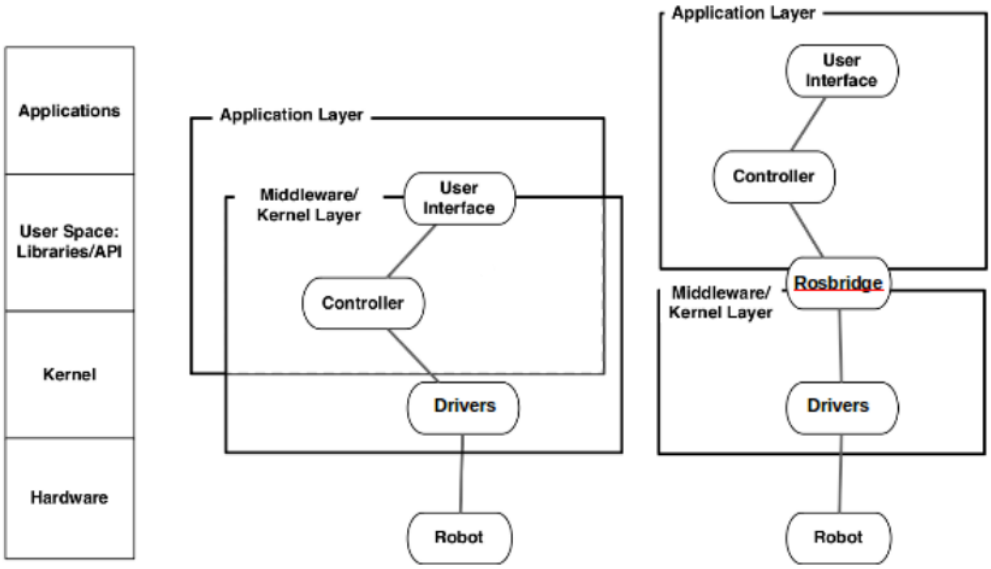


Figura 2.7: Architettura ROSBridge

ROSBridge apporta un ulteriore livello di astrazione sopra ROS (come mostrato nella Figura 2.7). Gli strati di cui uno sviluppatore deve tener conto sono quelli descritti a sinistra. ROSBridge tenta di stabilire un confine di astrazione più chiaro per occuparsi di questi strati contemporaneamente.

ROSBridge considera tutti gli elementi ROS dal punto di vista "backend": a tal proposito, lo sviluppatore medio non ha bisogno di una conoscen-

za approfondita delle interfacce di controllo a basso livello, né di costruire sofisticati sistemi middleware o algoritmi di controllo per il robot. Tuttavia, bisogna comprendere i meccanismi di costruzione e trasporto di informazione attraverso il middleware.

ROSBridge mette a disposizione un semplice protocollo di serializzazione socket che permette agli sviluppatori di creare applicazioni web. ROS, come abbiamo già descritto in precedenza, è in grado di controllare il robot attraverso appositi messaggi, oltre a fornire servizi per l'avvio e l'arresto dei singoli topic. ROSBridge è in grado di racchiudere questi due aspetti di ROS, presentando all'utente una vista unificata di un robot e del suo ambiente.

Il protocollo ROSBridge consente l'accesso ai messaggi e ai servizi ROS mediante oggetti JSON e fornisce, inoltre, il controllo sull'esecuzione dei nodi ROS e dei parametri a loro attribuiti.

ROSBridge gestisce lo scambio dei messaggi, sfruttando sia le Web Socket HTML5 sia le socket TCP/IP. È proposto, di seguito, un esempio: è possibile realizzare dei semplici client in JavaScript che sono in grado di pubblicare o sottoscrivere specifici topic, per inviare/ottenere informazioni specifiche al caso.

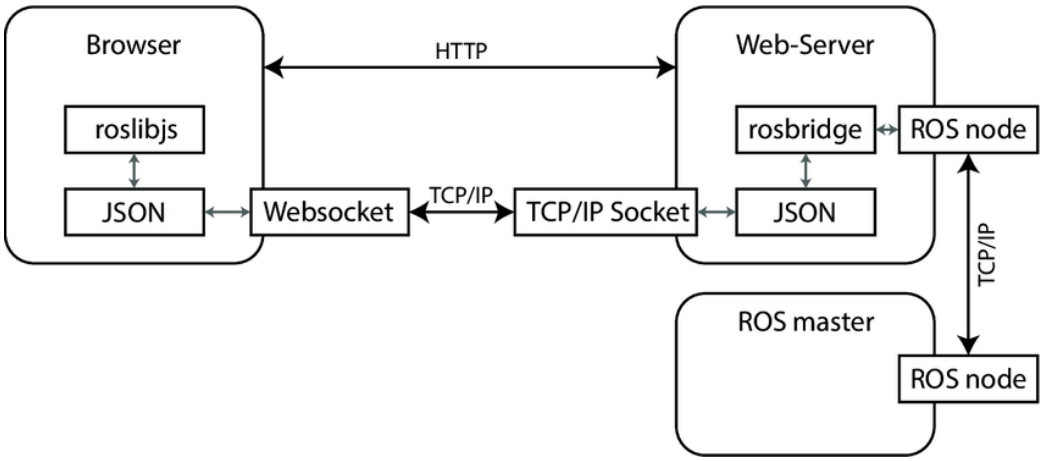


Figura 2.8: Comunicazione tra Browser Web e Server Web mediante ROSBridge

Nella Figura 2.8 è rappresentata una possibile comunicazione tra il Browser Web e il Server Web sfruttando le Web Socket messe a disposizione da

ROSBridge.

Questo paradigma può, tuttavia, essere utilizzato per qualsiasi linguaggio che supporti l'utilizzo delle socket TCP/IP. Per questo motivo, ROSBridge consente lo sviluppo di applicazioni robot in qualsiasi linguaggio di programmazione scelto dall'utente.

2.4.2 Configurazione di rosbridge_suite

L'architettura distribuita ROSBridge viene fornita mediante il package **rosbridge_suite**. Ciò che è in grado di fornire questo package riguarda, soprattutto, l'interazione tra ROS e la pagina web realizzata in JavaScript. Quest'ultimo è l'unico strumento, utilizzato in questo sistema, che garantisce l'interazione tra un'applicazione web e il mondo ROS.

Implementazione ROSBridge Il package rosbridge_suite è una raccolta di packages che implementa il protocollo ROSBridge e fornisce un livello di trasporto Web Socket. Tali packages includono:

- **rosbridge_library**: è un pacchetto core di ROSBridge. Ha il compito di prendere le stringhe JSON e inviare/ricevere i comandi a/da ROS;
- **rosapi**: rende accessibili determinate azioni ROS tramite chiamate di servizio riservate, di norma, alle librerie client ROS. Questo significa impostare e/o ottenere i parametri, acquisire l'elenco degli argomenti, etc.;
- **rosbridge_server**: fornisce una connessione Web Socket in modo tale che i Browser possano "parlare ROSBridge". *Roslibjs* è una libreria JavaScript per il Browser che può comunicare con ROS tramite *rosbridge_server*.

Installazione del package rosbridge_suite Il package presentato in precedenza può essere integrato al nostro ambiente di lavoro mediante il seguente comando:

```
$ sudo apt-get install ros-kinetic-rosbridge-suite
$ source /opt/ros/kinetic/setup.bash
```

Una volta installato il package, è possibile aprire il terminale ed eseguire il seguente comando per avviare server rosbridge:

```
$ roslaunch rosbridge_server rosbridge_websocket.launch
```

Di seguito viene illustrato il messaggio visualizzato dal terminale:

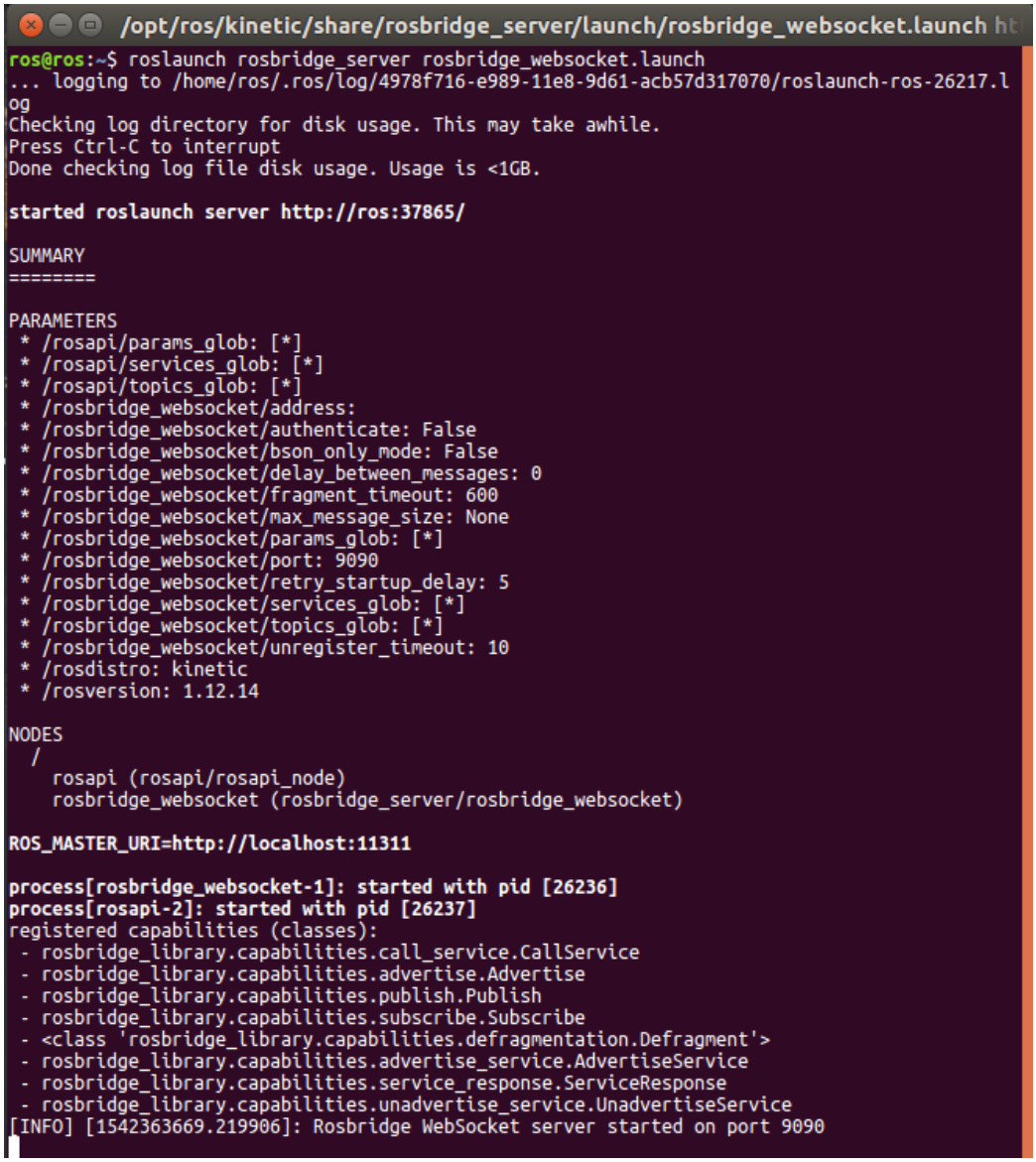


Figura 2.9: Avvio di ROSBridge mediante il comando *rosbridge_server*

Come si può notare dalla Figura 2.9, il server *ROSBridge WebSocket* è stato avviato con i parametri standard settati all'interno del package. Il server ROSBridge, in modalità standard, è in ascolto sulla porta 9090. Tuttavia,

è possibile modificare i parametri messi a disposizione dal package rosbridge_suite. Per esempio, è possibile modificare il valore della porta (valore: 8080) su cui il server si debba mettere in ascolto. Di seguito il comando per effettuare tale modifica:

```
$ roslaunch rosbridge_server rosbridge_websocket.launch
    port:= 8080
```

2.5 ROS web_video_server

Il framework ROS mette a disposizione, tra i vari packages proposti, un nodo in grado di garantire lo streaming HTTP di immagini provenienti da alcuni topic ROS. Tale package prende il nome di **web_video_server**.

Il suddetto nodo è stato realizzato per combinare le proprietà di due packages oramai in disuso in un unico nodo: *ros_web_video* e *mjpeg_server*.

2.5.1 Configurazione web_video_server

Il package web_video_server, come già antecedentemente sottolineato, mette a disposizione uno stream video di immagini ricavate da specifici topic ROS, a cui è possibile accedere tramite protocollo HTTP.

Installazione del package web_video_server Così come gli altri packages descritti nei precedenti paragrafi, anche il nodo web_video_server non è presente all'interno del framework ROS. Per renderlo disponibile, il package suddetto può essere installato sul nostro sistema operativo nel seguente modo:

```
$ sudo apt-get install ros-kinetic-web-video-server
$ source /opt/ros/kinetic/setup.bash
```

Una volta installato il package, è possibile eseguirlo mediante il comando *rosrun*:

```
$ rosrun web_video_server web_video_server
```

Questo comando è in grado di avviare un pacchetto eseguibile, a differenza del precedente *roslaunch*, il quale permette di eseguire nodi ROS con estensione *.launch*.

Di seguito viene illustrato il messaggio visualizzato dal terminale:

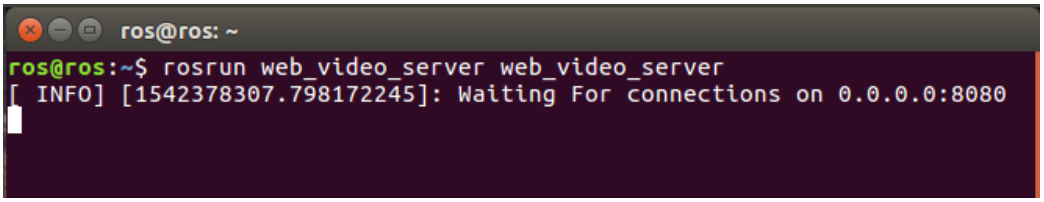


Figura 2.10: Avvio del server video mediante il comando *web_video_server*

Nella Figura 2.10 è possibile osservare il server video in ascolto di connessioni HTTP sull'indirizzo 0.0.0.0 alla porta 8080. Nel momento in cui viene rilevata una connessione, il terminale aggiorna le informazioni riportando le URL, le quali permettono di connettere il nodo al Browser Web.

Parametri presenti nel package web_video_server I parametri messi a disposizione dal suddetto package sono i seguenti:

- ~ **port** (integer, default: 8080): numero di porta su cui il server ascolta le richieste HTTP;
- ~ **address** (string, default: 0.0.0.0): indirizzo IP su cui il server ascolta le richieste HTTP;
- ~ **verbose** (boolean, default: true): visualizza le informazioni relative alle URL connesse al nodo;
- ~ **server_threads** (integer, default: 1): numero di thread utilizzati per servire le richieste HTTP;
- ~ **ros_threads** (integer, default: 2): numero di thread utilizzati per effettuare la codifica dell'immagine. Sono condivisi tra tutti gli stream. In questo modo il numero di questi thread non corrisponde al numero reale delle connessioni.

Di seguito viene mostrato come eseguire il package web_video_server che ascolta le richieste HTTP sulla porta 7070:

```
$ rosrun web_video_server web_video_server _port:=7070
```

URL utilizzabili per contattare il server Il package web_video_server mette a disposizione diverse URL per poter raggiungere il nodo. Ecco di seguito alcune di queste:

- `http://localhost:8080/stream_viewer?topic=ROS_TOPIC`: URL che permette di mostrare una pagina web con una schermata per lo stream video;
- `http://localhost:8080/stream?topic=ROS_TOPIC`: URL utilizzata per lo stream di un video in una pagina web creata dall'utente;
- `http://localhost:8080/snapshot?topic=ROS_TOPIC`: URL utilizzata per la visualizzazione dello snapshot dell'immagine successiva.

2.5.2 Settaggio dei parametri presenti nelle URL

Entrando nel dettaglio delle URL utilizzate per la connessione al nodo web_video_server, esistono diversi parametri sulla base del tipo di codec video utilizzato. La scelta del codec può essere effettuata mediante il parametro **type**. Una volta scelto il codec da utilizzare, è possibile settare gli ulteriori parametri per la codifica del video.

Esistono quattro parametri standard per tutte le tipologie di codec video:

- **width** (integer, default: original width): lo stream delle immagini viene ridimensionato ad una nuova larghezza ed altezza. Questo parametro deve essere utilizzato insieme al parametro *height*;
- **height** (integer, default: original height): lo stream delle immagini viene ridimensionato ad una nuova larghezza ed altezza. Questo parametro deve essere utilizzato insieme al parametro *width*;
- **invert** (none, default:): ruota l'immagine di 180 gradi prima di effettuare lo streaming;
- **default_transport** (string, default: raw): tipologia di traffico di dati trasportato sul canale HTTP.

MJPEG Motion JPEG è un codec video dove ogni singolo frame del video viene compresso in un'immagine JPEG. Esso non offre nessuna compressione tra frame, così che la qualità della compressione sia indipendente dal movimento presente nella singola immagine.

I parametri da utilizzare con il suddetto codec video sono i seguenti:

- **quality** (integer, default: 95): qualità dell'immagine jpeg (1...99). Questo parametro può essere utilizzato per ridurre la dimensione dello stream finale.

PNG Portable Network Graphics è un formato di file che permette la memorizzazione di immagini. Esso è in grado di immagazzinare immagini in modo *lossless*, ossia senza alcuna perdita di informazione.

Anche questo formato prevede il solo utilizzo del parametro **quality**; tuttavia, il valore di default, in questo caso, è 3.

ROS_COMPRESSED Non si tratta di un vero e proprio codec, tuttavia è in grado di settare in maniera automatica il formato dello stream video scegliendo quale sia il migliore tra *JPEG* e *PNG*. Inoltre, non ammette ulteriori parametri, a differenza dei precedenti formati codec video.

VP8 VP8 è un codec video, sviluppato da On2 Technologies, per sostituire il predecessore VP7. Esso viene utilizzato per la riproduzione di video in formato WebM per pagine realizzate in HTML5.

I parametri applicabili al seguente codec video sono i seguenti:

- **quality** (string, default: realtime): qualità del codec video;
- **bitrate** (string, default: 100000): valore massimo del bitrate. Un piccolo bitrate potrebbe aumentare significativamente la latenza a causa di una trasmissione ritardata di grandi frame;
- **qmin** (integer, default: 10): valore minimo di quantizzazione di un frame video;
- **qmax** (integer, default: 42): valore massimo di quantizzazione di un frame video;
- **gop** (integer, default: 250): determina la distanza massima tra i frame (intervallo keyframes).

VP9 VP9 è uno standard di compressione video sviluppato da Google. Si tratta di un miglioramento del codec video VP8, del quale mantiene la qualità ma utilizza un metodo di compressione più efficiente così da ridurre la dimensione del video.

Ugualmente al codec video VP8, lo standard VP9 utilizza i medesimi parametri per migliorare la qualità del video streaming.

H264 H264 o MPEG-4 è un formato standard di compressione video digitale con perdita di dati, creato dal Moving Picture Experts Group. Viene utilizzato dagli utenti finali per lo streaming video su Internet. È stato realizzato per fornire una buona qualità video a bit rate inferiori rispetto agli standard precedenti, senza aumentare la complessità e il costo di implementazione.

A differenza dei precedenti codec video, H264 prevede l'utilizzo dei parametri già visti nel caso dei codec VP8 e VP9; tuttavia, il parametro **quality** è stato sostituito dal parametro **preset**. Quest'ultimo è settato di default con il valore *ultrafast*. È possibile scegliere tra diversi possibili valori: superfast, veryfast, faster, fast, medium, slow, slower, very slow, placebo. Non si osservano, tuttavia, miglioramenti tra il tipo ultrafast o medium.

2.5.3 Estensione image_transport

Un'utile estensione per fornire il tipo di trasporto di immagini in formati compressi a larghezza di banda ridotta applicabile al nodo web_video_server è il package **image_transport**.

Lavorando con le immagini, è desiderabile adottare strategie di trasporto specializzate, come l'utilizzo della compressione delle immagini o dei codec video in streaming. Questo package fornisce classi e nodi per il trasporto delle immagini: ciò viene realizzato mediante appositi plugins. Le tipologie di trasporto offerte dal package image_transport sono le seguenti:

- **image_transport/compressed**: pubblica una *CompressedImage* utilizzando un tipo di compressione JPEG o PNG;
- **image_transport/compressedDepth**: pubblica una *CompressedDepthImage* utilizzando un tipo di compressione PNG;
- **image_transport/raw** (default publisher): pubblica l'immagine così com'è;

- **image_transport/theora**: pubblica un pacchetto stream video codificato utilizzando *Theora* [6].

Utilizzo di image_transport con web_video_server Il suddetto package può essere applicato al nodo web_video_server. Tuttavia, non essendo presente nella suite di ROS, bisogna installarlo digitando il seguente comando:

```
$ sudo apt-get install ros-kinetic-image-transport
$ source /opt/ros/kinetic/setup.bash
```

Una volta installato il package, lo si utilizza, generalmente, allorquando bisogna avviare il nodo web_video_server. Infatti, è possibile settare il parametro **_image_transport** sulla base delle scelte che vengono fatte. I parametri da utilizzare in potenza sono quelli definiti antecedentemente. Ad esempio, si setti il trasporto di immagini compresse come di seguito:

```
$ rosrun web_video_server web_video_server
   _image_trasnsport:=compressed
```

2.6 Librerie JavaScript

Nella programmazione web, **JavaScript** è un linguaggio di scripting orientato agli oggetti e agli eventi. Esso viene utilizzato comunemente lato client per la realizzazione di siti e applicazioni web.

In particolare, mediante l'utilizzo di funzioni di script invocate da eventi innescati dall'utente sulla pagina web in uso, la pagina presenta effetti dinamici e risulta interattiva. Queste funzioni di script, sia inserite all'interno di un file HTML5, che in file separati con estensione *.js*, rendono la pagina provvista di effetti dinamici e sezioni interattive per l'utente.

La presenza di opportune librerie JavaScript facilita al programmatore la realizzazione di pagine web e le rende adatte ai vari framework messi a disposizione nell'ambito della programmazione web.

6. Theora è un codec video progettato dalla Xiph.Org Foundation per competere con il video MPEG-4, RealVideo, Windows Media Video, e simili schemi di compressione video a basso bit rate. Theora fornisce il layer video nei flussi multimediali di tipo Ogg.

Anche il mondo della robotica, ed in particolare il framework ROS, mette a disposizione varie librerie JavaScript utili al programmatore per realizzare applicazioni web atte ad interagire sia con i robot che con i sensori presenti.

2.6.1 ROS JavaScript Library

La libreria JavaScript, maggiormente utilizzata nel primo approccio al mondo ROS per la realizzazione di applicazioni web, è **roslibjs**.

Roslibjs è la libreria JavaScript principale per interagire con ROS dal Browser Web. Per connettersi con ROSBridge, essa utilizza le Web Socket. Fornisce un modo semplice per gestire il paradigma di comunicazione *publish/subscribe* utilizzando oggetti JSON serializzati. Mette a disposizione, inoltre: chiamate di servizio, *actionlib*, analisi URDF e altre funzionalità essenziali in ROS.

L'utilizzo congiunto della libreria JavaScript roslibjs e di ROSBridge cela, certamente, la complessità del sistema ROS agli sviluppatori web e ne estende il suo utilizzo. Ciò consente di realizzare in maniera più semplice applicazioni robot e, magari, espandere il campo della robotica per gli sviluppatori di applicazioni web. Obiettivo di tale libreria è quello di rendere universale l'utilizzo nei vari Browser Web (Safari, Opera, Chrome e Firefox).

La libreria roslibjs supporta molte funzioni complesse per la visualizzazione e interazione con sofisticati algoritmi di manipolazione e navigazione basati su ROS. Tuttavia, esistono molte librerie JSON per semplificare la costruzione e meno inclini agli errori.

Roslibjs è stato progettato per soddisfare le esigenze degli sviluppatori con esperienza di programmazione web. Ci sono molti vantaggi nello sviluppare applicazioni robot lato Browser Web. Si tratta, difatti, di interfacce ampiamente utilizzate anche da utenti non esperti. Fornisce agli utenti approfondimenti su nuove applicazioni per la robotica, nonché uno strumento per migliorare le proprie conoscenze. JavaScript offre un'interfaccia utente rapida e flessibile. Le applicazioni sviluppate all'interno di un Browser Web possono essere trasferite su più piattaforme, e gli aggiornamenti, congiuntamente alle funzionalità, possono essere facilmente aggiunti.

Caratteristiche di roslibjs Le caratteristiche messe a disposizione dalla suddetta libreria JavaScript sono le seguenti:

- **Esposizione dei servizi e dei topic ROS**: gli utenti hanno la possibilità di creare un oggetto roslibjs per l'interazione con un robot che utilizza i servizi e i topic ROS. Inoltre, vengono fornite le interfacce utili per pubblicare e sottoscriversi ai vari topic offerti da ROS. Gli utenti possono pubblicare i topic tramite un apposito comando. I servizi offerti dalla libreria roslibjs sono i seguenti:
  - accesso ai topic messi a disposizione;
  - enumerazione dei topic messi a disposizione;
  - accesso al tipo di messaggio del topic;
  - accesso agli oggetti associati al tipo richiesto;
  - gestione dell'autenticazione.
- **Sicurezza**: quando si sviluppa un'applicazione web robotica disponibile su Internet ad utenti indefiniti, la sicurezza risulta essere fondamentale. Le forme di sicurezza messe a disposizione dalla libreria roslibjs sono due:
  - *servizi e topic protetti*: necessari quando sono presenti servizi critici con i quali il cliente non dovrebbe interferire: si tratta del meccanismo *straight-forward*;
  - *autorizzazione delle chiavi*: consente allo sviluppatore di limitare l'accesso a un'applicazione web robotica ad utenti specifici. L'autenticazione è basata su JSON con padding (jsonp): se fornito con l'URL di un appropriato servizio web, roslibjs lo userà per autenticare potenziali clienti. Una volta autorizzato, un cliente può eseguire le sue azioni fino al termine dell'autorizzazione stessa. Sulle macchine dove sono disponibili le librerie SSL, roslibjs supporterà automaticamente SSL.
- **Registrazione dei dati**: è presente un meccanismo di registrazione dei dati da esperimenti. Questi possono essere salvati localmente su un file che può essere caricato. È possibile, altresì, memorizzare le informazioni lato server, per esempio nel caso di studi dell'utente, come Robot Learning from Demonstration (LfD) e Human Robot Interaction.

Decisione progettuale Inizialmente, l'obiettivo è stato quello di realizzare una libreria in grado di utilizzare le Web Socket messe a disposizione da HTML5 come livello di trasporto. Questo permetteva una comunicazione bidirezionale tra Browser Web mediante una connessione TCP/IP.

Tuttavia, roslibjs utilizza, attualmente, un'implementazione Web Socket personalizzata poiché non esiste ancora una versione standard. Uno dei vantaggi dell'utilizzo delle Web Socket è la differenza, in potenza, tra il server che ospita l'applicazione robot rispetto a quello che ospita roslibjs.

Classi roslibjs Le classi messe a disposizione dalla libreria roslibjs sono molteplici. Se ne indichino alcune di seguito:

- **Message**: viene utilizzato per la pubblicazione e la sottoscrizione di topic; l'unico parametro messo a disposizione è il *value*.
- **Ros**: gestisce la connessione al server e tutte le interazioni con ROS. Emette i seguenti eventi:
  - *error*: se è presente, c'è stato un errore con ROS;
  - *connection*: connesso al Web Socket server;
  - *close*: disconnesso dal Web Socket server;
  - *<topicName>*: messaggio proveniente da rosbridge con un dato topic name;
  - *<serviceID>*: risposta di servizio proveniente da rosbridge con un dato ID.

  I parametri messi a disposizione dalla classe sono i seguenti:
  - *url* (optional): l'URL Web Socket per rosbridge o l'URL del nodo server per connettersi tramite *socket.io*;
  - *groovyCompatibility* (default: true): non utilizza le interfacce modificate dopo l'ultima release groovy o rosbridge_suite e gli strumenti correlati;
  - *transportLibrary* (optional) (default: *websocket*): una delle istanze che controlla come viene creata la connessione (websocket, socket.io, RTCPeerConnection);
  - *transportOptions* (optional): opzioni da utilizzare durante la creazione di una connessione. Attualmente utilizzato solo se *transportLibrary* è RTCPeerConnection.

  Di seguito elenchiamo alcuni metodi messi a disposizione dalla classe:
  - *connect(url)*: connessione alla specifica Web Socket;

  - *close()*: chiusura della connessione con il Web Socket server specifico;
  - *getMessageDetails(callback, message)*: recupera i dettagli relativi al messaggio ROS specifico;
  - *getNodeDetails(node, callback)*: recupera la lista dei topic iscritti e pubblicati e dei servizi di un nodo specifico;
  - *getNodes(callback)*: recupera la lista dei nodi attivi in ROS;
  - *getServices(callback)*: recupera la lista dei servizi attivi in ROS;
  - *getTopics(callback)*: recupera la lista dei topic attivi in ROS sottoforma di array.

- **Service**: un servizio client ROS. I metodi messi a disposizione sono i seguenti:

  - *advertise(callback)*: ogni qualvolta un messaggio viene pubblicato per l'argomento specificato, la callback è richiamata con l'oggetto messaggio;
  - *callService(request, callback, failedCallback)*: chiama il servizio e restituisce la risposta del servizio nella callback.

- **Topic**: pubblica o sottoscrive un topic in ROS. Emette i seguenti eventi:

  - *warning*: presenza di avvisi durante la creazione del topic;
  - *message*: dati del messaggio da rosbridge.

  I parametri messi a disposizione dalla classe sono i seguenti:

  - *ros*: gestisce la connessione ROSLIB.Ros;
  - *name*: nome del topic;
  - *messageType*: tipi di messaggio;
  - *compression*: tipo di compressione utilizzata;
  - *throttle_rate*: rate (in ms tra i messaggi) al quale limitare i topic;
  - *queue_size* (default: 100): coda creata lato bridge per la pubblicazione dei topic;
  - *latch*: blocco del topic durante la pubblicazione;

- *queue_length* (default: 0): lunghezza della coda lato bridge utilizzata durante l'iscrizione;
- *reconnect_on_close* (default: true): flag per abilitare la riscrittura e la lettura sull'evento *close*.

Di seguito elenchiamo alcuni metodi messi a disposizione dalla classe:

- *advertise()*: registrazione del topic come publisher;
- *publish(message)*: pubblicazione del messaggio;
- *subscribe(callback)*: ogni volta che un messaggio viene pubblicato per lo specifico topic, verrà richiamata la callback con l'oggetto messaggio;
- *unadvertise()*: annullamento della registrazione del topic come publisher;
- *unsubscribe(callback)*: annullamento della registrazione del topic come subscriber. Viene arrestata la sottoscrizione ad uno specifico topic e di conseguenza non vengono ricavate le informazioni del topic specifico.

2.6.2 2D Visualization JavaScript Library

È possibile realizzare delle mappe in JavaScript per la visualizzazione in 2D dell'ambiente con cui il robot sta interagendo. Lo standard JavaScript 2D per la gestione delle mappe in ROS è definito **ros2djs**. Molte funzioni standard di ROS, come le mappe, sono incluse in questa libreria.

Diverse classi sono presenti all'interno della libreria; ne citiamo alcune per completezza:

- **ImageMap**: l'immagine della mappa è in formato PNG scalata alle dimensioni definite nella classe *OccupancyGrid*;
- **NavigationArrow**: un indicatore di navigazione che può essere utilizzato per mostrare l'orientamento;
- **OccupancyGrid**: è in grado di convertire un messaggio della griglia di occupazione ROS in un oggetto Bitmap.

2.6.3 3D Visualization JavaScript Library

Oltre alla visualizzazione in 2D, JavaScript mette a disposizione una libreria per la visualizzazione in 3D dell'ambiente in cui si trova il robot. Tale libreria è definita **ros3djs**. Questa sfrutta le potenzialità delle librerie JavaScript *roslibjs* e *three.js*. Molte funzioni standard di ROS come i marcatori interattivi, URDF e le mappe sono incluse all'interno di questa libreria.

Tra le principali classi presenti in tale libreria, ricordiamo le seguenti:

- **Marker**: è in grado di convertire un messaggio marker ROS in un oggetto *three*;
- **Odometry**: un client che sfrutta le potenzialità dell'odometria mediante opportuni parametri;
- **Polygon**: un client PolygonStamped che ascolta un determinato topic e visualizza il poligono.

2.6.4 Keyboard Teleoperation JavaScript Library

A supporto della ROS JavaScript Library, è piuttosto utile l'utilizzo di un ulteriore strumento che consente, in maniera semplice, di associare i tasti presenti nella tastiera del PC per il monitoraggio del robot. Tale libreria JavaScript prende il nome di **keyboardteleopjs**.

Il widget principale associa una serie di tasti della tastiera che invierà i messaggi di tipo *geometry_msgs/Twist* ad uno specifico topic ROS.

L'unica classe implementata all'interno della libreria prende il nome *Teleop*. Questa classe gestisce la connessione al server e tutte le interazioni con ROS. I possibili parametri da settare sono i seguenti:

- **ros**: gestione della connessione ROSLIB.Ros;
- **topic** (optional): il topic ROS da pubblicare;
- **throttle** (optional): un valore costante di velocità.

Capitolo 3

Monitoraggio remoto di droni e rovers

L'uso di piattaforme web costituisce una risorsa di necessità sempre maggiore, a tal punto che, ad oggi, risulterebbe impensabile sottrarsene. L'utente medio tenta di utilizzare sempre più le infinite potenzialità di questi strumenti. La navigazione, sovente, riguarda ricerche d'informazione, cultura personale o la risoluzione a qualche dubbio.

Navigare su Internet significa anche saper interagire attivamente mediante applicazioni web. Siti di blog (web-log), aziendali, istituzionali, di commercio elettronico (e-commerce), social network, forum di discussione e applicazioni cloud sono, oggigiorno, una componente vitale del mondo web di cui l'utente non può farne a meno.

In informatica, il termine *Applicazione Web* indica tutte quelle piattaforme distribuite di tipo Web Based. Risulta accessibile via web per mezzo di una rete, come ad esempio una Intranet all'interno di un sistema informatico, oppure attraverso la rete Internet, dove viene sfruttata un'architettura di tipo *client/server*.

Una web application può essere realizzata per diversi scopi. Come già asserito in precedenza, è possibile sia creare un'interfaccia web per diffondere informazioni di carattere personale, come nel caso di un blog, che fornire informazioni riguardo l'azienda di appartenenza, tipico dei siti aziendali. È possibile anche interagire attivamente, come nel caso di acquisti e vendita di prodotti online presso siti di commercio elettronico. Inoltre, tramite l'uso di social network, gli utenti possono condividere esperienze, gusti ed interessi con la sfera sociale.

Attualmente, un maggiore peso nel mondo dell'informatica continua ad avere l'analisi dei dati che, se utilizzati correttamente, possono essere d'aiuto nelle scelte progettuali future. Ricavare informazioni sull'ambiente, attraverso opportune tecnologie, diventa interessante per migliorare la qualità di un servizio. Certamente l'uso di tool appropriati e dispositivi mobili in grado di muoversi liberamente nell'ambiente circostante permette una riduzione dei tempi dovuti alla raccolta e analisi dei dati.

Risulta interessante approfondire un campo della robotica di servizio che mette a disposizione vari operatori mobili (droni e rovers) in grado di acquisire informazioni necessarie all'utente mediante l'uso di opportuni sensori. Pertanto, realizzare un'interfaccia web in grado di monitorare da remoto il dispositivo mobile ed i sensori ad esso associati, permette di osservare il flusso di dati ricavati in un certo lasso di tempo senza che questi siano salvati ed in seguito analizzati.

3.1 Architettura dell'applicazione web

Per poter realizzare un'interfaccia web in grado di monitorare da remoto dispositivi mobili usufruendo del framework ROS, è fondamentale, anzitutto, configurare l'ambiente in modo tale che l'applicazione web sia in grado di comunicare correttamente.

La matrice di questo elaborato è data dalla realizzazione di uno schema logico in grado di mostrare come il framework ROS lavori e come sia in grado di interagire con la nostra interfaccia web mediante opportuni componenti. Lo schema logico realizzato prevede una serie di componenti, i quali sono in grado di comunicare tra di loro per mezzo di un nodo «master». Mediante quest'ultimo nodo, è possibile realizzare un ambiente configurato, all'interno del quale i nodi sono in grado di scambiarsi opportune informazioni tra di loro.

Nella rappresentazione logica dell'architettura dell'applicazione web è prevista una componente dedicata al Simulatore Gazebo. Tuttavia, questa è stata utilizzata solamente per le prove di teleoperazione mediante un supporto virtuale. Una volta testati i topic pubblicati dal nodo Gazebo, è stata posta l'attenzione sull'interazione tra l'applicazione web ed il Clearpath Jackal. Questa è descritta all'interno del quinto capitolo, dove vengono illustrati i passi per la configurazione del rover e come sia possibile teleoperarlo mediante l'interfaccia web **ROS Web Interface**.

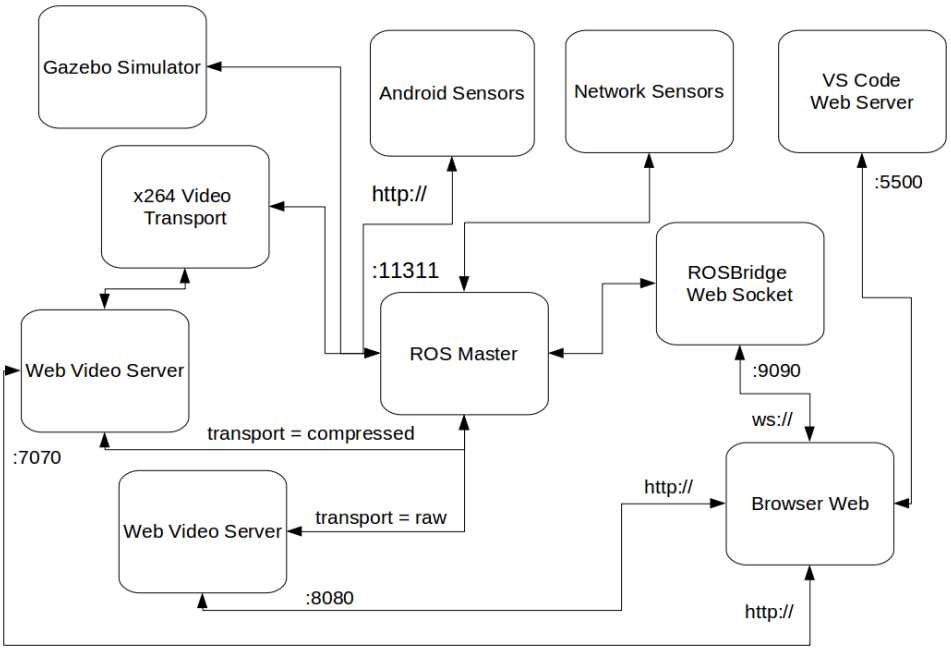


Figura 3.1: Schema logico dell'architettura dell'applicazione web

Nella Figura 3.1, è possibile individuare i vari componenti che interagiscono tra di loro per la realizzazione dell'ambiente su cui, successivamente, la nostra interfaccia web sarà in grado di lavorare.

Prima di entrare nel dettaglio dei vari componenti utilizzati per la configurazione dell'ambiente ROS, è fondamentale andare a settare delle variabili d'ambiente che saranno necessarie in seguito. Tra quelle messe a disposizione dal framework ROS, le uniche da configurare sono le seguenti:

- **ROS_MASTER_URI**: indica ai nodi ROS dove possono individuare il master. Il valore di questa variabile coinciderà con l'indirizzo IP associato al PC in uso collegato alla rete. Il valore può essere settato nel seguente modo:

  ```
  $ export ROS_MASTER_URI=http://ip_address:11311/
  ```

- **ROS_HOSTNAME**: si tratta di una variabile d'ambiente opzionale che imposta l'indirizzo di rete dichiarato da un nodo oppure da uno strumento ROS (per esempio, l'utilizzo di un robot). Il valore di questa

variabile, così come definito in precedenza, coinciderà con l'indirizzo IP associato al PC. La configurazione di tale variabile avviene nel seguente modo:

```
$ export ROS_HOSTNAME=ip_address
```

- **ROS_PACKAGE_PATH**: variabile d'ambiente facoltativa che consente di aggiungere più pacchetti ROS dall'origine all'ambiente configurato. Può essere composta da uno o più percorsi separati dallo standard separatore del sistema operativo (":" nel caso Unix). Nel nostro caso, è opportuno settare il percorso relativo alle librerie messe a disposizione dal framework ROS, oltre ai packages realizzati sui sensori di rete e sulle informazioni ricavate dal dispositivo android:

  ```
  $ export ROS_PACKAGE_PATH=/home/name_pc/packages/
      src:/opt/name_pc/kinetic/share
  ```

Tuttavia, qualora si decida di avviare i singoli componenti da terminale, è necessario ripetere i comandi precedenti allorquando si apre un nuovo terminale. Diversamente, è possibile usufruire di un apposito script, realizzato in bash, che permetta di configurare l'ambiente in maniera automatica. Lo vedremo in dettaglio nella prossima sezione, introducendo anche il tool *screen* presente nei sistemi operativi Unix.

3.1.1 ROS Master

Il «cuore» del framework ROS è rappresentato dal master. Esso permette l'inizializzazione dell'ambiente. Affinché i nodi ROS siano in grado di comunicare tra di loro, è necessario eseguire il core di ROS. Il comando utilizzato per avviarlo è il seguente:

```
$ roscore
```

Come è possibile notare dalla Figura 3.1, tutti i nodi comunicano con il master. Ognuno di essi è stato configurato in modo tale da connettersi al master e comunicare con gli altri nodi mediante l'utilizzo dei messaggi.

3.1.2 ROSBridge Web Socket

ROSBridge Web Socket, come già descritto nel capitolo precedente, fornisce uno strato di livello intermedio in grado di interfacciare l'applicazione

web al framework ROS. Anche in questo caso, per avviare il nodo, è possibile utilizzare il seguente comando:

```
$ roslaunch rosbridge_server rosbridge_websocket.launch
```

Nel momento in cui viene avviato il nodo *rosbridge_server*, viene creata una Web Socket che è in grado di comunicare con l'interfaccia web, dato un certo indirizzo IP e un valore della porta. Il valore standard della porta è 9090.

3.1.3 Web Video Server

Il package *web_video_server* fornisce un nodo in grado di garantire lo streaming HTTP di immagini che provengono da diversi topic ROS. Osservi dalla Figura 3.1: il nodo è stato avviato per trasmettere due tipologie di immagini. I comandi utilizzati per avviare il suddetto nodo sono i seguenti:

```
$ rosrun web_video_server web_video_server __name:=
    video_compressed _port:=7070 _image_transport:=
    compressed
$ rosrun web_video_server web_video_server __name:=
    video_raw _port:=8080 _image_transport:=raw
```

Entrambi i comandi settano un valore definito *__name*: ciò è dovuto all'impossibilità di avviare lo stesso nodo più di una volta senza aver impostato un nome specifico. Inoltre, nel primo comando è stato scelto come valore della porta 7070, mentre nel secondo caso è stato scelto 8080. Questo perché non è possibile avviare due nodi che siano, ambedue, sulla stessa porta.

Un ulteriore parametro settato per definire la tipologia di dati trasmessi sulla connessione HTTP è *_image_transport*. Come già introdotto nel precedente capitolo, è possibile scegliere la tipologia di dati che viene trasmessa all'interno delle connessioni HTTP. In questo caso, è stata scelta la porta 8080 per trasmettere immagini di tipo *raw* e la porta 7070 per la trasmissione di dati nel formato *compressed*.

3.1.4 x264 Video Transport

Il package *x264_video_transport* è stato realizzato appositamente per la trasmissione delle immagini sfruttando il codec video *H264*. Tale package, la cui installazione viene effettuata sfruttando il file bash *install_packages.sh*, sfrutta due componenti fondamentali:

- **x264_bridge**: ha il compito di modificare il topic utilizzato per la trasmissione delle immagini *compressed*, e ripubblicarlo sfruttando la codifica *H264*;
- **x264_image_transport**: la libreria mette a disposizione la codifica video *H264* per le immagini pubblicate.

Per poter avviare tale nodo, è opportuno eseguire il seguente comando:

```
$ roslaunch x264_video_transport playback.launch name:=
    topicName
```

Il nodo descritto, tuttavia, deve essere avviato manualmente dall'utente, in quanto il comando di *roslaunch* non è stato inserito all'interno del file bash *init.sh*. Questo è dovuto al fatto di non sapere, dapprima, quale sia la codifica utilizzata per la trasmissione delle immagini. Se si decide, a priori, di trasmettere le immagini nel formato *H264*, allora tale nodo deve essere avviato prima di eseguire l'applicazione web. Nel caso in cui, invece, si vogliano acquisire le immagini nel formato *jpeg*, non è opportuno avviare il suddetto nodo. La sola configurazione contenuta all'interno di *init.sh* è sufficiente alla corretta esecuzione dell'interfaccia web. La variabile *topicName* individua il nome del topic dal quale è possibile acquisire le immagini multimediali trasmesse. Un esempio di esecuzione del succitato nodo è il seguente:

```
$ roslaunch x264_video_transport playback.launch name
    :=/phoneName/android/camera
```

Affinché tale package possa essere utilizzato, è opportuno aver installato nel proprio sistema operativo la libreria **FFmpeg** [7]. Per poterne usufruire, è necessario eseguire i prossimi comandi d'installazione:

```
$ sudo apt-get update
$ sudo add-apt-repository ppa:jonathonf/ffmpeg-4
$ sudo apt-get install ffmpeg
```

7. Si tratta di una libreria software in grado di convertire e riprodurre audio e video. Tale suite si basa sulle librerie *libavcodec*, *libavformat* e *libavutil* in grado di codificare audio e video. Tale strumento è gratuito e open source per i sistemi Unix. È possibile impostare frequenze di campionamento e ridimensionare i video.
Cfr. https://www.ffmpeg.org

3.1.5 Gazebo Simulator

Il simulatore *Gazebo* è uno strumento fondamentale che permette di verificare, in tempo reale, i movimenti che vengono effettuati dal robot ad ogni comando impartito dall'utente. Questo simulatore è stato utile per testare la teleoperazione, presente nell'interfaccia web, mediante un apposito joystick. Tuttavia, il simulatore non beneficia dei sensori di cui dispone lo smartphone, dal quale sono state condotte varie misurazioni. Pertanto, è stato necessario l'utilizzo di quest'ultimo per concludere il lavoro di tesi.

Ad ogni modo, per avviare il simulatore Gazebo contenente TurtleBot, è possibile utilizzare il seguente comando:

```
$ roslaunch turtlebot_gazebo turtlebot_world.launch
```

È possibile, invece, eseguire il simulatore Gazebo per il robot Clearpath Jackal avviandolo nel seguente modo:

```
$ roslaunch jackal_gazebo jackal_world.launch
```

Una volta avviato il simulatore Gazebo, questo mette a disposizione diversi nodi e relativi topic con i quali è possibile ricavare importanti informazioni. Una di queste, in particolare, riguarda la telecamera presente nel TurtleBot. Pertanto appare motivata la scelta di utilizzare due nodi web_video_server in grado di gestire immagini di tipo *compressed* provenienti dal dispositivo android ed immagini di tipo *raw* provenienti dalla kinect presente nel TurtleBot. Queste informazioni viaggiano su connessioni HTTP diverse in quanto la tipologia di dato trasportato è differente.

3.1.6 Android Sensors

L'interfaccia web realizzata per il monitoraggio del robot, tuttavia, prevede delle sezioni che permettono di osservare l'andamento di opportuni sensori presenti nei dispositivi android. Per questo motivo, mediante l'utilizzo di uno smartphone in cui è installata l'applicazione **PIC4SeR Monitoring**, è possibile ricavare i dati relativi ad alcuni dispositivi di controllo: i sensori di movimento (accelerometro, giroscopio, vettore di rotazione); i sensori d'ambiente (illuminanza, termometro, barometro, campo magnetico, coordinate di geolocalizzazione); i sensori di rete (potenza e velocità del segnale) e informazioni relative alla cella a cui è connesso lo smartphone (potenza e velocità del segnale LTE, RSRP, RSRQ, RSSNR, etc.).

Affinché sia possibile ricavare i dati dai sensori, è necessario connettere il dispositivo android alla stessa rete a cui è già connesso il PC. È opportuno, altresì, indicare l'indirizzo IP con cui è individuato il PC quando bisogna scegliere il master a cui connettere lo smartphone. Si ricordi, inoltre, che quest'ultimo funge da vero e proprio nodo ROS. A questo punto, il dispositivo android sarà connesso al master ROS e fungerà da nodo, mettendo a disposizione i topic e i servizi presenti in esso.

3.1.7 Network Sensors

Le informazioni ricavate dal dispositivo android non sono le uniche messe a disposizione dall'interfaccia web. Infatti, è stato realizzato uno script in Python che permette di monitorare la scheda di rete NIC presente all'interno del PC, ricavando: i valori di banda occupata in download e upload; i valori della potenza del segnale RSSI; la qualità del segnale.

Lo script Python è in grado di generare un nuovo nodo. Questo verrà collegato automaticamente al master ed, una volta avviato, è possibile usufruire di tutte le informazioni pubblicate. Inoltre, i topic messi a disposizione permettono di ricavare i valori rappresentati graficamente secondo un determinato andamento temporale.

Il package realizzato può essere installato eseguendo lo script messo a disposizione:

```
$ bash install_packages.sh
```

Questo script permette di compilare il codice sorgente in Python e di realizzare, mediante il comando *catkin_make*, un vero e proprio nodo *network_sensor* provvisto di un topic, il quale pubblica informazioni di tipo *NetworkSensor*. Entreremo più nel dettaglio nella sezione che approfondisce la realizzazione del codice per la generazione di questo nodo.

Tuttavia, l'esecuzione del suddetto script permette, altresì, di generare e includere all'interno del framework ROS, installato sul PC, i messaggi che verranno pubblicati dall'applicazione android e il package utilizzato per la codifica H264: *x264_video_transport*.

3.1.8 VS Code Web Server

Dal momento in cui l'ambiente è stato configurato opportunamente, non resta che avviare l'interfaccia web. Questo può essere fatto utilizzando semplicemente il file *index.html* presente all'interno della cartella *src* del progetto, oppure è possibile utilizzare un server virtuale messo a disposizione da Visual Studio Code. Tale server è definito **Live Server**.

Live Server è un'estensione presente in Visual Studio Code scaricabile gratuitamente dall'apposita sezione *Extensions*. È in grado di avviare un server di sviluppo locale con funzionalità di ricarica live per pagine statiche e dinamiche. Tale server è avviato sulla porta 5500 cliccando sul tasto *Go Live* presente nella status bar in Visual Studio Code. Il tasto permette di avviare e arrestare il server.

All'utente, pertanto, si consiglia: la prima modalità per provare l'interfaccia web; la seconda affinché modifichi il codice sorgente e voglia vedere in tempo reale le modifiche apportate.

3.2 Configurazione dell'ambiente di lavoro

L'ambiente di lavoro, come precedentemente illustrato, può essere configurato manualmente da parte dell'utente. Tale configurazione, tuttavia, prevede una minima conoscenza, da parte dell'utilizzatore, sia di interazione con il terminale, che dei vari packages che bisogna installare prima di configurare il tutto.

Tuttavia, è possibile configurare in maniera automatica l'ambiente di lavoro avvalendosi dell'apporto di uno script, realizzato in bash, presente all'interno del codice sorgente. Affinché sia possibile usufruire di questo script è opportuno installare sul proprio sistema operativo un tool presente nei comuni sistemi operativi Unix: **GNU Screen**.

3.2.1 GNU Screen

GNU Screen è un emulatore di terminale sviluppato dal Progetto GNU. L'utente può accedere sia a sessioni di terminale multiple che separate, tutto simultaneamente. Può, quindi, gestire più programmi da riga di comando e separare i vari terminali in cui è stato avviato.

Fornisce talune caratteristiche che permettono all'utente di usare i programmi in una singola interfaccia in maniera produttiva:

- **Persistenza**: permette all'utente di avviare applicazioni da un computer. Inoltre, connettendosi da un altro computer, è possibile continuare ad usare la stessa applicazione senza farla ripartire.
- **Finestre multiple**: è possibile creare sessioni multiple, ognuna delle quali ospita una singola applicazione. L'utente può passare da una finestra ad un'altra con il solo uso della tastiera.
- **Condivisione di sessione**: più computer possono connettersi alla medesima sessione nello stesso lasso di tempo, permettendo così la collaborazione tra più utenti. Lo stesso computer può essere utilizzato per effettuare più connessioni simultanee.

Mediante l'utilizzo di screen, è stato possibile realizzare uno script in grado di eseguire più nodi contemporaneamente.

3.2.2 Script di configurazione *init.sh*

La configurazione dell'ambiente di lavoro può essere realizzata in maniera automatica mediante l'ausilio dello script *init.sh*. Questo permette di avviare il core del framework ROS ed i vari nodi utili affinché l'applicazione web funzioni in modo corretto.

Per eseguire lo script, è necessario utilizzare il seguente comando:

```
$ bash init.sh name_pc ip_address
```

Lo script viene eseguito correttamente se vengono passati sulla linea di comando due argomenti. Il primo individua il nome del PC, ovvero il nome dell'account che identifica il proprio personal computer. Il secondo, invece, è un indirizzo IP valido che verrà utilizzato per configurare correttamente l'ambiente di lavoro. Questo indirizzo IP verrà salvato all'interno delle variabili globali *ROS_MASTER_URI* e *ROS_HOSTNAME*: la prima è utile per permettere al dispositivo android di comunicare con l'applicazione web tramite il master ROS e ROSBridge; la seconda, invece, serve per identificare il personal computer con il quale il rover dovrà comunicare.

Lo script è stato realizzato usufruendo del comando *screen*. A questo comando, sono stati passati vari argomenti, tra cui *-dm* [8] e *-S* [9]. Tutti i nodi avviati mediante il suddetto comando sono identificati da un nome.

Ciascun nodo, utilizzato per configurare l'ambiente, viene eseguito all'interno di un daemon. Al comando screen, difatti, viene passato come ulteriore parametro il nome del file *.sh* al cui interno sono contenuti i comandi visti in precedenza per eseguire i nodi manualmente.

In questo modo, è possibile utilizzare un unico comando per avviare più nodi contemporaneamente. Inoltre, screen mette a disposizione vari comandi, tra cui:

- *screen -ls*: per visualizzare quali sono i nodi eseguiti nei vari terminali avviati in background;
- *screen -r [name]*: per visualizzare il contenuto del terminale associato al nodo passato come parametro;
- *Ctrl+A+D*: permette di effettuare il detach di un deamon quando si è all'interno del terminale di esecuzione;
- *Ctrl+C*: forza l'uscita dal terminale concludendo l'esecuzione dello specifico deamon;
- *pkill screen*: vengono arrestati tutti i deamon lanciati in precedenza.

3.3 Sviluppo di un package ROS per l'analisi della rete

L'applicazione web, realizzata per monitorare da remoto l'ambiente circostante al rover o drone, prevede una sezione dedicata all'analisi dei dati provenienti dai sensori di rete. In particolare, accedendo alla sezione *Network Sensors*, l'utente può monitorare da remoto, contemporaneamente: lo stato della scheda di rete NIC presente all'interno del proprio PC; i dati di rete (RSSI, Link Speed) provenienti dallo smartphone mediante l'applicazione android realizzata appositamente per la trasmissione delle informazioni.

8. Il parametro -dm specifica al tool *GNU Screen* di avviare la sessione del terminale in background.
9. Il parametro -S permette di associare un nome alla sessione di terminale avviata.

NIC (*Network Interface Controller*, anche *Network Interface Card*), in ambito informatico, è costituita a livello hardware da una scheda elettronica che, solitamente, viene inserita all'interno di un personal computer, server, stampante, etc. Essa svolge le funzioni necessarie affinché l'apparato informatico sia in grado di connettersi ad una rete informatica.

Mediante l'utilizzo di un apposito script realizzato in Python, è stato possibile effettuare un monitoraggio della qualità della rete, analizzando sia la potenza del segnale, indicata dall'acronimo RSSI (*Received Signal Strength Indication*), sia l'occupazione della banda in download e upload, oltre alla qualità del segnale.

3.3.1 Creazione di un package ROS

Il framework ROS mette a disposizione vari comandi che permettono di creare in maniera automatica degli schemi di packages ROS. Questi, una volta compilati, sono utili per realizzare opportuni nodi in grado di ricavare varie informazioni sfruttando, ad esempio, i sensori a bordo di un robot oppure, ed è questo il nostro caso, quelli presenti all'interno di un dispositivo android.

Di seguito sono descritti due comandi messi a disposizione dal framework ROS per la realizzazione di un package ROS. È, altresì, spiegato come questi debbano essere utilizzati affinché la creazione vada a buon fine.

roscreate Contiene un tool in grado di creare automaticamente le risorse del filesystem ROS. Esso fornisce *roscreate-pkg* il quale è in grado di generare una nuova directory del package, inclusi i file di build e manifest.xml. I packages, di frequente creati dagli utenti copiando un pacchetto esistente e modificando i suoi file di build e manifest.xml, presentano spesso degli errori: pertanto, lo strumento roscreate-pkg è pensato per essere semplice e meno incline agli errori.

roscreate-pkg è uno strumento da riga di comando per la creazione di un nuovo package ROS. Nel momento in cui viene realizzato il package, esso conterrà i più comuni file di configurazione: manifest.xml, CMakeLists.txt, mainpage.dox e Makefile [10].

10. Un package, affinché possa essere considerato un catkin package, deve contenere: un file *package.xml*, il quale fornisce le meta informazioni sul package; un file *CMakeLists.txt* che usa catkin. I metapackages catkin devono avere un file di testo standard CMakeLists.txt ed, inoltre, non può essere presente più di un package in ogni folder.

Per creare un nuovo package nella directory corrente, bisogna digitare il seguente comando:

```
$ roscreate-pkg pkgname
```

Inoltre, è possibile specificare le dipendenze che si vogliono utilizzare all'interno del package di modo che non debbano essere incluse succesivamente. Tra le varie dipendenze messe a disposizione, le più comuni sono le librerie *roscpp* [11], *rospy* [12] e *std_msgs* [13].

Il comando per creare un package che presenti le suddette dipendenze è il seguente:

```
$ roscreate-pkg pkgname roscpp rospy std_msgs
```

catkin Un altro tool, messo a disposizione nei sistemi Unix, è *catkin*. Può essere utilizzato per la creazione di packages in ROS e, pertanto, è incluso di default quando viene installato ROS. Tuttavia, se la versione di ROS non prevede catkin, può essere incluso nel proprio sistema operativo nel seguente modo:

```
$ sudo apt-get install ros-kinetic-catkin
```

Una volta installato, possono essere sfruttati i comandi per la creazione di un nuovo package ROS, includendo anche le dipendenze (come visto nel precedente comando). Inizialmente, bisogna creare una workspace, in cui è presente una cartella *src*. Una volta conclusa la creazione, è opportuno eseguire il seguente comando a partire dalla directory *name_workspace*:

```
$ catkin_create_pkg pkgname roscpp rospy std_msgs
```

11. Si tratta di un'implementazione in C++ di ROS. Fornisce una libreria client che consente ai programmatori C++ di interfacciarsi rapidamente con i topic, con i servizi e con i parametri ROS. *roscpp* è la libreria client ROS più diffusa ed è progettata per essere la libreria ad alte prestazioni per ROS.
12. È una libreria client Python per ROS. L'API rospy client consente ai programmatori di interfacciarsi rapidamente con ROS. La progettazione di rospy favorisce la velocità di implementazione rispetto alle prestazioni a runtime, di modo che gli algoritmi possano essere rapidamente «prototipizzati» e testati all'interno di ROS. Molti degli strumenti ROS sono scritti in rospy, ad esempio *rostopic* e *rosservice*.
13. Si tratta di messaggi ROS standard i quali includono i tipi comuni (dati primitivi e altri costrutti). Tra i tipi standard sono presenti: Bool, Byte, Char, Float32, Int16, MultiArray, etc.

Questo comando, a sua volta, genera una cartella chiamata *pkgname* contenente un file *package.xml* e *CMakeLists.txt*, parzialmente riempiti con delle informazioni generate proprio dal comando catkin_create_pkg.

3.3.2 Package *ros-network-analysis*

L'applicazione web prevede una sezione dedicata all'analisi della rete. È possibile esaminare opportunamente le informazioni ricavate dalla NIC osservando, nello specifico, i parametri messi a disposizione all'interno di alcuni file contenuti nella cartella di sistema */proc/net*. I file contenuti all'interno di questa directory permettono di analizzare il traffico scambiato dal proprio personal computer con altri dispositivi collegati alla stessa rete.

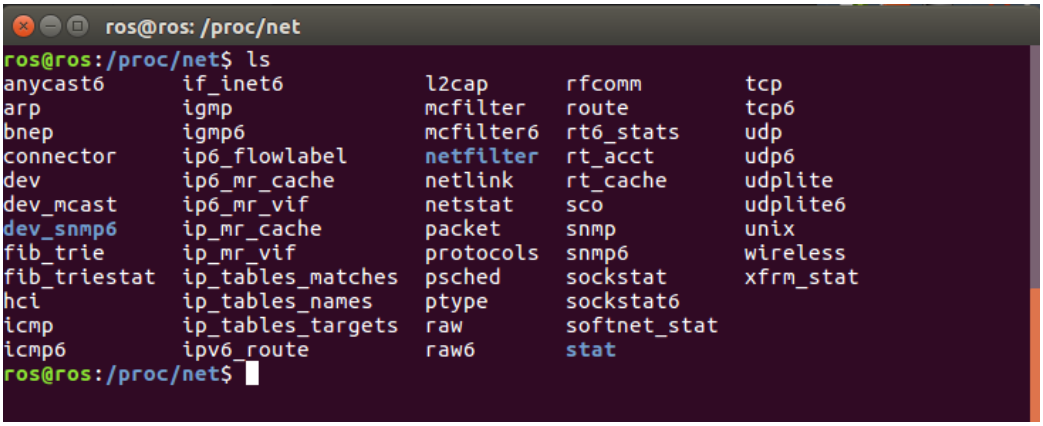


Figura 3.2: File contenuti all'interno della cartella */proc/net*

Come si può notare dalla Figura 3.2, sono presenti diversi file. Ciascuno di essi contiene le informazioni relative ai diversi protocolli di rete: sia quelli di livello applicativo, che quelli di trasporto che di livello di rete.

L'analisi è stata effettuata utilizzando solamente tre di questi file: *dev*, *snmp* e *wireless*.

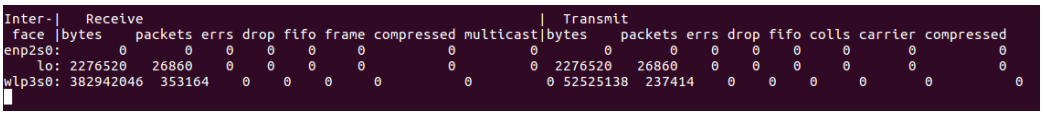


Figura 3.3: Informazioni contenute all'interno del file *dev*

Nella Figura 3.3 sono presenti le informazioni contenute all'interno del file *dev*. Esso mostra la quantità di dati scambiati, sia in trasmissione che in ricezione. Queste informazioni sono espresse in numero di pacchetti e in dimensioni (bytes). I dati scambiati sono suddivisi in base alle interfacce di rete messe a disposizione dal proprio PC. Nel caso specifico, le interfacce di rete presenti sono: *enp2s0* (interfaccia della rete ethernet), *lo* (interfaccia di loopback) e *wlp3s0* (interfaccia della rete wireless). Tutte le informazioni riguardano i dati trasmessi a livello di rete IP.

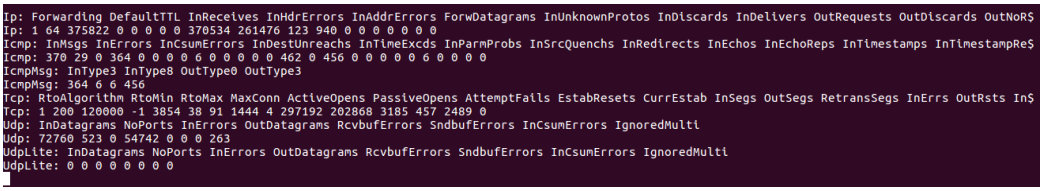


Figura 3.4: Informazioni contenute all'interno del file *snmp*

Nella Figura 3.4 sono presenti, invece, le informazioni relative al file *snmp*. Il file prevede una sezione riguardante i protocolli TCP e UDP, dai quali è possibile ricavare precise informazioni sulla qualità della rete. Dal primo protocollo di trasporto sono stati ricavati i dati riguardanti i segmenti trasmessi e ricevuti (OutSegs e InSegs), mentre dal secondo protocollo sono stati analizzati i datagrammi trasmessi e ricevuti (OutDatagrams e InDatagrams).

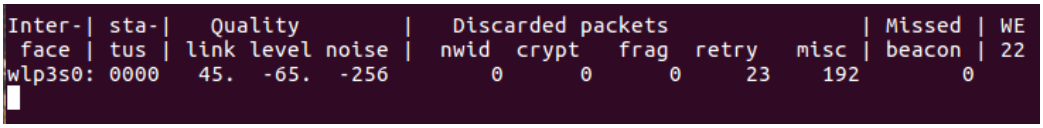


Figura 3.5: Informazioni contenute all'interno del file *wireless*

Nella Figura 3.5 sono riportate le informazioni relative al file *wireless*. Il suddetto file mostra i dati relativi all'interfaccia di rete wireless [14]: la potenza del segnale, il livello del segnale ed il rumore. Di questi tre valori, solamente i primi due sono stati oggetto di analisi. Di seguito è stato realizzato il confronto con quelli ricavati dal dispositivo android.

Il package *ros-network-analysis* è stato realizzato mediante l'utilizzo delle informazioni ricavate in precedenza. Dopo aver creato lo schema del package,

14. L'interfaccia di rete wireless considerata in questo caso è la wlp3s0.

sono stati modificati opportunamente i file package.xml e CMakeLists.txt di modo che questo fosse compilato e reso adatto all'applicazione web. Di seguito vengono mostrati in dettaglio i file che compongono il package.

Package.xml Il file *package.xml* fornisce le meta informazioni del package creato.

```xml
<?xml version="1.0"?>
<package>
  <name>network_analysis</name>
  <version>0.1.0</version>
  <description>The ros-network-sensor package</description>
  <maintainer email="federico_barone@hotmail.it">Federico Barone
  </maintainer>
  <license>BSD</license>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>message_generation</build_depend>
  <build_depend>actionlib_msgs</build_depend>
  <run_depend>rospy</run_depend>
  <run_depend>std_msgs</run_depend>
  <run_depend>message_runtime</run_depend>
  <run_depend>actionlib_msgs</run_depend>
</package>
```

Figura 3.6: File di configurazione *package.xml*

Nella Figura 3.6 è riportato il contenuto del file *package.xml*. Si tratta di un file XML, pertanto, le singole parole chiavi sono individuate dalle parentesi angolari. Sono presenti diversi tag, tra cui le dipendenze, le quali sono state installate di seguito alla creazione del package. Oltre alle dipendenze, il file è, altresì, individuato da: un nome, una versione, una breve descrizione del package e un costruttore.

CMakeLists.txt All'interno del file *CMakeLists.txt* sono presenti le informazioni generate, automaticamente, all'atto della creazione del package. Esso non va assolutamente modificato, in quanto ciò comporterebbe la perdita di dati e una compilazione del package non corretta.

NetworkSensor.msg Nel package *ros-network-analysis* è presente una cartella contenente i tipi di messaggio creati ad hoc ed utilizzati per salvare le informazioni ricavate dalla scheda di rete.

```
Header header

# Name of the wireless interface (e.g. wlan0, wlan1, wlp3s0, etc.)
string iface

# TCP related information on link utilization for a given (NIC) interface
int64 tcp_tx_segments
int64 tcp_rx_segments
float64 tcp_tx_segmentrate
float64 tcp_rx_segmentrate

# UDP related information on link utilization for a given (NIC) interface
int64 udp_tx_datagrams
int64 udp_rx_datagrams
float64 udp_tx_datagramrate
float64 udp_rx_datagramrate

# Total (IP: TCP + UDP) link utilization for a given (NIC) interface
int64 total_tx_packets
int64 total_tx_bytes
int64 total_rx_packets
int64 total_rx_bytes
float64 total_tx_mbps
float64 total_rx_mbps

# Information about the status of wireless
float64 link_quality
float64 signal_level
```

Figura 3.7: File *NetworkSensor.msg*

Nella Figura 3.7 è riportato il contenuto del file NetworkSensor.msg. Questo file contiene le variabili alle quali verranno assegnati i valori pubblicati dal nodo *network_sensor* non appena avviato. Le informazioni contenute riguardano i protocolli di trasporto TCP e UDP e il protocollo di rete IP. Ci si è soffermati, antecedentemente, sul traffico di rete, analizzando sostanzialmente i segmenti e datagrammi trasmessi e ricevuti, oltre ad aver ricavato i valori di velocità di trasmissione e ricezione in Mbps. Queste informazioni sono state acquisite dai file *dev* e *snmp*, mentre i dati riguardanti la qualità del link e il livello del segnale sono stati ricavati dal file *wireless*.

network_sensor.py Affinché il package *ros-network-analysis* venga eseguito correttamente, è necessario realizzare uno script in Python che permetta di ricavare le informazioni contenute all'interno dei file presenti nella directory */proc/net*. Lo script suddetto sfrutta le dipendenze istanziate nel momento della creazione del package, ovvero *rospy* e *std_msgs*.

Il file consta di un main principale in cui viene richiamata la funzione *network_sensor_publisher()* nella quale vengono acquisite le informazioni dalla scheda di rete.

```
# Inside /proc/net it is possible to obtain some values about the quality of the network
cmd = "cat /proc/net/dev | grep " + interface_name
cmd_netstat_tcp = "cat /proc/net/snmp | grep Tcp:"
cmd_netstat_udp = "cat /proc/net/snmp | grep Udp:"
cmd_wireless = "cat /proc/net/wireless | grep " + interface_name

# Create the publisher for ROS
pub = rospy.Publisher('/network_analysis/network_sensor', NetworkSensor, queue_size = 10)
rospy.init_node('network_sensor_publisher', anonymous = True)
```

Figura 3.8: File *network_sensor.py* - Funzione network_sensor_publisher()

Nella Figura 3.8 vengono riportate le linee di codice contenute all'interno della funzione *network_sensor_publisher()* che permettono di ricavare i valori utili per l'analisi della rete. Inoltre, è possibile osservare l'utilizzo della libreria rospy per la creazione del publisher e l'inizializzazione del nodo ad esso associato. In questo modo, è possibile visualizzare, all'interno della lista dei nodi attivi, anche quello creato.

I valori ricavati dalla scheda di rete, affinché possano essere pubblicati sotto forma di messaggio ROS, devono essere salvati all'interno del messaggio creato precedentemente. Tuttavia, non tutti i valori presenti all'interno dei file *dev*, *snmp* e *wireless* sono stati utilizzati: solo alcuni di essi sono stati acquisiti e memorizzati. Per fare ciò, è stato necessario prima suddividere l'array contenente le varie informazioni ed in seguito memorizzare solamente quelle necessarie. Di seguito vengono illustrati i passaggi svolti per ricavare i dati e memorizzarli all'interno delle variabili istanziate nel messaggio NetworkSensor.msg.

```
# Retrieve information about IP: TCP + UDP
f = Popen(cmd, shell = True, stdout = PIPE, stderr = PIPE)
cmd_output = f.stdout.read()
ans = cmd_output.split()
if (len(ans) < 1): return 0
msg.total_tx_packets = int(ans[10])
msg.total_tx_bytes = int(ans[9])
msg.total_rx_packets = int(ans[2])
msg.total_rx_bytes = int(ans[1])
```

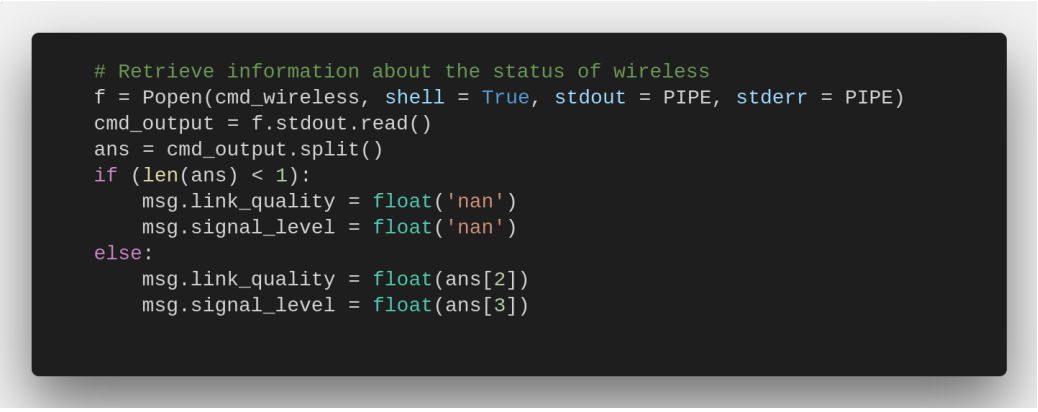
Figura 3.9: File *network_sensor.py* - Informazioni IP

```
# Retrieve information about only TCP
f = Popen(cmd_netstat_tcp, shell = True, stdout = PIPE, stderr = PIPE)
cmd_output = f.stdout.read()
ans = cmd_output.split()
insegs = len(ans) - 6
outsegs = len(ans) - 5
msg.tcp_rx_segments = int(ans[insegs])
msg.tcp_tx_segments = int(ans[outsegs])
```

Figura 3.10: File *network_sensor.py* - Informazioni TCP

```
# Retrieve information about only UDP
f = Popen(cmd_netstat_udp, shell = True, stdout = PIPE, stderr = PIPE)
cmd_output = f.stdout.read()
ans = cmd_output.split()
rxdatagrams = len(ans)/2 + 1
txdatagrams = len(ans)/2 + 4
msg.udp_rx_datagrams = int(ans[rxdatagrams])
msg.udp_tx_datagrams = int(ans[txdatagrams])
```

Figura 3.11: File *network_sensor.py* - Informazioni UDP



```
# Retrieve information about the status of wireless
f = Popen(cmd_wireless, shell = True, stdout = PIPE, stderr = PIPE)
cmd_output = f.stdout.read()
ans = cmd_output.split()
if (len(ans) < 1):
    msg.link_quality = float('nan')
    msg.signal_level = float('nan')
else:
    msg.link_quality = float(ans[2])
    msg.signal_level = float(ans[3])
```

Figura 3.12: File *network_sensor.py* - Informazioni Wireless

Nella Figura 3.9 sono state acquisite le informazioni dal file *dev* e memorizzate all'interno del messaggio precedentemente istanziato nelle variabili *total_tx_packets*, *total_rx_packets*, *total_tx_bytes* e *total_rx_bytes*.

Nelle Figure 3.10 e 3.11 sono state acquisite le informazioni dal file *snmp* riguardanti i protocolli di trasporto TCP e UDP e memorizzati in maniera opportuna (informazioni sui segmenti e datagrammi trasmessi e ricevuti).

Infine, nella Figura 3.12 le informazioni vengono ricavate a partire dal file *wireless* e memorizzate opportunamente all'interno del messaggio NetworkSensor.msg. Quando il proprio personal computer non è collegato alla rete wireless, ma usufruisce di una connessione cablata, queste informazioni risultano nulle. Per tale motivo, è stata inserita una condizione per evitare di visualizzare informazioni errate.

Capitolo 4

Gestione dei dati multimediali

La comunicazione è basata sullo scambio di messaggi secondo delle regole precise. Rappresenta un elemento fondamentale all'interno di una struttura sociale, senza la quale non è possibile trasmettere informazioni tra individui presenti nello stesso luogo (o in luoghi diversi).

In ambito informatico, la comunicazione dei dati [15] avviene secondo un preciso trasferimento di informazioni: un flusso digitale di bits o un segnale analogico digitalizzato è trasmesso su un canale di comunicazione point-to-point o point-to-multi-point. Esistono diversi canali di comunicazione, dai più semplici fili di rame sino ad arrivare alle complesse fibre ottiche, passando per i canali di comunicazione wireless.

Ciò che viene trasferito all'interno di questi canali di comunicazione sono i *dati*. Questi possono presentarsi sotto diverse forme: voce, immagini, video o audio. Utilizzano, altresì, un segnale continuo che varia in ampiezza, fase o qualche altra proprietà in proporzione a quella di una variabile. I dati dipendono dal codice e dal formato utilizzati, inoltre la loro elaborazione può portare alla conoscenza di un'informazione. In informatica, il dato viene tipicamente rappresentato da un valore numerico, il *bit*, il quale può essere elaborato e/o trasformato da un elaboratore elettronico.

Il termine *multimedialità* individua una comunicazione caratterizzata dall'interazione di più linguaggi (immagini, testi scritti, animazioni, suoni). I «contenuti multimediali» esplicano la funzione di trasmettere informazioni, tramite molteplici mezzi di comunicazione: immagini, video, testi, musica.

15. La comunicazione dei dati viene definita, anche, trasmissione dei dati o comunicazione digitale.

Con l'avvento del XXI secolo, la multimedialità è divenuta una componente essenziale (all'interno dei servizi offerti da Internet). I contenuti multimediali rappresentano, all'interno delle nuove pagine web dinamiche, un elemento indispensabile: grazie all'utilizzo di immagini, video e suoni, il programmatore web è in grado di realizzare pagine web atte alla divulgazione di informazioni. La diffusione di contenuti multimediali audio-video è resa possibile dalla parallela diffusione di tecniche di compressione dati, sempre più efficienti, che riducono l'informazione ridondante permettendo l'abbassamento del bit-rate associato e, dunque, un minore spreco di banda utilizzata.

Questi elementi sono indispensabili per comprendere come sia possibile realizzare un'applicazione web, sia dal punto di vista strutturale, sia riguardo le informazioni che si vogliono trasmettere. L'applicazione web realizzata, a tal proposito, si propone di visualizzare, in tempo reale, l'andamento temporale di alcuni sensori presenti all'interno di un dispositivo android fissato su un robot. I dati raccolti, oltre ad essere visualizzati, verranno salvati in modo tale da poterne usufruire in futuro per ulteriori analisi.

4.1 Introduzione alle tecnologie Web Based

In informatica, lo sviluppo delle applicazioni riveste un ruolo fondamentale. Un'*applicazione* identifica un programma che può sia essere installato sul proprio personal computer che risiedere all'interno di un server ed essere accessibile da remoto. Fine ultimo delle applicazioni è quello di offrire delle funzionalità, servizi o strumenti selezionabili dall'utente tramite un'interfaccia utente. Spesso, l'utilizzatore è in grado di interagire con esse mediante strumenti di input.

Il software applicativo può essere realizzato in differenti modi, a seconda di come il programmatore decida rilasciarlo all'utente finale. Tra le varie tipologie di software applicativo è possibile distinguere:

- **Desktop**: definito software di office automation; in questa categoria rientrano i fogli di calcolo, le presentazioni, le elaborazioni di testi, la grafica, etc.;
- **Client-Server**: si tratta di applicazioni distribuite all'interno della rete dove un client è in grado di connettersi ad un server ed usufruire dei servizi ad esso associati;

- **Software Development**: sviluppo di applicazioni mediante l'utilizzo di editor di testo, definiti comunemente IDE [16];
- **Business** o **Enterprise**: questa macrocategoria ingloba le applicazioni «aziendali», suddivise a loro volta in software gestionali, e-commerce, produzione, marketing, personale, etc.

Le applicazioni web generalmente sfruttano l'architettura *client-server* [17]. Il client è un programma che richiede un servizio a un computer collegato in rete, mentre il server è un programma in grado di rispondere alle richieste di servizi. Un'applicazione web, definita anche *Web Based*, è un programma in cui tutte le funzionalità messe a disposizione sono accessibili da un semplice motore di ricerca (Google Chrome, Firefox, Internet Explorer, Opera, Safari, etc.). Per tale motivo, non è necessario installare alcun software all'interno del proprio personal computer, ma è possibile interagire con l'applicazione da qualsiasi sede, utilizzando qualsivoglia PC.

Il software applicativo, realizzato per il monitoraggio remoto del robot, sfrutta le caratteristiche messe a disposizione dalla programmazione web. I vari linguaggi di scripting consentono la realizzazione e lo sviluppo di applicazioni web. Talvolta questi sono realizzati in maniera statica, oppure, come sovente accade, sono sviluppati in modo dinamico.

L'interfaccia web è stata realizzata utilizzando lo standard **HTML5** per comporre la pagina web, sfruttando i recenti tag offerti dall'ultima versione. Lo stile e la formattazione usufruiscono dei principi del linguaggio **CSS3**, mentre le funzionalità che permettono l'interazione con il robot e l'applicazione android sono state progettate mediante il linguaggio di scripting **JavaScript**.

16. L'IDE (*Integrated Development Environment*, anche *Integrated Design Environment* o *Integrated Debugging Environment*) è un software che permette ai programmatori di sviluppare in maniera più semplice il codice sorgente di un programma. Esso permette di individuare errori relativi alla sintassi del codice in fase di scrittura, oltre a supportare una serie di funzionalità relative al debugging e allo sviluppo.
17. L'architettura *client-server* indica un sistema di rete dove, generalmente, un computer *client* si connette ad un *server* in modo tale da fruire di un certo servizio, utilizzando un'architettura protocollare sottostante. Il server ha il compito di gestire gli accessi alle risorse in modo tale da evitare conflitti di utilizzo da parte dei computer client. Cfr. https://it.wikipedia.org/wiki/Sistema_client/server

4.1.1 Linguaggi di programmazione Web Based

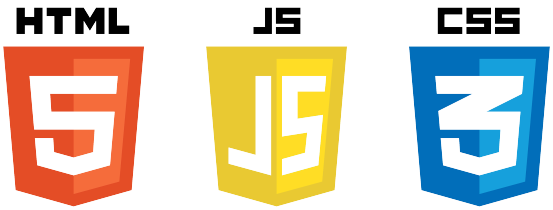


Figura 4.1: Linguaggi di programmazione Web Based

HyperText Markup Language è un linguaggio di markup utilizzato per la formattazione e impaginazione di siti web. Si tratta di un linguaggio pubblico, la cui sintassi è stata definita dal World Wide Web Consortium (W3C). La versione più recente prende il nome di *HTML5*, in cui sono state introdotte alcune funzionalità che, precedentemente, non erano ancora disponibili, ma bisognava fruirne tramite estensioni proprietarie esterne. Un altro vantaggio a favore della nuova versione, rispetto alla precedente, riguarda la compatibilità tra i diversi browser. Inoltre, vengono rimarcate l'indipendenza della piattaforma software utilizzata e la propensione all'utilizzo sui dispositivi mobili.

JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi. Viene comunemente utilizzato per la realizzazione di pagine web lato client, in modo tale da introdurre effetti dinamici tramite funzioni di script. Queste ultime vengono invocate tramite gli eventi innescati in vari modi dall'utente. Tali funzioni di script sono inserite all'interno di opportuni file HTML oppure, in appositi file separati e richiamate, all'occorrenza, all'interno delle pagine HTML. Sfruttando HTML5 e JavaScript, è possibile realizzare semplici applicazioni web di qualunque tipo: siti blog, e-commerce, pagine web personali, social media, etc.

Cascading Style Sheets, comunemente denominato CSS, è un linguaggio fortemente utilizzato per la formattazione dei documenti HTML e XML usati per la realizzazione di pagine e siti web. L'introduzione del CSS è stata utile per rendere più chiara e fluida la stesura di una pagina HTML. Ciò ha permesso di separare il codice dedicato allo stile di formattazione da quello strutturale proprio di HTML. La versione attualmente in corso di utilizzo è *CSS3*. A differenza della precedente versione, quest'ultima suddivide le varie specifiche in apposite sezioni chiamate *moduli*. HTML5 e CSS3 rappresentano, insieme, una componente completa che permette di realizzare pagine web

statiche. Insieme al linguaggio di scripting JavaScript, possono essere considerati i linguaggi di programmazione standard utili per la programmazione web.

4.1.2 Visual Studio Code

Il software **Visual Studio Code** è un IDE che permette la realizzazione di codice sorgente. Sviluppato da Microsoft, è disponibile nelle varie release per Windows, Linux e macOS. Mette a disposizione differenti supporti per il Debugging [18], il controllo per il Git [19], la Syntax highlighting, IntelliSense [20], Snippet e refactoring del codice.

Figura 4.2: Visual Studio Code

Una delle caratteristiche peculiari del suddetto editor di testo è il suo adattamento ai vari linguaggi di programmazione, tra cui: C, C++, C#, HTML, JAVA, JavaScript, CSS, etc. Pertanto, sono messe a disposizione varie estensioni, scaricabili direttamente dal repository ufficiale. Queste ultime permettono di adattare l'IDE al linguaggio di programmazione che viene utilizzato per la realizzazione del software applicativo. Sfruttando tali estensioni, il programmatore può scegliere quali di esse utilizzare, in modo tale da non appesantire l'editor di testo. A tale scopo, è possibile disabilitare quelle non necessarie ed, invece, abilitare quelle utili al software che si voglia realizzare.

18. Il codice di Debug viene visualizzato direttamente dall'editor Visual Studio Code. È possibile eseguire direttamente il debug segnando i vari punti di interruzione sull'IDE, analizzare lo stack di chiamate e sfruttare una console interattiva con il codice.
19. I comandi Git sono integrati all'interno dell'editor di testo. Questi permettono il controllo dei file contenuti all'interno del repository, l'esecuzione dei commit direttamente dall'editor, così come i push e i merge.
20. Visual Studio Code è in grado di mettere in evidenza la sintassi (specifica per ciascun linguaggio di programmazione) e il completamento automatico mediante l'IntelliSense, che fornisce i completamenti intelligenti basati sui vari tipi di variabili, le definizioni di funzioni e i vari moduli o estensioni importati.

4.1.3 Struttura dell'applicazione ROS Web Interface

L'applicazione web, adibita al monitoraggio da remoto del robot e alla gestione multimediale dei dati, è stata realizzata sfruttando le potenzialità dei linguaggi di programmazione web presentati precedentemente. L'interfaccia web prende il nome di **ROS Web Interface**. La ricerca condotta dal presente elaborato, culminata nella realizzazione dell'applicazione web e sintetizzata dalla stesura finale, ha sfruttato l'applicazione android **PIC4SeR Monitoring** al fine di acquisire informazioni sui sensori.

L'applicazione web è stata realizzata affinché sia possibile monitorare da remoto un robot. La presenza di appositi sensori all'interno del dispositivo android, inoltre, è stata utile per effettuare verifiche e analisi dei dati acquisiti. Sfruttando l'interfaccia web, è stato possibile monitorare, in tempo reale, l'andamento dei vari sensori e memorizzare i valori ricavati con lo scopo di rappresentarli graficamente e/o analizzarli successivamente.

L'applicativo sfrutta le caratteristiche presenti all'interno delle varie librerie JavaScript utilizzate. L'interazione con il framework ROS è stata resa possibile mediante l'utilizzo della libreria **roslibjs**, mentre i grafici bidimensionali sono stati realizzati mediante il supporto della libreria JavaScript **RGraph.js**. All'interno dell'applicazione web è stata inserita una mappa per localizzare il robot; tale mappa sfrutta le API messe a disposizione dalla libreria **Mapbox**.

L'interfaccia web è stata suddivisa opportunamente in correlazione ai molteplici sensori presenti all'interno del dispositivo android. Nel menu in testa alla pagina web, è possibile scegliere quale tipologia di sensori l'utente voglia analizzare. Tale menu è composto da quattro sezioni: la prima riguarda i sensori di movimento, a seguire vi sono i sensori relativi all'ambiente circostante, la terza mostra i sensori di rete ed, infine, sono presenti le informazioni riguardo la connessione LTE.

Una volta selezionato il dispositivo android collegato all'interfaccia web, è possibile attivare/disattivare i singoli attuatori dai quali ricavare i valori ed osservare l'andamento temporale. Tra i sensori di movimento, sono stati realizzati appositi grafici per l'accelerometro, il giroscopio e il vettore di rotazione. Riguardo i sensori d'ambiente, i dispositivi android sono in grado di trasmettere informazioni sull'illuminazione, i valori delle coordinate del

campo magnetico, i valori di Roll, Pitch e Azimuth [21] relativi alla bussola, le informazioni ricavate dal barometro e dal termometro. L'analisi della rete è stata effettuata sfruttando sia le informazioni ricavate dalla NIC presente all'interno del proprio personal computer, che quelle acquisite dal dispositivo android. Infine, sono stati analizzati i dati della cella telefonica a cui lo smartphone è collegato.

I dati acquisiti dallo smartphone hanno permesso di realizzare una panoramica dell'ambiente circostante al robot. Inoltre, mediante il supporto della teleoperazione messa a disposizione dal robot stesso e le informazioni relative alla geolocalizzazione inviate dal dispositivo android, è stato possibile monitorare il robot da remoto. Sfruttando opportune telecamere (una ubicata sul robot e l'altra messa a disposizione dal dispositivo android), è stato possibile visualizzare il tragitto e l'ambiente attorno al robot stesso.

4.2 Trasmissione e gestione delle immagini multimediali

L'interfaccia web, come già introdotto in precedenza, prevede una sezione dedicata alla trasmissione delle immagini multimediali riprese dal dispositivo android collegato al master ROS. Le immagini acquisite, inoltre, provengono anche da una videocamera montata a bordo del robot. La selezione di un tasto consente di scegliere quale camera utilizzare. Entrambe acquisiscono informazioni sull'ambiente circostante, le quali verranno trasmesse su dei canali HTTP. Il video real-time trasmesso, pertanto, è composto da una sequenza di frames: questi vengono visualizzati all'interno di un riquadro predisposto a tale scopo.

Affinché l'interfaccia web sia in grado di interagire con l'architettura ROS e con le sue proprietà, è necessario creare una Web Socket. Sfruttando le caratteristiche presenti nel nodo ROSBridge, è stato possibile realizzare tale struttura di comunicazione in JavaScript. Una volta stabilita la connessione ad uno specifico indirizzo IP (ad esempio 127.0.0.1) e ad una determinata porta (valore standard: 9090), è stato possibile sottoscrivere i topic e richia-

21. Roll, Pitch e Azimuth descrivono l'insieme delle rotazioni di un corpo rigido rispetto agli assi del sistema di riferimento cartesiano. Roll indica la rotazione di un angolo ψ attorno all'asse X; la rotazione attorno all'asse Y di un angolo θ è definita Pitch; Azimuth indica l'angolo formato dal piano verticale passante per un astro con il piano meridiano del luogo di osservazione.

mare i servizi realizzati appositamente per l'applicazione web e pubblicati dal dispositivo android.

Nella Figura 4.3 è descritto il metodo che permette di creare una Web Socket sfruttando la libreria *roslibjs*. Il parametro URL definito per la connessione è stato settato in modo tale che l'interfaccia web sia in grado di comunicare con il nodo *rosbridge_server*. Per questo motivo, il canale di comunicazione è stato creato impiegando i valori standard di indirizzo IP e porta utilizzati da ROSBridge (127.0.0.1 e 9090).

```
// Init handle for rosbridge_websocket
// This function connects to the rosbridge server running on the local computer on port 9090
ros = new ROSLIB.Ros({
    url: 'ws://' + robot_IP + ':9090'
});
```

Figura 4.3: Sorgente JavaScript per la creazione di una Web Socket in ROS

L'architettura dell'applicazione prevede due nodi relativi alla trasmissione delle immagini: uno avviato per la trasmissione di frames compressi ed un altro realizzato per i dati grezzi. Pertanto, all'interno dell'applicazione web, è previsto l'utilizzo di due URL che consentono di acquisire le opportune informazioni. Alle due URL utilizzate per l'acquisizione delle immagini di tipo *compressed* e di tipo *raw* è stata aggiunta un'ulteriore URL utilizzata per l'acquisizione delle immagini nel formato video *H264*. La scelta di utilizzare due modalità differenti di acquisizione delle immagini è servita, soprattutto, ad analizzare la qualità dei dati, nonché la latenza di trasmissione. Dalle varie misurazioni effettuate, difatti, sono state osservate diverse situazioni, traendo opportune conclusioni.

```
<div class="col-xs-10 col-sm-10 col-md-10 col-lg-10 col-xl-10">
  <div id="img-wrapper" class="border border-dark d-flex justify-content-center img-wrapper">
    <img src="" style="object-fit:contain; width: 250px; height: 190px;" id="video" />
  </div>
</div>
```

Figura 4.4: Sorgente HTML per la trasmissione di immagini

Dalla Figura 4.4 è possibile osservare come vengono visualizzati i frames acquisiti dalla videocamera, sia nel caso del dispositivo android, che nel caso della videocamera presente nel robot. Entrambi, infatti, utilizzano il tag HTML <img> dove il parametro *src* viene, di volta in volta, aggiornato sfruttando la funzione chiamata all'interno del file JavaScript. L'attributo *object-fit:contain*, inoltre, permette di ridimensionare il contenuto delle immagini all'interno del box di visualizzazione in modo tale da mantenere le proporzioni.

```
/**
 * SHOW RESOLUTION
 * This function is used to read the value of resolution.
 */
function showResolution(topic) {

    // Init topic object
    setShowResolution = new ROSLIB.Topic({
        ros: ros,
        name: topic,
        messageType: 'sensor_msgs/CameraInfo'
    });

    oldWidth = 0;
    oldHeigth = 0;
    setShowResolution.subscribe(function resolution(callback) {
        if (callback.width != oldWidth || callback.height != oldHeigth) {
            document.getElementById('valueResolution').innerHTML = callback.width + 'x' + callback.height;
            oldWidth = callback.width;
            oldHeigth = callback.height;
        }
        scaled_height = 250 * callback.width / callback.height;
```

Figura 4.5: Sorgente JavaScript per la trasmissione di immagini [1]

```
        if (document.getElementById('myswitchcamera').value == 'phone') {
            if (codec_video == 'jpeg') {
                // Populate video source with android camera (codec compressed)
                // In this case we are using the type ros_compressed (sequence of jpeg img) with the tag img html
                video.src = "http://" + robot_IP + ":7070/stream?topic=" + phoneName +
"/android/camera/image&width=250&height=" + Math.round(scaled_height) + "&type=ros_compressed";
            }
            if (codec_video == 'x264') {
                // Populate video source with android camera (codec x264)
                // In this case we are using the codec video x264 with the tag img html
                video.src = "http://" + robot_IP + ":7070/stream?topic=/republished/" + phoneName +
"/android/camera/image&width=" + Math.round(scaled_height) + "&height=250&type=ros_compressed";
            }
        } else if (document.getElementById('myswitchcamera').value == 'robot') {
            // Populate video source with Robot camera
            // In this case we are using the type raw (sequence of jpeg img) with the tag img html
            video.src = "http://" + robot_IP + ":8080/stream?topic=/camera/rgb/image_raw&width=200&height=" +
Math.round(scaled_height) + "&type=mjpeg&quality=100";
        }

        setShowResolution.unsubscribe();

    });
}
```

Figura 4.6: Sorgente JavaScript per la trasmissione di immagini [2]

Nelle Figure 4.5 e 4.6 è possibile osservare la funzione di script che permette di visualizzare le immagini trasmesse all'interno del box predisposto. Nel dettaglio di cui dispone, è possibile selezionare la camera presente sul robot oppure quella del dispositivo android.

Nel primo caso, vengono utilizzate due URL differenti in base al tipo di codifica scelto per la trasmissione delle informazioni. Per la trasmissione delle immagini nel formato *jpeg* si utilizza la porta 7070 ed il topic */phoneName/android/camera/image/compressed*. La variabile *phoneName* è sostituita dal nome scelto dall'utente quando viene avviata l'applicazione android. Questo viene anteposto a tutti i topic pubblicati dal dispositivo android. Affinché sia possibile trasmettere i frames compressi, è opportuno settare il topic escludendo l'ultima parola chiave (*/compressed*). Quest'ultima, pertanto, sarà inserita all'interno della variabile *type* messa a disposizione dal nodo *web_video_server* utilizzando la notazione *ros_compressed*. Per la trasmissione delle immagini nel formato *H264* si utilizza, invece, il topic */republished/phoneName/android/camera/image*. Anche in questo caso la variabile *phoneName* assume il valore associato al dispositivo android nel momento dell'esecuzione dell'applicazione **PIC4SeR Monitoring**. Il package *x264_video_transport* ha l'obiettivo di acquisire le immagini pubblicate dal dispositivo android nel formato video *H264* e successivamente ripubblicare lo stesso topic anteponendo la parola chiave */republished*.

Nel secondo caso, invece, il parametro *src* del tag HTML <img> è stato inizializzato utilizzando uno stream di dati che viaggia sulla porta 8080. Il topic utilizzato è quello fornito dalla camera stessa, ovvero: */camera/rgb/image_raw* a cui sono stati settati i valori di *width* e *height*, oltre al tipo di formato delle immagini: *mjpeg*.

L'applicazione android fornisce anche dei servizi utili che permettono, da remoto, la modifica di alcune impostazioni proprie del dispositivo android. In particolare, sono stati realizzati quattro servizi: due di questi permettono di ruotare l'immagine trasmessa dalla camera presente nello smartphone e scegliere tra la camera anteriore e posteriore; i restanti sono stati realizzati per modificare la compressione delle immagini trasmesse e la loro risoluzione. Nell'interfaccia web è stato possibile usufruire di taluni metodi messi a disposizione dalla libreria *roslibjs* in modo tale da richiamare questi servizi.

Anzitutto, è stata realizzata la funzione *rotateImage()* la quale permette di ruotare l'immagine in senso orario ogni qualvolta l'utente seleziona l'apposito tasto. Tale servizio risulta molto utile in quanto il dispositivo android

acquisisce le immagini in maniera standard. Ciò vuol dire che qualunque tipo di smartphone ha un suo orientamento di acquisizione. Una rotazione dell'immagine, pertanto, si rende necessaria qualora si voglia osservare in maniera corretta l'ambiente circostante al robot.

```
// Init service object
setRotateImage = new ROSLIB.Service({
    ros: ros,
    name: phoneName + '/camera/rotate_image',
    messageType: 'senspub_msgs/RotateImage'
});
```

Figura 4.7: Sorgente JavaScript per la rotazione della camera

Nella Figura 4.7, è stato possibile sfruttare il servizio offerto dall'applicazione android */phoneName/camera/rotate_image* per ruotare l'immagine trasmessa. È possibile impostare il valore da trasmettere al servizio mediante una *ServiceRequest* la quale, non appena richiamata la callback, risulta in grado di ruotare l'immagine del valore (in gradi) scelto.

Un ulteriore servizio è stato realizzato per scegliere, da remoto, la fotocamera presente sullo smartphone. Infatti, mediante il servizio */phoneName/camera/switch*, è possibile selezionare quale camera (anteriore o posteriore) del dispositivo android utilizzare.

```
// Init service object
setChangeCamera = new ROSLIB.Service({
    ros: ros,
    name: phoneName + '/camera/switch',
    messageType: 'senspub_msgs/SwitchCamera'
});
```

Figura 4.8: Sorgente JavaScript per lo switch della fotocamera android

Mediante l'utilizzo di tale servizio, come mostrato in Figura 4.8, è possibile scegliere da remoto quale fotocamera utilizzare per effettuare le riprese.

Qualora il robot assumesse una posizione scomoda per la ripresa delle immagini, è possibile scegliere una delle due fotocamere presenti sullo smartphone come alternativa, migliorando, così, la ripresa delle immagini.

Il metodo *modifyResolution()* permette di diminuire o aumentare la risoluzione delle immagini trasmesse. Questo servizio è utile per scegliere un'opportuna risoluzione delle immagini sulla base di quanta banda si decida di occupare. Di fatto, nel momento in cui non si ha la possibilità di disporre di una banda necessaria per la trasmissione di frames ad alta risoluzione, l'utente può diminuire quest'ultima evitando, in questo modo, di consumare banda utile [22].

```
// Init service object
setResolution = new ROSLIB.Service({
    ros: ros,
    name: phoneName + '/camera/set_resolution',
    messageType: 'senspub_msgs/SetImageResolution'
});
```

Figura 4.9: Sorgente JavaScript per la modifica della risoluzione delle immagini

È possibile osservare dalla Figura 4.9 come modificare la risoluzione delle immagini acquisite. Mediante l'utilizzo del servizio */phoneName/camera/-set_resolution*, è possibile modificare la risoluzione delle immagini. L'argomento della *ServiceRequest* può assumere i seguenti valori: +1 o -1. Ogni qualvolta venga utilizzato il valore +1, la nuova risoluzione impostata sarà la successiva più grande tra quelle a disposizione. Così, come nel servizio precedente, una volta settati i parametri e richiamata la callback, le nuove immagini trasmesse avranno la risoluzione desiderata.

Infine, è stato necessario usufruire del servizio dedicato alla compressione delle immagini trasmesse. Mediante un apposito range slider presente all'interno della pagina web, è possibile modificare il valore di compressione delle immagini acquisite. L'intervallo di possibili valori che può assumere il parametro *quality* è compreso tra 1 e 99, e dipende, fondamentalmente, dal

22. Maggiore è la risoluzione delle immagini trasmesse, maggiore sarà la banda occupata e minore sarà la velocità di trasmissione delle immagini.

valore di risoluzione utilizzato per l'acquisizione delle immagini. Così facendo, è possibile scegliere la qualità delle immagini trasmesse: se la banda a disposizione è piuttosto limitata, converrà utilizzare una qualità minore per non avere ritardi nella trasmissione dei frames.

```
// Init service object
setQuality = new ROSLIB.Service({
    ros: ros,
    name: phoneName + '/camera/set_quality',
    messageType: 'senspub_msgs/SetImageQuality'
});
```

Figura 4.10: Sorgente JavaScript per la modifica della compressione delle immagini

Sfruttando il metodo JavaScript mostrato in Figura 4.10, è possibile richiamare il servizio utilizzando come argomento il valore di compressione desiderato, compreso tra 1 e 99. Il valore utilizzato è ricavato dal range slider presente nell'interfaccia web. Quando l'utente decide di modificare tale valore, automaticamente viene inviata una richiesta al servizio attraverso cui sarà in grado di modificare la compressione delle immagini trasmesse.

I servizi descritti in precedenza sono utilizzabili esclusivamente per tale applicazione. Perciò, non è possibile sfruttarli quando si utilizza la camera presente nel robot. Sono disponibili, pertanto, sia per il codec video *H264* che *jpeg*.

4.3 Geolocalizzazione del robot mediante API Mapbox

Uno degli elementi fondamentali per monitorare un robot da remoto è certamente l'utilizzo di una mappa che permette la sua localizzazione. Viene definita *geolocalizzazione* l'identificazione della posizione geografica di un dato oggetto all'interno del globo reale. Nel contesto dell'applicazione web, la geolocalizzazione avviene sfruttando il dispositivo android collocato sul robot oppure il sensore di geolocalizzazione presente sul robot stesso. Esistono diverse tecnologie che permettono di localizzare un oggetto:

- **GPS**: è una tecnologia basata sui segnali radio. Tali informazioni vengono ricavate dai satelliti artificiali in orbita sulla Terra;
- Localizzazione tramite le **celle** della **rete telefonica cellulare**: mediante l'utilizzo della potenza del segnale radio delle celle telefoniche, è possibile analizzare e individuare a quale stazione radio base risulti collegato il dispositivo mobile, calcolandone la distanza in base all'attenuazione dell'ambiente di radiopropagazione;
- Rete **WIFI** o **WLAN**: informazioni derivanti da diverse reti WIFI, a loro volta localizzate tramite la rete Internet;
- Localizzazione in **loco**: utilizzo dei sistemi *ARVA* [23] e *RECCO* [24].

L'applicazione web prevede un'apposita sezione dedicata alla visualizzazione della mappa, in cui è possibile localizzare il robot. È possibile scegliere il sensore GPS integrato sul robot Clearpath Jackal oppure il sensore GPS del dispositivo android. Una volta effettuata la scelta, non è possibile modificarla, se non aggiornando l'interfaccia web. La scelta del ricevitore GPS deve essere effettuata prima di selezionare il dispositivo android, poiché questa verrà disabilitata una volta confermato lo smartphone da utilizzare. JavaScript mette a disposizione varie librerie per l'inserimento di una mappa all'interno di una pagina web. Nel progetto sviluppato, sono state utilizzate le API messe a disposizione dalla libreria **Mapbox**.

4.3.1 API Mapbox e sue funzionalità

Figura 4.11: Mapbox

Nel mondo delle applicazioni web sono presenti molteplici fornitori di mappe online personalizzabili in base alle proprie esigenze. Uno di questi è **Mapbox**. Questo mette a disposizione alcune librerie e applicazioni open source per rappresentare le mappe all'interno di un applicativo web o mobile. I dati vengono acquisiti da fonti aperte, come *OpenStreetMap*.

23. Apparecchio di Ricerca Valanga, più comunemente chiamato ARVA, è uno strumento elettronico che viene utilizzato per ricercare le persone travolte nelle valanghe. Si tratta di una ricetrasmittente di segnale funzionante sulla frequenza di 457 kHz.
24. Sistema elettronico per il ritrovo di persone sepolte nella neve a seguito di una valanga. Anch'esso è un trasmettitore/ricevitore che emette un segnale direzionale da un'antenna e, se il segnale colpisce un riflettore, viene ritrasmesso al rilevatore stesso.

È possibile scegliere tra diverse tipologie di mappe, quali: Mapbox Light, Dark, Streets, Outdoors, Satellite Streets, Navigation, etc. Il suddetto fornitore gestisce anche la navigazione e, pertanto, tutte le caratteristiche ad essa associate: il calcolo del percorso, il traffico, le indicazioni stradali, le tipologie di trasporto, etc. Inoltre, è possibile effettuare precise ricerche che permettono di individuare specifici luoghi. Posizionare la ricerca e la geocodifica risulta fondamentale per la realizzazione delle mappe, per la navigazione ed è alla base di ogni applicazione che permette agli utenti di esplorare il mondo.

Mapbox fornisce una serie di strumenti per creare mappe all'interno di un'applicazione web. **Mapbox GL JS** e **Mapbox.js** sono ambedue librerie JavaScript open source che permettono di visualizzare le mappe messe a disposizione online da Mapbox. Possono, altresì, essere aggiunti comandi affinché sia possibile l'interazione e la personalizzazione di questi strumenti. Inoltre, vengono forniti una serie di plugins che estendono le funzionalità della mappa: strumenti di disegno, interfacce alle API di alcuni servizi [25], etc.

Mapbox GL JS è una libreria JavaScript per le creazione di applicazioni web con la tecnologia di mappatura. Affinché sia possibile inserire all'interno dell'applicazione web la mappa, è necessario importare le opportune librerie JavaScript e CSS messe a disposizione online dal sito ufficiale di Mapbox.

```
map = new mapboxgl.Map({
    container: 'google_map',
    style: 'mapbox://styles/mapbox/streets-v10',
    center: [position.coords.longitude, position.coords.latitude],
    zoom: 15
});
```

Figura 4.12: Sorgente JavaScript per la creazione della mappa Mapbox

Nella Figura 4.12 è stata inizializzata la mappa all'interno della quale è stato localizzato il robot. *Mapbox GL JS* fornisce degli opportuni metodi che permettono di mostrare una mappa all'interno di un tag realizzato

25. Mapbox dispone di diverse API, tra cui: *API di Mapbox Geocoding* e *API Mapbox Directions*.

in HTML. Affinché la mappa venga visualizzata correttamente, è necessario settare alcuni parametri fondamentali.

I campi riportati all'interno della dichiarazione della mappa, nel suddetto caso, sono i seguenti:

- **Container**: si tratta dell'elemento HTML in cui è stata posizionata la mappa. L'elemento in questione ha valore id="google_map";
- **Style**: la mappa carica lo stile *mapbox://styles/mapbox/streets-v10*. Si tratta di una URL di un file remoto da cui è possibile scaricare i tileset e i vari plugins inclusi;
- **Center**: la mappa viene centrata sulle coordinate di longitudine e latitudine utilizzate come parametri. L'ordine dei parametri corrisponde all'ordine delle coordinate in GeoJSON;
- **Zoom**: livello di zoom in cui la mappa deve essere inizializzata. Il valore può essere modificato dall'utente nel momento in cui effettua lo zoom sulla mappa.

Una volta inizializzata la mappa, la geolocalizzazione del robot è determinata dalle informazioni provenienti dai topic pubblicati dall'applicazione **PIC4SeR Monitoring**, oppure da quelli del robot Clearpath Jackal. Questa applicazione acquisisce le informazioni sulla posizione corrente del robot. Dai valori raccolti, è stato possibile localizzare il robot e tracciare il percorso da esso compiuto.

4.3.2 ROS Topic e Message NavSatFix

La mappa inserita all'interno dell'interfaccia web permette di monitorare da remoto gli spostamenti effettuati dal robot. Di fatto, entrambi i sensori di geolocalizzazione sono in grado di raccogliere varie informazioni che consentono di individuare il robot sulla mappa. Tra quelle messe a disposizione dallo specifico messaggio, solamente i valori di latitudine e longitudine sono stati presi in considerazione per individuare il robot sulla mappa.

Il framework ROS mette a disposizione un package definito *sensor_msgs* che definisce i messaggi per i sensori più comuni, includendo anche quello relativo alla localizzazione. Tale messaggio prende il nome di **NavSatFix**.

Ambedue i sensori di geolocalizzazione usufruiscono del succitato messaggio, in quanto standardizzato dal framework ROS.

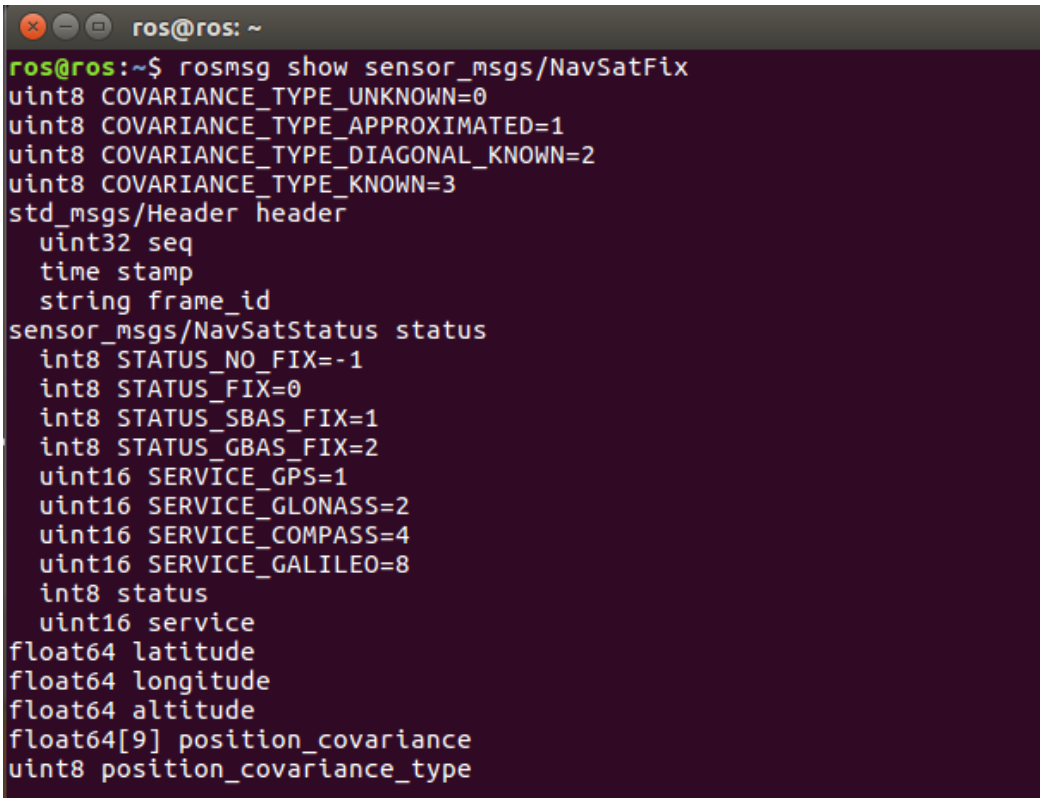


Figura 4.13: ROS Message *sensor_msgs/NavSatFix*

Nella Figura 4.13 è riportato il messaggio utilizzato per localizzare il robot all'interno della mappa. Tale messaggio raccoglie i dati acquisiti dal Sistema Globale di Navigazione Satellitare. È costituito dai seguenti campi:

- **Header header**: messaggio standard utilizzato per visualizzare le informazioni relative all'ora e alla tipologia del frame;
- **NavSatStatus status**: informazioni sullo stato del satellite;
- **float64 latitude**: valore della coordinata latitudine (in gradi);
- **float64 longitude**: valore della coordinata longitudine (in gradi);
- **float64 altitude**: valore della coordinata altitudine (in metri);

- **float64[9] position_covariance**: matrice contenente i valori della covarianza relativi alla posizione (in metri al quadrato);
- **uint8 position_covariance_type**:
  - se non è possibile ricavare il valore della covarianza, tale valore sarà 0 (COVARIANCE_TYPE_UNKNOWN);
  - se il valore della covarianza è approssimato, assumerà valore 1 (COVARIANCE_TYPE_APPROXIMATED);
  - se il valore della covarianza è fornito dal GPS, allora assumerà valore 2 (COVARIANCE_TYPE_DIAGONAL_KNOWN);
  - se il valore della covarianza è conosciuto, assumerà valore 3 (COVARIANCE_TYPE_KNOWN).

Mediante l'utilizzo del messaggio *sensor_msgs/NavSatFix* è stato possibile localizzare il robot all'interno della mappa. Sfruttando i valori di latitudine e longitudine, è stato possibile identificare la posizione corrente del robot mediante l'utilizzo di un marker. Quest'ultimo permette di visualizzare i movimenti effettuati dal robot stesso e consente di tracciare il percorso effettuato.

```
/**
 * READ GPS FROM ANDROID
 * This function is used to activate the geolocation from android.
 */
function readGPSfromAndroid(phoneName) {

    var topicGeolocation;

    if (gps == 'android') {
        topicGeolocation = phoneName + '/android/location/fix';
    } else {
        topicGeolocation = '/navsat/fix';
    }

    // Init topic object
    geolocation = new ROSLIB.Topic({
        ros: ros,
        name: topicGeolocation,
        messageType: 'sensor_msgs/NavSatFix'
    });
```

Figura 4.14: Sorgente JavaScript per l'inizializzazione dei topic */phoneName/android/location/fix* e */navsat/fix*

I topic utilizzati per ricavare le informazioni relative alle coordinate di geolocalizzazione sono quelli illustrati nella Figura 4.14. Una volta sottoscritti, permettono di ricavare i dati relativi alle coordinate di geolocalizzazione. Il marker, pertanto, sfrutta proprio i valori ricavati dai suddetti topic affinché sia possibile inserirlo all'interno della mappa. Questi topic vengono utilizzati in maniera esclusiva. Per tale motivo, l'utente deve, anzitutto, scegliere quale sensore di geolocalizzazione utilizzare. Le possibili scelte sono:

- spuntare la casella presente sotto la voce *Robot* nel caso in cui si voglia utilizzare il sensore presente nel robot;
- non spuntare tale casella significa utilizzare il topic pubblicato dal dispositivo android successivamente selezionato.

La sezione relativa alla visualizzazione della mappa prevede, inoltre, una tabella in cui sono riportati i valori di latitudine e longitudine acquisiti ed un valore di velocità (ricavato dalle coordinate e dal tempo trascorso). È presente, inoltre, un tasto che permette, una volta premuto, il celamento del tragitto effettuato dal robot [26]. Pur tuttavia, ogni qualvolta è ricaricata la pagina web, la storia delle posizioni e, quindi, l'intero percorso risulta definitivamente perso. A queste informazioni si aggiunge, come già detto in precedenza, la possibilità di scegliere la sorgente da cui ricavare le informazioni di localizzazione (robot o dispositivo android).

Infine, è stato realizzato un cerchio di imprecisione della posizione. Sfruttando le ultime posizioni acquisite e le proprietà offerte da Mapbox, il cerchio di imprecisione è stato utilizzato per individuare l'errore di precisione relativo alla posizione del robot stesso. La possibilità che il robot si trovi all'interno del cerchio di imprecisione è alta. I valori utili per poter realizzare tale cerchio sono stati ricavati sfruttando la definizione di *Deviazione Standard*. Tale strumento effettua la media ponderata degli scarti dalla media aritmetica elevati al quadrato, ovvero di quanto ogni valore si allontana dalla media aritmetica dei valori considerati. Tali valori sono stati ricavati a partire dalle ultime posizioni acquisite dal sensore e aggiunti, di volta in volta, ai valori correnti di latitudine e longitudine. Così facendo, è stato possibile raffigurare un cerchio di imprecisione, il quale è in grado di mostrare l'area di possibile ubicazione del robot.

26. Ciò avviene tramite il richiamo di una funzione script, la quale viene invocata alla selezione del suddetto tasto.

4.4 Teleoperazione del robot

L'applicazione web prevede la teleoperazione da remoto del robot connesso. Una volta avviato il robot, esso mette a disposizione i suoi topic e servizi al ROS master. La comunicazione avviene mediante un'opportuna Web Socket e gestita dal nodo ROSBridge.

Affinché il robot possa essere comandato a distanza, è necessario utilizzare il topic pubblicato dal robot in questione. In questo progetto di tesi è stato utilizzato un Clearpath Jackal, il quale mette a disposizione il topic */jackal_velocity_controller/cmd_vel*. Si tratta di un messaggio di tipo *geometry_msgs/Twist*.

```
/**
 * INIT VELOCITY PUBLISHER
 * This function create velocity message publisher.
 */
function initVelocityPublisher() {

    // Init message with zero values
    twist = new ROSLIB.Message({
        linear: {
            x: 0,
            y: 0,
            z: 0
        },
        angular: {
            x: 0,
            y: 0,
            z: 0
        }
    });

    // Init topic object
    cmdVel = new ROSLIB.Topic({
        ros: ros,
        name: '/jackal_velocity_controller/cmd_vel',
        messageType: 'geometry_msgs/Twist'
    });

    // Register publisher within ROS system
    cmdVel.advertise();
}
```

Figura 4.15: Sorgente JavaScript per l'inizializzazione del topic */jackal_velocity_controller/cmd_vel*

In JavaScript, è possibile configurare il topic suddetto e il relativo messaggio come mostrato nella Figura 4.15. I valori scelti per teleloperare il robot sono quelli definiti dalle coordinate spaziali, sia per eseguire movimenti lineari che angolari. Settando opportunamente i campi contenuti all'interno del messaggio, è possibile orientare il robot all'interno dell'ambiente circostante. Il metodo *advertise()*, inoltre, ha il compito di registrare il topic al nodo master, di modo che possa essere utilizzato correttamente. I comandi vengono impartiti usufruendo di un apposito joystick realizzato sfruttando la libreria JavaScript **Nipplejs**.

JavaScript offre una libreria in grado di teleoperare da tastiera il robot, senza l'ausilio del joystick presentato in precedenza. Tale libreria è chiamata **Keyboardteleopjs**.

```
/**
 * INIT TELEOP KEYBOARD
 * This function create a keyboard controller object.
 */
function initTeleopKeyboard() {
    // Use w, s, a, d keys to drive your robot

    // Initialize the teleop
    teleop = new KEYBOARDTELEOP.Teleop({
        ros: ros,
        topic: '/jackal_velocity_controller/cmd_vel'
    });
}
```

Figura 4.16: Sorgente JavaScript per l'inizializzazione del topic */jackal_velocity_controller/cmd_vel* mediante la libreria *keyboardteleopjs*

Nella Figura 4.16 viene inizializzata la teleoperazione mediante la tastiera del PC utilizzando la libreria *keyboardteleopjs*. In particolare, la classe contenuta all'interno della libreria JavaScript prevede due parametri: il primo individua la Web Socket utilizzata per la connessione; il secondo permette di inserire il topic relativo al comando a distanza (nel suddetto caso */jackal_velocity_controller/cmd_vel*). Una volta inizializzata tale funzione, è possibile teleoperare il robot sfruttando i comandi WASD. La libreria effettua automaticamente il porting del topic pubblicato ai comandi WASD, senza dover assegnare manualmente i valori di velocità lineare e angolare ai codici ASCII della tastiera.

4.5 Sensori di movimento

L'applicazione web realizzata per la gestione dei dati multimediali prevede una sezione dedicata interamente ai sensori di movimento. Le informazioni ricavate da tali sensori sono state rappresentate mediante l'ausilio di grafici bidimensionali e di una tabella contenente i valori acquisiti. I sensori di movimento realizzati sono: **accelerometro**, **giroscopio** e **vettore di rotazione**.

4.5.1 ROS Topic e Message Imu

I sensori di movimento presenti all'interno del dispositivo android acquisiscono specifici dati che vengono pubblicati all'interno di un messaggio standard definito dal framework ROS: *sensor_msgs/Imu*.

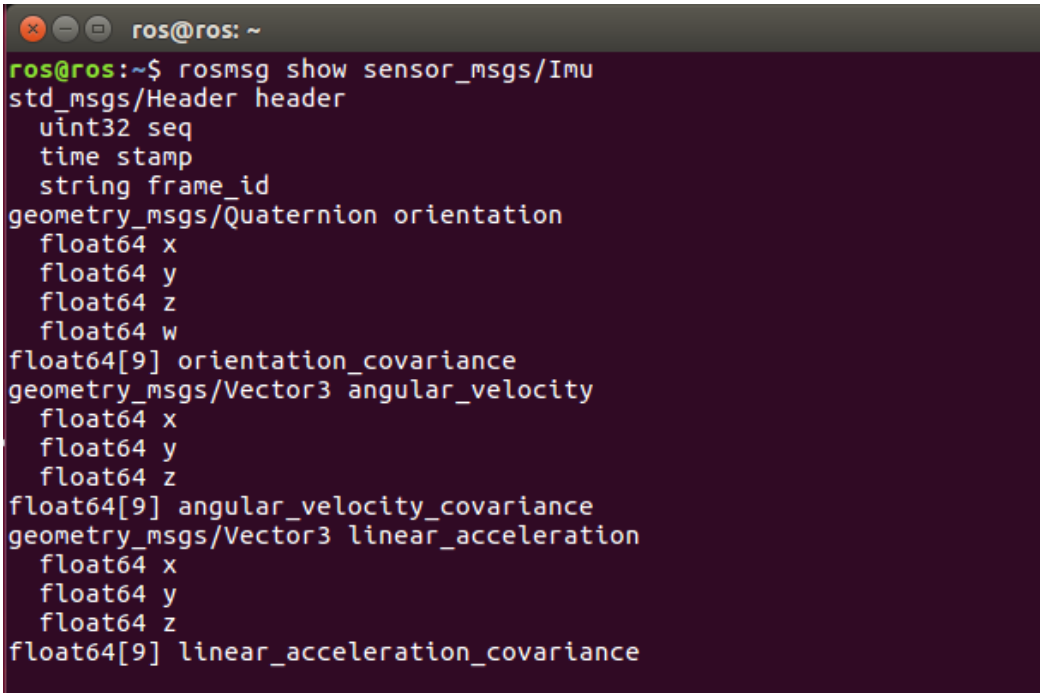


Figura 4.17: ROS Message *sensor_msgs/Imu*

Il messaggio visualizzato nella Figura 4.17 è stato utilizzato preminentemente per la realizzazione di grafici relativi ai sensori di movimento. Tali informazioni sono raggruppate all'interno del messaggio denominato IMU

(*Inertial Measurement Unit*). I campi presenti all'interno del suddetto messaggio ROS sono i seguenti:

- **Header header**: messaggio standard utilizzato per visualizzare le informazioni relative all'ora e alla tipologia del frame;
- **geometry_msgs/Quaternion orientation**: si tratta di un quaternione contenente i quattro valori associati al vettore di rotazione (lungo gli assi X, Y, Z e W);
- **geometry_msgs/Vector3 angular_velocity**: la velocità angolare è rappresentata da un vettore tridimensionale contenente le componenti utilizzate per il giroscopio;
- **geometry_msgs/Vector3 linear_acceleration**: l'accelerazione lineare è ricavata sulla base delle tre coordinate spaziali ed è utilizzata per il sensore accelerometro.

4.5.2 Accelerometro, Giroscopio e Vettore di rotazione

I sensori di movimento, come esposto in precedenza, sfruttano il medesimo messaggio Imu per pubblicare le loro informazioni al master ROS. Tuttavia, è opportuno sottoscrivere correttamente le informazioni pubblicate dal topic (tre informazioni diverse), in modo tale da non sbagliare, erroneamente, i valori da visualizzare nei grafici.

```
/**
 * READ ACCELEROMETER
 * This function read the sensor from android about accelerometer.
 */
function readAccelerometer(phoneName) {

    // Init topic object
    setAccelerometer = new ROSLIB.Topic({
        ros: ros,
        name: phoneName + '/android/imu',
        messageType: 'sensor_msgs/Imu'
    });

    oldTimestamp_acc = 0;
}
```

Figura 4.18: Sorgente JavaScript per l'inizializzazione del topic */phoneName/android/imu* riguardo l'accelerometro

Per ricavare le coordinate dell'accelerometro, dal topic mostrato nella Figura 4.18, bisogna acquisire i valori dell'accelerazione lineare. Una volta sottoscritto il suddetto topic, la *callback* pubblica varie informazioni. È opportuno memorizzare solamente quelle contenute all'interno del parametro *linear_acceleration*, il quale, a sua volta, è composto dalle tre componenti lungo gli assi cartesiani: X, Y e Z. Tali valori permettono di osservare l'accelerazione assunta dal robot quando esso è in movimento. Dovuta attenzione bisogna prestare ai valori assunti dalle tre coordinate cartesiane. Questi ultimi, difatti, dipendono dal posizionamento dello smartphone sul robot. Per tale motivo, variano in base al collocamento del dispositivo android. Inoltre, è opportuno osservare la presenza dell'accelerazione di gravità, la quale si aggiunge ad una delle tre coordinate cartesiane in base all'orientamento dello smartphone.

```
/**
 * READ GYROSCOPE
 * This function read the sensor from android about gyroscope.
 */
function readGyroscope(phoneName) {

    // Init topic object
    setGyroscope = new ROSLIB.Topic({
        ros: ros,
        name: phoneName + '/android/imu',
        messageType: 'sensor_msgs/Imu'
    });

    oldTimestamp_gyr = 0;
}
```

Figura 4.19: Sorgente JavaScript per l'inizializzazione del topic */phoneName/android/imu* riguardo il giroscopio

Nella Figura 4.19 è stato inizializzato il topic per ricavare le informazioni necessarie alla realizzazione del grafico relativo al giroscopio. Anche il giroscopio, così come l'accelerometro, prevede tre componenti lungo gli assi cartesiani. Il parametro da utilizzare all'interno del messaggio Imu, in questo caso, è *angular_velocity*. Una volta sottoscritto il topic, è possibile ricavare le informazioni sulla velocità angolare e sfruttare tali dati per osservare come il dispositivo android ruoti attorno ai tre assi di riferimento.

```javascript
/**
 * READ ORIENTATION
 * This function read the sensor from android about orientation.
 */
function readOrientation(phoneName) {

    // Init topic object
    setRotationVector = new ROSLIB.Topic({
        ros: ros,
        name: phoneName + '/android/imu',
        messageType: 'sensor_msgs/Imu'
    });

    oldTimestamp_rot = 0;
}
```

Figura 4.20: Sorgente JavaScript per l'inizializzazione del topic */phoneName/android/imu* riguardo il vettore di rotazione

Il vettore di rotazione è definito dalle componenti della variabile *orientation* ricavate dal topic in Figura 4.20. Tali valori, a differenza di quelli acquisiti per l'accelerometro e per il giroscopio, sono quattro. Ai normali assi di rotazione X, Y e Z, bisogna aggiungere la componente W, rappresentata anch'essa all'interno del grafico bidimensionale (così come le altre componenti).

Gli strumenti citati risultano utili per comprendere i movimenti effettuati dal robot. Nella fattispecie, è possibile osservare: la velocità lineare assunta dal robot in movimento; la rotazione attorno all'asse perpendicolare al terreno; le modalità con cui il dispositivo evita gli ostacoli; etc. Pertanto, l'utilizzo di tali strumenti è fondamentale per monitorare un robot a distanza, specie in spazi poco noti. Per valutare al meglio le condizioni del percorso effettuato dal dispositivo mobile, non è sufficiente solamente osservare le immagini acquisite dal robot tramite una o più videocamere. Pertanto, i dati acquisiti dai sensori di movimento permettono di comprendere, in maniera più analitica, i singoli movimenti compiuti dal robot lungo il tragitto effettuato.

4.6 Sensori d'ambiente

L'applicazione web prevede una sezione dedicata interamente ai sensori d'ambiente. Le misurazioni ricavate sono mostrate in tempo reale mediante specifici grafici bidimensionali. I sensori d'ambiente utilizzati nell'interfaccia web sono i seguenti: **illuminanza**, **magnetometro**, **bussola**, **termometro** e **barometro**.

4.6.1 ROS Topic e Message Illuminance

Il sensore illuminanza, presente all'interno del dispositivo android, rileva l'illuminamento fotometrico misurato lungo l'asse X del sensore (l'area di rilevamento è il piano Y-Z). L'illuminamento fotometrico assume valore 0 quando non è stato rilevato alcun valore, altrimenti registra un valore positivo: quest'ultimo dipende, fondamentalmente, dalla sorgente luminosa rilevata. Tale sensore, per l'esattezza, pubblica la misura della sensibilità dell'occhio umano quando la luce incontra o attraversa una superficie. ROS ha definito tale messaggio *sensor_msgs/Illuminance*.

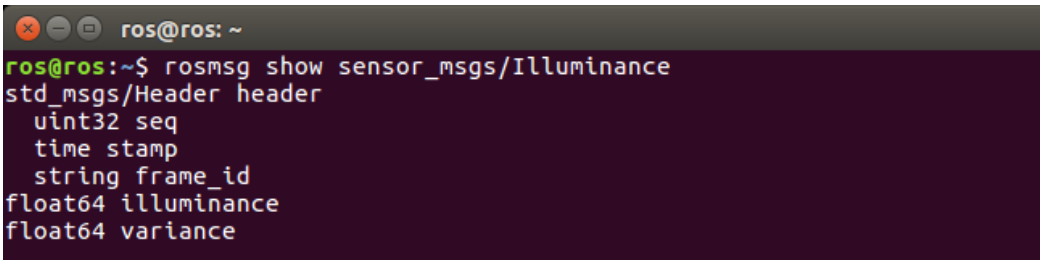


Figura 4.21: ROS Message *sensor_msgs/Illuminance*

Il messaggio Illuminance utilizzato per sottoscrivere i dati acquisiti dal dispositivo android è descritto nella Figura 4.21. Tale messaggio comprende i seguenti campi:

- **Header header**: messaggio standard utilizzato per visualizzare le informazioni relative all'ora e alla tipologia del frame;
- **float64 illuminance**: misura relativa all'illuminamento fotometrico (unità di misura Lux).

Tale messaggio è stato utilizzato all'interno del codice sorgente JavaScript in modo tale da poter acquisire il valore pubblicato dal dispositivo android.

```javascript
/**
 * READ PHOTOMETRIC ILLUMINANCE
 * This function read the sensor from android about illuminance.
 */
function readPhotometricIlluminance(phoneName) {

    // Init topic object
    setIlluminance = new ROSLIB.Topic({
        ros: ros,
        name: phoneName + '/android/illuminance',
        messageType: 'sensor_msgs/Illuminance'
    });

    oldTimestamp_ill = 0;
}
```

Figura 4.22: Sorgente JavaScript per l'inizializzazione del topic */phoneName/android/illuminance*

Il topic inizializzato nella Figura 4.22 permette di ricavare il valore dell'illuminamento fotometrico visualizzato, a sua volta, all'interno del grafico presente nell'interfaccia web. Le informazioni vengono acquisite dal sensore presente nel dispositivo android, solitamente posto di fianco all'altoparlante vocale. Memorizzando i dati relativi all'illuminanza fotometrica, è possibile osservare l'andamento temporale dell'intensità luminosa. Ciò permette di analizzare quali luoghi, visionati dal robot, risultino più o meno esposti alla luce del sole, così da poter ricavare opportune considerazioni future (agricoltura, analisi termiche, infrastrutture critiche, etc.).

4.6.2 ROS Topic e Message MagneticField

L'applicazione web prevede due sensori d'ambiente che utilizzano il medesimo messaggio ROS. Tuttavia, i dati acquisiti sono differenti. Il messaggio utilizzato prevede un campo contenente tre componenti. Questi sensori d'ambiente sono il *magnetometro* e la *bussola*. Entrambi utilizzano il messaggio ROS *sensor_msgs/MagneticField*. Nel primo caso, vengono visualizzate le informazioni relative al campo magnetico, acquisendo gli opportuni valori delle componenti lungo gli assi cartesiani X, Y e Z. Nel secondo caso, invece, i valori memorizzati all'interno delle componenti X e Y corrispondono, rispettivamente, ai valori di Roll e Pitch. La componente Z, invece, non descrive

esattamente il valore Azimuth acquisito; nonostante ciò, è stata forzata tale associazione evitando la creazione di un nuovo messaggio ad hoc.

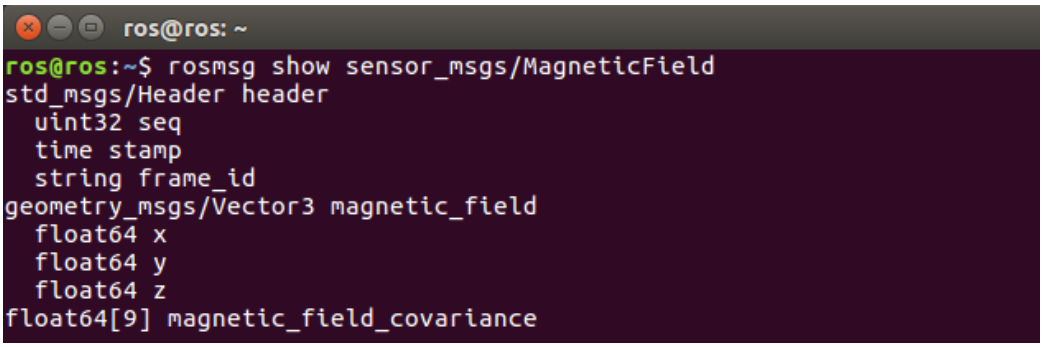


Figura 4.23: ROS Message *sensor_msgs/MagneticField*

Il messaggio ROS mostrato nella Figura 4.23 è composto dai seguenti campi:

- **Header header**: messaggio standard utilizzato per visualizzare le informazioni relative all'ora e alla tipologia del frame;
- **geometry_msgs/Vector3 magnetic_field**: tale campo include le componenti X, Y e Z del campo magnetico. L'unità di misura utilizzata nel suddetto messaggio è Tesla. Quando il sensore non è presente all'interno del dispositivo android, le tre componenti assumono valore NaN (*Not A Number*).

È stato utilizzato all'interno del codice JavaScript in ambedue le occasioni. Di fatto, entrambi i topic, uno dedicato al magnetometro e l'altro alla bussola, vengono sottoscritti allo stesso modo, tenendo conto delle medesime componenti. L'unica differenza è proprio il nome del topic. Nel caso del magnetometro, questo assume il valore di */phoneName/android/magnetic_field*; invece, per quanto riguarda la bussola, il nome del topic è */phoneName/android/compass*. Per compattezza viene riportato nella Figura 4.24 solamente il caso del magnetometro. I dati ricavati, ovvero le componenti lungo gli assi cartesiani, sono stati rappresentati all'interno di un grafico bidimensionale e visualizzati all'interno di una tabella. Le medesime azioni sono state compiute per ricavare i valori di Roll, Pitch e Azimuth, ovvero le tre componenti utili per calcolare il Nord Magnetico.

```javascript
/**
 * READ MAGNETIC FIELD
 * This function read the sensor from android about magnetic_field.
 */
function readMagneticField(phoneName) {

    // Init topic object
    setMagneticField = new ROSLIB.Topic({
        ros: ros,
        name: phoneName + '/android/magnetic_field',
        messageType: 'sensor_msgs/MagneticField'
    });

    oldTimestamp_magn = 0;
}
```

Figura 4.24: Sorgente JavaScript per l'inizializzazione del topic */phoneName/android/magnetic_field*

4.6.3 ROS Topic e Message Temperature e FluidPressure

Il dispositivo android fissato sul robot acquisisce ulteriori informazioni relative alla temperatura esterna e alla pressione atmosferica. Il framework ROS mette a disposizione due messaggi specifici che permettono di memorizzare le informazioni opportunamente. Tali messaggi prendono il nome di *sensor_msgs/Temperature* e *sensor_msgs/FluidPressure*.

```
ros@ros: ~
ros@ros:~$ rosmsg show sensor_msgs/Temperature
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
float64 temperature
float64 variance
```

Figura 4.25: ROS Message *sensor_msgs/Temperature*

Il messaggio ROS relativo al sensore della temperatura è descritto nella Figura 4.25. Tale messaggio è utilizzato per la lettura del valore di temperatura. Quest'ultimo è composto dai seguenti campi:

- **Header header**: messaggio standard utilizzato per visualizzare le informazioni relative all'ora e alla tipologia del frame;
- **float64 temperature**: raccoglie le informazioni sulla temperatura rilevata in gradi centigradi Celsius;
- **float64 variance**: assume valore 0 quando il valore di varianza è sconosciuto.

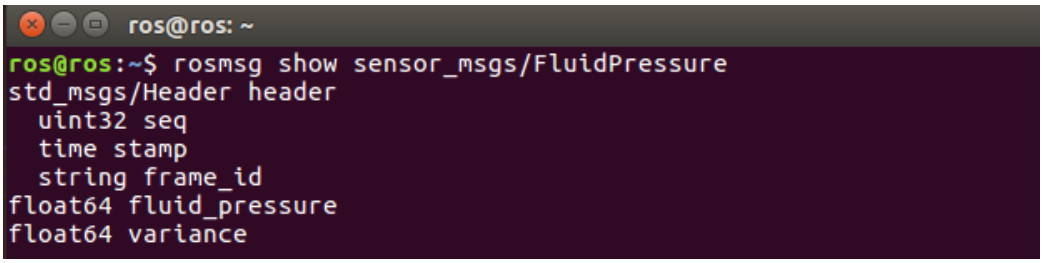


Figura 4.26: ROS Message *sensor_msgs/FluidPressure*

Nella Figura 4.26, invece, è descritto il messaggio ROS relativo all'acquisizione delle informazioni sulla pressione atmosferica. Mediante il suddetto messaggio, viene effettuata la lettura della pressione atmosferica o barometrica. I campi all'interno del messaggio succitato sono i seguenti:

- **Header header**: messaggio standard utilizzato per visualizzare le informazioni relative all'ora e alla tipologia del frame;
- **float64 fluid_pressure**: raccoglie le informazioni sulla pressione atmosferica misurata in Pascal;
- **float64 variance**: assume valore 0 quando il valore di varianza è sconosciuto.

Entrambi i valori acquisiti sono stati sottoscritti sfruttando la libreria *roslibjs*. I dati ricavati sono stati rappresentati in grafici bidimensionali e raccolti all'interno di una tabella. I valori di temperatura sono stati raccolti in gradi Celsius, mentre quelli di pressione atmosferica sono stati memorizzati nell'unità di misura Atmosfere, convertendo opportunamente il dato acquisito in Pascal.

```
/**
 * READ TEMPERATURE
 * This function read the sensor from android about temperature.
 */
function readTemperature(phoneName) {

    // Init topic object
    setTemperature = new ROSLIB.Topic({
        ros: ros,
        name: phoneName + '/android/temperature',
        messageType: 'sensor_msgs/Temperature'
    });

    oldTimestamp_temp = 0;
}
```

Figura 4.27: Sorgente JavaScript per l'inizializzazione del topic */phoneName/android/temperature*

```
/**
 * READ PRESSURE
 * This function read the sensor from android about pressure.
 */
function readPressure(phoneName) {

    // Init topic object
    setPressure = new ROSLIB.Topic({
        ros: ros,
        name: phoneName + '/android/barometric_pressure',
        messageType: 'sensor_msgs/FluidPressure'
    });

    oldTimestamp_pres = 0;
}
```

Figura 4.28: Sorgente JavaScript per l'inizializzazione del topic */phoneName/android/barometric_pressure*

Nelle Figure 4.27 e 4.28 sono stati ricavati i valori acquisiti dai suddetti sensori. Entrambi sono stati sottoscritti sfruttando il metodo *subscribe()*. Quando i succitati sensori sono attivati, vengono sottoscritti i topic pubblicati

dall'applicazione android. Le uniche informazioni utili da ambedue le callback sono i valori di temperatura e pressione: nel primo caso, il valore della temperatura è contenuto all'interno della variabile *temperature*; nel secondo caso, invece, la misura della pressione è data dal parametro *fluid_pressure*. Nel momento in cui i sensori vengono spenti, viene richiamato il metodo *unsubscribe()*, il quale è in grado di sospendere la sottoscrizione dei topic.

4.7 Sensori di rete

L'interfaccia web è composta da un'ulteriore sezione dedicata, in questo caso, ai sensori di rete. Sia la scheda di rete presente all'interno del personal computer che quella all'interno del dispositivo android sono state utilizzate per acquisire informazioni utili per l'analisi della rete wireless. I parametri, rappresentati mediante grafici bidimensionali all'interno dell'applicazione web, sono elencati di seguito:

- **RSSI**: *Received Signal Strength Indicator* rappresenta la misura della potenza presente in un segnale radio ricevuto [27];
- **Link Quality**: indica la qualità del segnale, determinata misurando e valutando i parametri di collegamento, come il Bit Error Rate (BER) e il livello tra segnale radio e distorsione di quest'ultimo (SINAD) [28];
- **Link Speed**: la velocità di trasmissione indica la quantità di dati che può essere trasferita all'interno di una canale di comunicazione in un dato intervallo di tempo [29].

4.7.1 ROS Topic e Message NetworkSensor

L'applicazione web è stata realizzata in modo tale da poter monitorare la rete wireless. Il robot, il PC (su cui è avviata l'interfaccia web) e il dispositivo android sono tutti connessi alla stessa rete locale. In tal modo, è possibile analizzare: la qualità della rete, la potenza del segnale e la quantità di banda occupata in download e upload. Il framework ROS non prevede alcun tipo di messaggio per il network monitoring; pertanto, è stato realizzato uno specifico messaggio contenente i parametri necessari a tale scopo. Il messaggio è stato generato con l'esecuzione del package *ros-network-analisys* illustrato nel precedente capitolo.

27. Cfr. https://en.wikipedia.org/wiki/Received_signal_strength_indication
28. Cfr. https://en.wikipedia.org/wiki/Link_quality_analysis
29. Cfr. https://it.wikipedia.org/wiki/Velocità_di_trasmissione

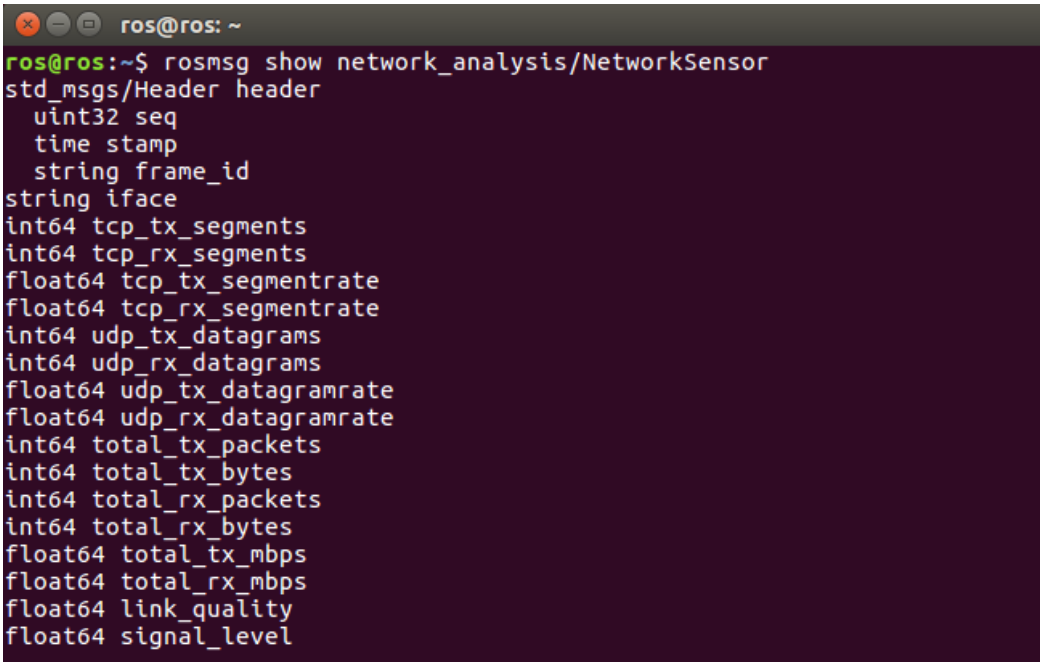


Figura 4.29: ROS Message *network_analysis/NetworkSensor*

Il messaggio, utilizzato per acquisire le informazioni dalla scheda di rete all'interno del PC, è visualizzato nella Figura 4.29. Quest'ultimo è composto dai seguenti campi:

- **Header header**: messaggio standard utilizzato per visualizzare le informazioni relative all'ora e alla tipologia del frame;
- **int64 tcp_tx_segments**: quantità di segmenti TCP trasmessi;
- **int64 tcp_rx_segments**: quantità di segmenti TCP ricevuti;
- **float64 tcp_tx_segmentrate**: differenza tra i segmenti TCP trasmessi e quelli inviati nel precedente lasso di tempo;
- **float64 tcp_rx_segmentrate**: differenza tra i segmenti TCP ricevuti e quelli ottenuti nell'intervallo di tempo antecedente;
- **int64 udp_tx_datagrams**: quantità di datagrammi UDP trasmessi;
- **int64 udp_rx_datagrams**: quantità di datagrammi UDP ricevuti;
- **float64 udp_tx_datagramrate**: differenza tra i datagrammi UDP trasmessi e quelli inviati antecedentemente;

- **float64 udp_rx_datagramrate**: differenza tra i datagrammi UDP ricevuti e quelli ottenuti nell'istante di tempo precedente;
- **int64 total_tx_packets**: quantità di pacchetti IP trasmessi;
- **int64 total_tx_bytes**: quantità di bytes trasmessi;
- **int64 total_rx_packets**: quantità di pacchetti IP ricevuti;
- **int64 total_rx_bytes**: quantità di bytes ricevuti;
- **float64 total_tx_mbps**: valore totale di Mbps relativo ai dati trasmessi;
- **float64 total_rx_mbps**: valore totale di Mbps relativo ai dati ricevuti;
- **float64 link_quality**: indicatore della qualità del link;
- **float64 signal_level**: indicatore del livello del segnale.

I parametri appena descritti assumono i rispettivi valori mediante l'utilizzo del topic */network_analysis/network_sensor*. Questo è stato generato da un apposito script realizzato in Python (come già descritto nel capitolo precedente). Tale topic permette di monitorare il traffico in entrata ed uscita dalla scheda di rete NIC. Sono stati, inoltre, realizzati due grafici atti a monitorare sia la potenza del segnale RSSI che la qualità della connessione in tempo reale.

```
/**
 * READ NIC BANDWITH
 * This function read the sensor from NIC bandwith.
 */
function readNICBandwith() {

    // Init topic object
    setNICBandwith = new ROSLIB.Topic({
        ros: ros,
        name: '/network_analysis/network_sensor',
        messageType: 'network_analysis/NetworkSensor'
    });

    oldTimestamp_nic_bandwith = 0;
}
```

Figura 4.30: Sorgente JavaScript per l'inizializzazione del topic */network_analysis/network_sensor* riguardo la banda in download e upload

```
/**
 * READ NIC RSSI
 * This function read the sensor from NIC RSSI.
 */
function readNICRSSI() {

    // Init topic object
    setNICRSSI = new ROSLIB.Topic({
        ros: ros,
        name: '/network_analysis/network_sensor',
        messageType: 'network_analysis/NetworkSensor'
    });

    oldTimestamp_nic_rssi = 0;
}
```

Figura 4.31: Sorgente JavaScript per l'inizializzazione del topic */network_analysis/network_sensor* riguardo la potenza del segnale

```
/**
 * READ NIC LINK QUALITY
 * This function read the sensor from NIC link quality.
 */
function readNICLinkQuality() {

    // Init topic object
    setNICLinkQuality = new ROSLIB.Topic({
        ros: ros,
        name: '/network_analysis/network_sensor',
        messageType: 'network_analysis/NetworkSensor'
    });

    oldTimestamp_nic_linkquality = 0;
}
```

Figura 4.32: Sorgente JavaScript per l'inizializzazione del topic */network_analysis/network_sensor* riguardo la qualità del segnale

Nelle Figure 4.30, 4.31 e 4.32 sono stati illustrati i metodi che permettono di inizializzare il topic deputato all'acquisizione delle informazioni sopracitate. Quando l'utente avvia il sensore specifico, il topic stesso può essere sottoscritto. I valori acquisiti vengono visualizzati all'interno dei grafici e nelle tabelle poste sotto ciascun dato. Tali informazioni risultano interessanti nel momento in cui vengono confrontate con quelle acquisite dal dispositivo android.

4.7.2 ROS Topic e Message wifi

Il dispositivo android, di cui l'applicazione web fa largo utilizzo per l'acquisizione dei dati, prevede un topic atto alla pubblicazione delle informazioni relative alla connessione wireless dello smartphone. Tale topic prende il nome di *senspub_msgs/wifi*.

```
ros@ros: ~
ros@ros:~$ rosmsg show senspub_msgs/wifi
string SSID
string BSSID
string MAC
string Supplicant_State
int32 RSSI
int32 Link_speed
int32 Frequency
int32 NetworkId
```

Figura 4.33: ROS Message *senspub_msgs/wifi*

Il messaggio utilizzato per l'acquisizione dei dati relativi alla connessione wireless è mostrato nella Figura 4.33. I campi contenuti all'interno del succitato messaggio sono descritti di seguito:

- **string SSID**: acronimo di *Service Set Identifier*, indica il nome con cui una rete Wifi si identifica ai suoi utenti;
- **string BSSID**: iniziali di *Basic Service Set Identifier*, identifica, in maniera univoca, l'access point a cui è collegato il dispositivo android;
- **string MAC**: acronimo di *Media Access Control*, rappresenta un codice di 48 bit assegnato dal produttore ad ogni scheda di rete ethernet o wireless;
- **string Supplicant_State**: restituisce, in maniera dettagliata, lo stato di negoziazione con un access point. Tale parametro può assumere i seguenti valori:
  - ASSOCIATED: completamente associato;
  - COMPLETED: autenticazione completata;
  - DISCONNECTED: il client non risulta associato.
- **int32 RSSI**: cfr. p. 85;

- **int32 Link_speed**: cfr. p. 85;
- **int32 Frequency**: rappresenta la frequenza corrente in MHz;
- **int32 NetworkId**: ogni rete configurata è identificata univocamente da un ID, utilizzato quando si eseguono operazioni sul supplicant.

Così come i parametri del messaggio *network_analysis/NetworkSensor*, anche il sopracitato è stato utilizzato per acquisire le informazioni relative alla connessione wireless. Questa volta, però, le informazioni ottenute riguardano lo stato wireless relativo al dispositivo android. Il topic utilizzato varia in base al nome assegnato al dispositivo android (*/phoneName/android/wifi*). Lo stesso topic, una volta sottoscritto, viene utilizzato per ricavare le opportune informazioni. Di seguito vengono illustrati i metodi JavaScript che inizializzano il topic sopracitato, dai quali è possibile ottenere le informazioni relative alla connessione wireless, alla potenza del segnale e alla velocità di trasmissione delle informazioni. Le informazioni relative alla connessione wireless sono contenute all'interno di una tabella, mentre i valori della potenza del segnale ricevuto e della velocità di trasmissione sono visualizzati anche mediante grafici bidimensionali, così come visto per il caso della scheda di rete del personal computer. Ogni funzione ha il compito di attivare l'acquisizione di specifiche informazioni, usufruendo unicamente dello stesso topic. In questo modo, l'utente può scegliere quali informazioni utili visualizzare.

```
/**
 * READ ANDROID NETWORK INFO
 * This function read the sensor from Android Network Info.
 */
function readAndroidNetworkInfo(phoneName) {

    // Init topic object
    setAndroidNetworkInfo = new ROSLIB.Topic({
        ros: ros,
        name: phoneName + '/android/wifi',
        messageType: 'senspub_msgs/wifi'
    });

}
```

Figura 4.34: Sorgente JavaScript per l'inizializzazione del topic */phoneName/android/wifi* riguardo le informazioni di rete

```javascript
/**
 * READ ANDROID RSSI
 * This function read the sensor from Android RSSI.
 */
function readAndroidRSSI(phoneName) {

    // Init topic object
    setAndroidRSSI = new ROSLIB.Topic({
        ros: ros,
        name: phoneName + '/android/wifi',
        messageType: 'senspub_msgs/wifi'
    });

}
```

Figura 4.35: Sorgente JavaScript per l'inizializzazione del topic */phoneName/android/wifi* riguardo la potenza del segnale

```javascript
/**
 * READ ANDROID LINK SPEED
 * This function read the sensor from Android link speed.
 */
function readAndroidLinkSpeed(phoneName) {

    // Init topic object
    setAndroidLinkSpeed = new ROSLIB.Topic({
        ros: ros,
        name: phoneName + '/android/wifi',
        messageType: 'senspub_msgs/wifi'
    });

}
```

Figura 4.36: Sorgente JavaScript per l'inizializzazione del topic */phoneName/android/wifi* riguardo la velocità di trasmissione del segnale

4.8 Informazioni sulla rete cellulare LTE

All'interno dell'applicazione web è prevista un'apposita sezione dedicata all'analisi della rete cellulare. Si definisce rete cellulare «una rete di telecomunicazione che permette la comunicazione tra più nodi di rete» [30]. Mediante l'ausilio di ricevitori presenti all'interno del dispositivo android, è stato possibile acquisire dati dedicati alla valutazione di una rete cellulare. In particolare, l'interfaccia web prevede l'analisi dedicata allo standard di telefonia mobile **LTE** [31]. I parametri analizzati per valutare la qualità di connessione LTE, presente sul dispositivo android, sono i seguenti:

- **LTE Signal Strength**: rappresenta l'intensità di potenza del trasmettitore ricevuta da un'antenna di riferimento ad una certa distanza dall'antenna trasmittente [32];
- **LTE RSSNR**: acronimo di *Reference Signal to Noise Ratio*, rappresenta il rapporto tra la potenza del segnale e la potenza del rumore di fondo [33];
- **LTE RSRP**: *Reference Signal Received Power* è la potenza dei segnali di riferimento LTE distribuiti su tutta la larghezza di banda e banda stretta [34];
- **LTE RSRQ**: *Reference Signal Received Quality* indica la qualità del segnale di riferimento ricevuto [34];
- **LTE CQI**: acronimo di *Channel Quality Indicator*, si tratta di un indicatore che riporta le informazioni sulla qualità del canale di comunicazione [35].

4.8.1 ROS Topic e Message telephone

Le informazioni relative alla rete cellulare LTE sono contenute all'interno di un messaggio ROS creato su misura: *senspub_msgs/telephone*.

30. Cfr. www.treccani.it/enciclopedia/rete-cellulare
31. *Long Term Evolution* rappresenta la più recente tecnologia dedicata agli standard di telefonia mobile. Obiettivo è quello di promuovere l'utilizzo della banda larga in modo tale da raggiungere una velocità di connessione anche superiore a 1 Gbit/s.
32. Cfr. https://en.wikipedia.org/wiki/Signal_strength_in_telecommunications
33. Cfr. https://en.wikipedia.org/wiki/Signal-to-interference-plus-noise_ratio
34. Cfr. https://wiki.teltonika.lt/view/RSRP_and_RSRQ
35. Cfr. http://www.sharetechnote.com/html/Handbook_LTE_CQI.html

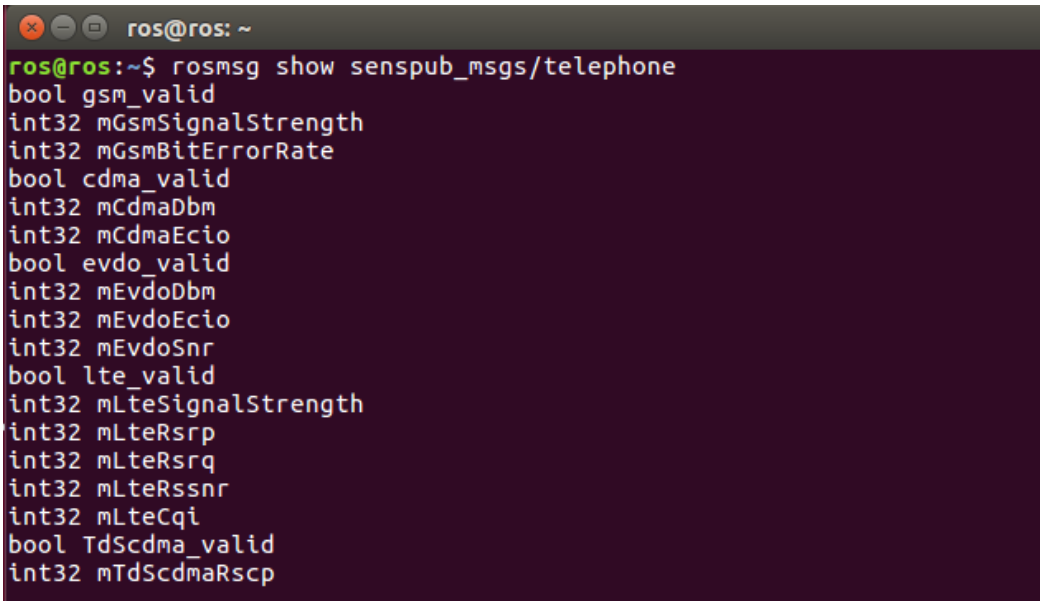


Figura 4.37: ROS Message *senspub_msgs/telephone*

Il messaggio ROS realizzato appositamente per l'analisi della rete cellulare è descritto nella Figura 4.37. Questo prevede molteplici campi che descrivono, in maniera dettagliata, le varie tecnologie di telefonia mobile. Nel messaggio sono presenti cinque variabili booleane che permettono di individuare quale sia lo standard di telefonia mobile in corso d'uso. L'applicazione web è stata creata per analizzare lo standard LTE. I campi soggetti a tale tecnologia sono i seguenti: **bool lte_valid** (tale valore booleano indica che i dati acquisiti sono di tipo LTE), **int32 mLteSignalStrength**, **int32 mLteRsrp**, **int32 mLteRsrq**, **int32 mLteRssnr**, **int32 mLteCqi** [36].

Pertanto, il suddetto messaggio è stato utilizzato per ricavare i parametri atti ad una corretta analisi della rete cellulare. Il topic messo a disposizione dall'applicazione android, installata sullo smartphone, è definito */phoneName/android/telephony*. Le informazioni acquisite sono state visualizzate all'interno di grafici bidimensionali, parimenti avviene per i sensori già descritti. Mediante la libreria *roslibjs*, è stato possibile sottoscrivere il suddetto topic e ricavarne i parametri opportuni. Le informazioni acquisite sono state trascritte all'interno di una tabella. I valori di potenza del segnale e qualità del canale di comunicazione, inoltre, sono stati rappresentati mediante opportuni grafici bidimensionali.

36. Cfr. p. 92

```javascript
/**
 * READ LTE NETWORK INFO
 * This function read the sensor from Android LTE Network Info.
 */
function readLTENetworkInfo(phoneName) {

    // Init topic object
    setLTENetworkInfo = new ROSLIB.Topic({
        ros: ros,
        name: phoneName + '/android/telephony',
        messageType: 'senspub_msgs/telephone'
    });

}
```

Figura 4.38: Sorgente JavaScript per l'inizializzazione del topic */phoneName/android/telephony* riguardo le informazioni di rete LTE

```javascript
/**
 * READ LTE RSRP RSRQ
 * This function read the sensor from Android LTE RSRP RSRQ.
 */
function readLTERsrpRsrq(phoneName) {

    // Init topic object
    setLTERsrpRsrq = new ROSLIB.Topic({
        ros: ros,
        name: phoneName + '/android/telephony',
        messageType: 'senspub_msgs/telephone'
    });

}
```

Figura 4.39: Sorgente JavaScript per l'inizializzazione del topic */phoneName/android/telephony* riguardo la potenza e la qualità del segnale di riferimento

```
/**
 * READ LTE CQI
 * This function read the sensor from Android LTE CQI.
 */
function readLTECqi(phoneName) {

    // Init topic object
    setLTECqi = new ROSLIB.Topic({
        ros: ros,
        name: phoneName + '/android/telephony',
        messageType: 'senspub_msgs/telephone'
    });

}
```

Figura 4.40: Sorgente JavaScript per l'inizializzazione del topic */phoneName/android/telephony* riguardo la qualità del canale di comunicazione

Capitolo 5

Analisi dei risultati ottenuti

L'applicazione web realizzata in concomitanza con lo svolgimento del lavoro di tesi è stata creata per monitorare da remoto un dispositivo mobile mediante le tecnologie Web Based. Questo lavoro si è rivelato utile per comprendere come sia possibile interfacciarsi al robot mediante il framework Robot Operating System. Attraverso l'utilizzo di opportuni messaggi contenuti all'interno della libreria ROS, è stato possibile comunicare con il rover utilizzato e con il dispositivo android ubicato sul robot. Quest'ultimo, sfruttando l'applicazione **PIC4SeR Monitoring**, ha fornito informazioni utili, le quali sono state monitorate usufruendo dell'applicazione web **ROS Web Interface**.

Le varie misurazioni sono state condotte mediante l'utilizzo di un dispositivo android posizionato su un robot. Sono stati osservati: l'andamento temporale delle informazioni acquisite dai vari sensori; l'interazione tra il rover e la stazione remota. Inoltre, a partire dai dati acquisiti, sono stati realizzati i relativi grafici. In definitiva, il presente elaborato propone un quadro generale che mira al contempo ad esplorare ed individuare gli sviluppi futuri. Si faccia un'osservazione ulteriore: le misurazioni sono state realizzate sfruttando il rover **Clearpath Jackal**, da remoto tramite l'utilizzo dei topic ROS messi a disposizione. Tuttavia, è stato necessario effettuare tutte le dovute configurazioni affinché il sistema fosse in grado di interagire correttamente.

5.1 Clearpath Jackal

Clearpath Robotics Inc. è un'industria robotica fondata nel 2009 da un gruppo di quattro ricercatori dell'Università di Waterloo ed ha sede nella

regione Waterloo in Canada [37]. Inizialmente, l'obiettivo dell'azienda è stato quello di realizzare un robot per ricerche universitarie. Successivamente, la vendita è aumentata ed è stata rivolta anche ad ambienti industriali, commercializzando la linea di veicoli OTTO [38].

Il rover della linea Clearpath Robotics Inc. messo a disposizione dal Politecnico di Torino, in collaborazione con il team di ricerca *PIC4SeR* [39], è stato il **Clearpath Jackal Unmanned Ground Vehicle**. Si tratta di una piccola piattaforma di ricerca robotica mobile. La strumentazione comprende un computer a bordo e due sensori completamente integrati (GPS e IMU), ai quali è possibile aggiungere una moltitudine di accessori per gli sviluppi futuri.

Figura 5.1: Clearpath Jackal

5.1.1 Specifiche del Clearpath Jackal

Il Clearpath Jackal utilizzato per effettuare vari test di verifica e successive analisi presenta le specifiche elencate di seguito.

SIZE AND WEIGHT	
External Dimensions (L x W x H)	508 x 403 x 250 mm (20 x 17 x 10 in)
Internal Storage Dimensions	250 x 100 x 85 mm (10 x 4 x 3 in)
Weight	17 kg (37 lbs)
Ground Clearance	65 mm (2.6 in)

Tabella 5.1: Dimensioni del Clearpath Jackal

37. Cfr. https://en.wikipedia.org/wiki/Clearpath_Robotics
38. Il marchio OTTO è stato annunciato nel 2016 ed è stato utilizzato per la prima volta da Clearpath Robotics Inc. per il commercio di veicoli autonomi. Il primo cliente è stato General Electric, al quale si sono aggiunti John Deere e Toyota.
    Cfr. https://en.wikipedia.org/wiki/Clearpath_Robotics
39. *Polito Interdepartmental Centre for Service Robotics* - Centro di sviluppo per la robotica con l'obiettivo di progettare, simulare e realizzare le basi che favoriranno l'avvento della nuova era dei «robot di servizio». Diversi scenari applicativi vengono presi in considerazione: agricoltura di precisione, ricerca e salvataggio, sostegno alla vita per anziani e disabili, città intelligenti, indagine archeologica/sicurezza/protezione, monitoraggio ambienti circostanti, etc.
    Cfr. https://www.polito.it/ricerca/centri/pic4ser/

SPEED AND PERFORMANCE	
Maximum Payload	20 kg (44 lbs)
All-Terrain Payload	10 kg (22 lbs)
Maximum Speed	2.0 m/s (6.6 ft/s)
Drive Power	500 W

Tabella 5.2: Performance del Clearpath Jackal

BATTERY AND POWER SYSTEM	
Battery Chemistry	Lithium Ion
Capacity	270 Watt hours
Charge Time	4 hours
Run Time	Heavy usage: 2 hours, Basic usage: 8 hours
User Power	5 V at 5 A, 12 V at 10 A, 24 V at 20 A

Tabella 5.3: Sistema di potenza del Clearpath Jackal

INTERFACING AND COMMUNICATION	
Control Modes	Kinematic Commands - velocity, angular velocity
	Open Loop Motor Driver Commands - voltage
	Wheel Velocity Commands
Feedback	Battery and motor current
	Wheel velocity and travel
	Integrated GPS receiver
	Integrated gyroscope and accelerometer
Communication	Ethernet, USB 3.0, RS232. (IEEE 1394 available)
Drivers and APIs	Packaged with ROS Kinetic
Integrated Accessories (included)	Wireless Game controller
	GPS, IMU
	On-Board Computer
	WIFI Adapter
	Accessory Mounting Plates

Tabella 5.4: Interfacciamento e comunicazione del Clearpath Jackal

COMPUTER	**Standard**	**Performance**
	Celeron J1800	Intel Core i5 4570T
	Dual core, 2.4 GHz	Dual core, 2.9 GHz
	2 GB RAM	4 GB RAM
	WIFI Adapter	WIFI Adapter
	32 GB Hard Drive	128 GB Hard Drive

Tabella 5.5: Specifiche del computer del Clearpath Jackal

ENVIRONMENTAL	
Operating Ambient Temperature	-20°C to 45°C (-4°F to 113°F)
IP Rating	62 (65 available on select configurations)

Tabella 5.6: Specifiche ambientali del Clearpath Jackal

5.1.2 Configurazione del Clearpath Jackal

Affinché il robot Clearpath Jackal sia in grado di comunicare con l'applicazione web, si rende necessaria la sua configurazione. Questo permette al robot di inviare e ricevere correttamente i messaggi ROS (antecedentemente descritti). Il tempo debito all'accensione del PC interno al robot è esattamente 30 secondi dalla selezione del relativo tasto d'avvio. Una volta acceso, è possibile connetterlo al proprio personal computer, mediante connessione wireless, eseguendo tre procedure di configurazione.

Configurazione IP Statico La configurazione statica dell'indirizzo IP prevede di impostare la porta ethernet del proprio laptop su un indirizzo IP statico. L'indirizzo da utilizzare è: 192.168.1.51. Per configurarlo sul sistema operativo Ubuntu, bisogna effettuare le seguenti procedure:

1. Cliccare sull'icona WiFi in alto a destra dello schermo, e selezionare **Modifica connessioni**;
2. Nella finestra **Connessioni di rete**, all'interno della voce **Ethernet**, scegliere la connessione via cavo e dopo cliccare su **Modifica**;
3. Premere il tab **Impostazioni IPv4** e modificare il **Metodo** al valore *Manuale*;
4. Cliccare sul tasto **Aggiungi** per aggiungere un nuovo indirizzo IP;
5. Inserire *192.168.1.51* come indirizzo IP statico nella colonna **Indirizzo**, e *255.255.255.0* nella colonna **Maschera**, e dopo premere il tasto **Salva**.

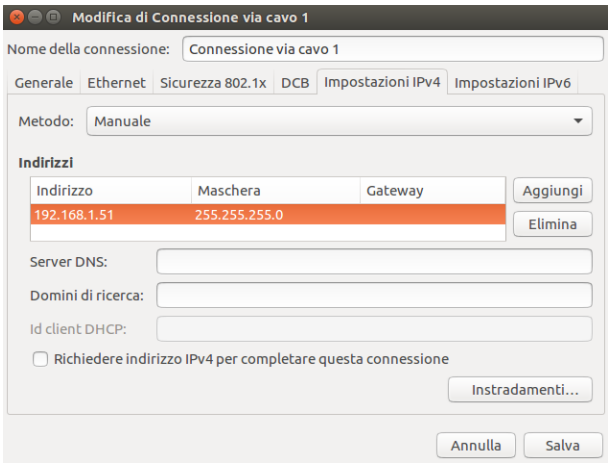


Figura 5.2: Configurazione IP Statico

Connessione mediante SSH over Ethernet A questo punto, si proceda alla connessione al Clearpath Jackal utilizzando il tool **SSH** [40].

Per effettuare la connessione tramite SSH, bisogna eseguire il seguente comando da terminale:

```
$ ssh administrator@192.168.1.11
```

Affinché vada a buon fine, bisogna immettere una password. Quella predefinita è: *clearpath*.

Connessione mediante Wireless Network Una volta aperta la connessione con il rover tramite SSH, è possibile collegare il Clearpath Jackal alla rete locale. Pertanto, è possibile utilizzare uno dei tool di configurazione dell'interfaccia wireless presente nei sistemi operativi Linux: **WICD** [41]. Dal

40. SSH (SSH client) è un tool che permette sia l'accesso che l'esecuzione di comandi su una macchina remota. Fornisce delle comunicazioni crittografate sicure tra due host su una rete non sicura. È possibile inoltrare informazioni su connessioni X11, porte TCP arbitrarie e socket di dominio UNIX. SSH permette la connessione e la registrazione dell'host in base al nome specificato. L'utente deve mostrare la sua identità alla macchina remota utilizzando vari metodi.
Cfr. https://it.wikipedia.org/wiki/Secure_Shell
41. WICD è un gestore di rete cablata e wireless open source per i sistemi Linux che mira a fornire una semplice interfaccia per connettersi alle reti con un'ampia varietà di impostazioni. Fornisce supporto solo per reti cablate e wireless, ma è una buona alternativa a Network Manager se questo dà problemi.
Cfr. https://help.ubuntu.com/community/WICD

terminale, si digiti il seguente comando:

```
$ wicd-curses
```

A questo punto, è presente un elenco delle reti che il robot ha rilevato. Usando i tasti freccia, si selezioni la rete desiderata e la si configuri premendo la freccia destra. Una volta conclusa la configurazione, bisogna premere **F10** per salvare, e dopo **C** per effettuare la connessione. Quindi, il robot Clearpath Jackal risulta connesso.

Per visualizzare l'indirizzo IP della connessione wireless del robot, è necessario eseguire da terminale il seguente comando:

```
$ ifconfig
```

Dunque figura un elenco di connessioni di rete. Individuata la rete wireless, bisogna prendere nota dell'indirizzo IP del robot e, quindi, terminare l'istanza SSH mediante il comando *exit*. Una volta configurato, è possibile connettersi al robot, mediante la rete wireless, eseguendo il comando:

```
$ ssh administrator@<IP_OF_JACKAL>
```

Tali sessioni SSH permettono di controllare il computer interno al robot (download pacchetti, esecuzione di aggiornamenti, aggiunta/rimozione/trasferimento file, etc.).

5.1.3 Interazione tra ROS Web Interface e Clearpath Jackal

Avendo configurato il robot Clearpath Jackal si comprende, pertanto, come interagire con esso sfruttando l'applicazione **ROS Web Interface**. Anzitutto, è necessario connettersi al robot (conoscendo già il suo indirizzo IP) eseguendo il comando:

```
$ ssh administrator@<IP_OF_JACKAL>
```

Successivamente, è necessario modificare il file di configurazione *.bashrc* del sistema operativo presente all'interno del rover, mediante la configurazione delle variabili globali *ROS_MASTER_URI* e *ROS_HOSTNAME*. Per fare ciò, è possibile eseguire i seguenti comandi:

```
$ echo "export ROS_MASTER_URI=http://ip_address_pc
  :11311" >> ~/.bashrc
$ echo "export ROS_HOSTNAME=ip_address_jackal" >> ~/.
  bashrc
$ source ~/.bashrc
```

La variabile globale *ROS_MASTER_URI* è stata configurata di modo che il rover Clearpath Jackal sia in grado di comunicare con il «master» ROS, avviato nel personal computer. Inoltre, alla variabile d'ambiente *ROS_HOSTNAME* è stato associato l'indirizzo IP che identifica il robot.

Affinché l'applicazione web sia in grado di interagire correttamente con il rover, è opportuno avviare due nodi. Questi pubblicano i topic relativi alla teleoperazione e alla trasmissione delle immagini mediante telecamera a bordo. I nodi devono essere eseguiti all'interno della sessione SSH, grazie alla quale è stato possibile connettersi al robot. Il comando atto all'avvio del nodo dedicato alla teleoperazione del robot è il seguente:

```
$ roslaunch jackal_base base.launch
```

Tuttavia, il succitato file *.launch* non è stato utilizzato solamente per teleoperare il rover mediante messaggi ROS; è stato, altresì, fondamentale per la sottoscrizione del topic */navsat/fix*, il quale fornisce le informazioni sulla geolocalizzazione del robot stesso. Un ulteriore comando da utilizzare per usufruire della telecamera, ubicata sul robot, è il seguente:

```
$ roslaunch realsense_camera r200_nodelet_rgbd.launch
```

Una volta avviati entrambi i nodi, l'utente può utilizzare l'applicazione web interagendo con il Clearpath Jackal. L'interfaccia web è stata realizzata per poter comunicare con il rover nei seguenti tre casi:

- gestione del monitoraggio mediante i comandi per la teleoperazione;
- visualizzazione delle informazioni relative alla geolocalizzazione del robot mediante il topic */navsat/fix*;
- visualizzazione delle immagini trasmesse dalla telecamera montata a bordo.

5.2 ROS Web Interface

L'applicazione **ROS Web Interface**, realizzata per il monitoraggio da remoto di un robot e per la gestione dei dati multimediali, è una soluzione software esclusivamente di tipo *client*. Di fatto, le informazioni acquisite sono gestite lato front-end sfruttando la libreria JavaScript **roslibjs**. Inoltre, l'applicazione è in grado di sottoscrivere le informazioni pubblicate dal dispositivo android solo se l'ambiente ROS è stato antecedentemente configurato. Una volta installati i packages (introdotti nel primo capitolo) all'interno del proprio personal computer, è possibile utilizzare l'interfaccia web effettuando i seguenti passi:

1. Aprire un terminale ed eseguire da linea di comando il file **init.sh** (come descritto nel terzo capitolo); affinché tutti i nodi siano stati eseguiti correttamente, una volta digitato il comando *screen -ls*, i nodi attivi dovranno essere cinque;

2. Eseguire il software **Visual Studio Code**, in cui è stata installata l'estensione **Live Server**;

3. Avviare un server locale sulla porta 5500 cliccando sul tasto *Go Live* presente nella status bar del software **Visual Studio Code**;

4. Accedere alla directory del progetto sfruttando un **Browser Web**: a tale scopo, è opportuno digitare la URL *localhost:5500*;

5. Aprire il file sorgente *index.html* del progetto a partire dalla directory */src/html*.

5.2.1 Funzionamento dell'applicazione web

Eseguendo i passaggi descritti precedentemente per la configurazione dell'ambiente e l'avvio del server virtuale, l'interfaccia web è pronta per poter interagire con il robot **Clearpath Jackal** e con l'applicazione android **PIC4SeR Monitoring**.

L'utente, affinché possa interagire con l'applicazione android, deve necessariamente collegare i due dispositivi al medesimo *access point*. Una volta digitato il corretto indirizzo IP sull'applicazione android, essa è in grado di connettersi al **ROS Master** avviato sul PC e, quindi, pubblicare i messaggi in modo corretto.

L'interfaccia web è stata realizzata su un'unica pagina web; pertanto, è possibile scegliere una specifica sezione senza che sia, di volta in volta, necessario ricaricare l'interfaccia web. Di fatto, effettuando uno scrolling della pagina web, viene visualizzata la relativa sezione. Tale scelta progettuale permette di visualizzare i sensori dedicati senza perdere le informazioni acquisite. Inoltre, l'applicazione web è stata progettata rispettando l'approccio del *Responsive Design* [42].

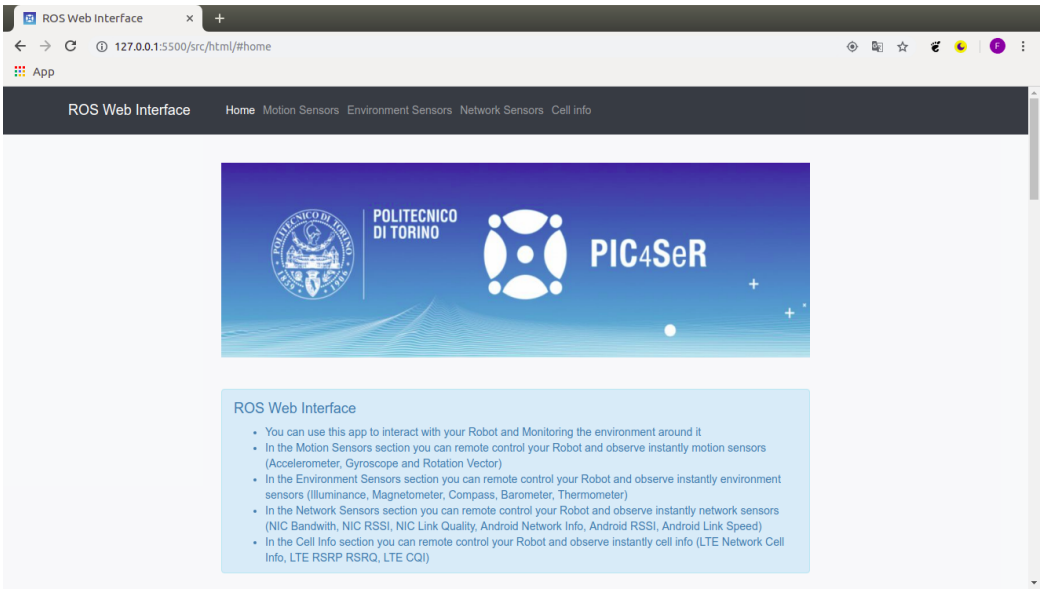


Figura 5.3: ROS Web Interface - Home Page

Nella Figura 5.3 è mostrata la *Home Page* dell'interfaccia web. È presente una breve descrizione delle funzionalità a disposizione. Il menu in alto, inoltre, permette di accedere direttamente alla sezione di interesse affinché l'utente possa effettuare le dovute misurazioni. Il menu rimane fisso in alto, nonostante l'utente modifichi la sezione dell'interfaccia web.

ROS Web Interface prevede quattro sezioni dalle quali è possibile interagire. Tale soluzione trova la sua esplicazione nella suddivisione delle tipologie di sensori da analizzare. La prima sezione prevede i sensori di movimento, dai quali è possibile osservare l'andamento dell'accelerometro, del

42. Con *Responsive Design* si indichi quell'approccio per il quale la progettazione e lo sviluppo di un'interfaccia web dovrebbero adattarsi al comportamento e all'ambiente dell'utente in base a fattori come le dimensioni dello schermo, la piattaforma e l'orientamento del dispositivo.

giroscopio e del vettore di rotazione. Nella sezione seguente, invece, sono presenti i valori acquisiti dai sensori d'ambiente. È possibile individuare: i valori acquisiti dal sensore dell'illuminamento; le coordinate cartesiane del campo magnetico; i valori di Roll, Pitch e Azimuth che individuano il Nord Magnetico; i valori acquisiti dai sensori di temperatura e di pressione presenti all'interno del dispositivo android. La terza sezione mostra l'andamento dei parametri prestazionali della scheda di rete NIC e della scheda wireless android. Infine, nella quarta sezione, sono mostrate le informazioni relative alla cella telefonica alla quale il dispositivo android è collegato, visualizzando anche i parametri LTE.

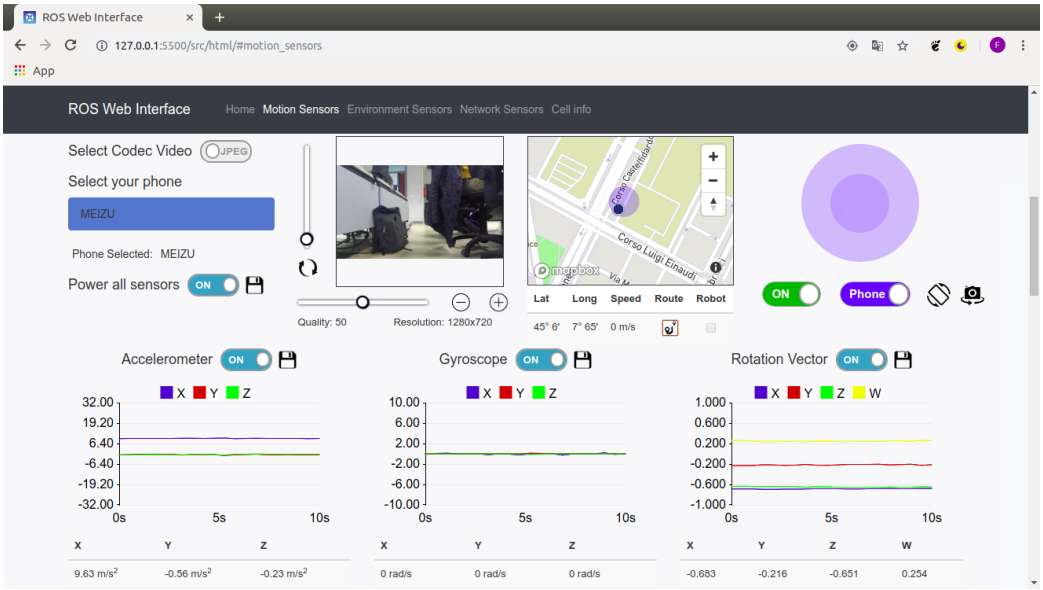


Figura 5.4: ROS Web Interface - Motion Sensors

La sezione dedicata ai sensori di movimento prevede, come mostrato in Figura 5.4, i tasti di selezione che permettono all'utente di usufruire di varie opzioni. Anzitutto, è opportuno decidere quale codec video utilizzare per l'acquisizione delle immagini. Mediante un tasto di selezione, di fatto, l'utente può scegliere la tipologia *x264* oppure *jpeg*. Una volta effettuata tale scelta, è possibile decidere come acquisire le informazioni relative alla geolocalizzazione. Questa può essere effettuata spuntando o meno la relativa casella posta sotto la mappa. Selezionando tale casella, si decide di acquisire le informazioni dal sensore GPS presente sul robot. Successivamente, è possibile scegliere il dispositivo android dal quale effettuare le misurazioni. La

scelta di tale dispositivo fa sì che le scelte antecedenti non possano essere più mutate. È possibile modificarle solamente ricaricando la pagina web.

In un secondo momento, l'utente può selezionare quale telecamera utilizzare per l'acquisizione delle immagini video. La selezione avviene usufruendo di due tasti di selezione situati sotto il joystick. Difatti, questi permettono l'accensione della fotocamera e la scelta tra le due disponibili. Inoltre, sono interessanti i servizi a disposizione della fotocamera android:

- è possibile scegliere la risoluzione delle immagini trasmesse tra quelle supportate dallo smartphone;
- l'opportunità di modificare la compressione delle immagini acquisite sfruttando un opportuno range slider;
- la possibilità di ruotare le immagini visualizzate;
- l'opportunità di scegliere tra la fotocamera posteriore o anteriore.

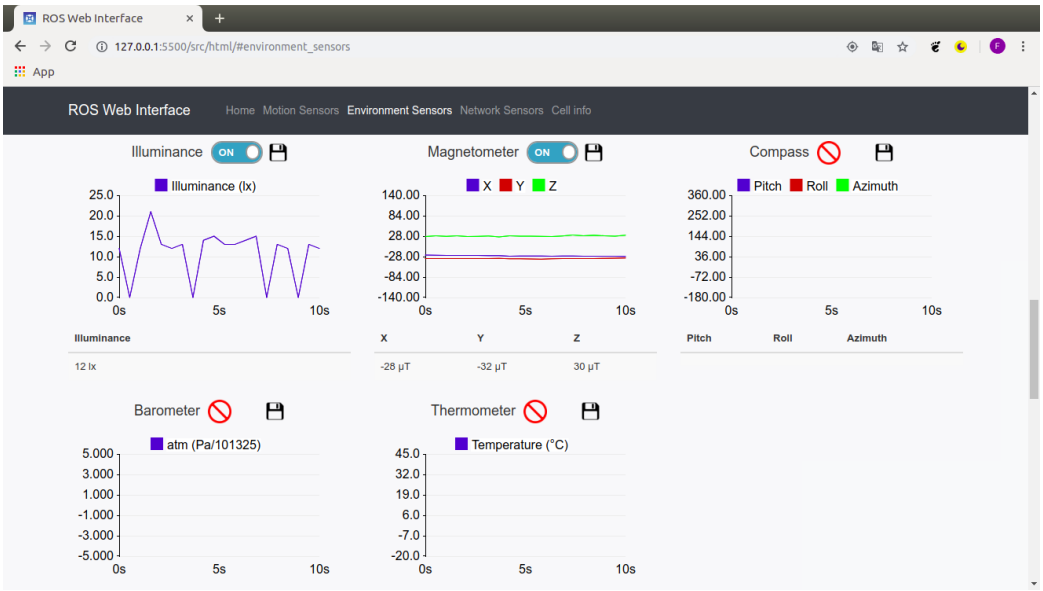


Figura 5.5: ROS Web Interface - Environment Sensors

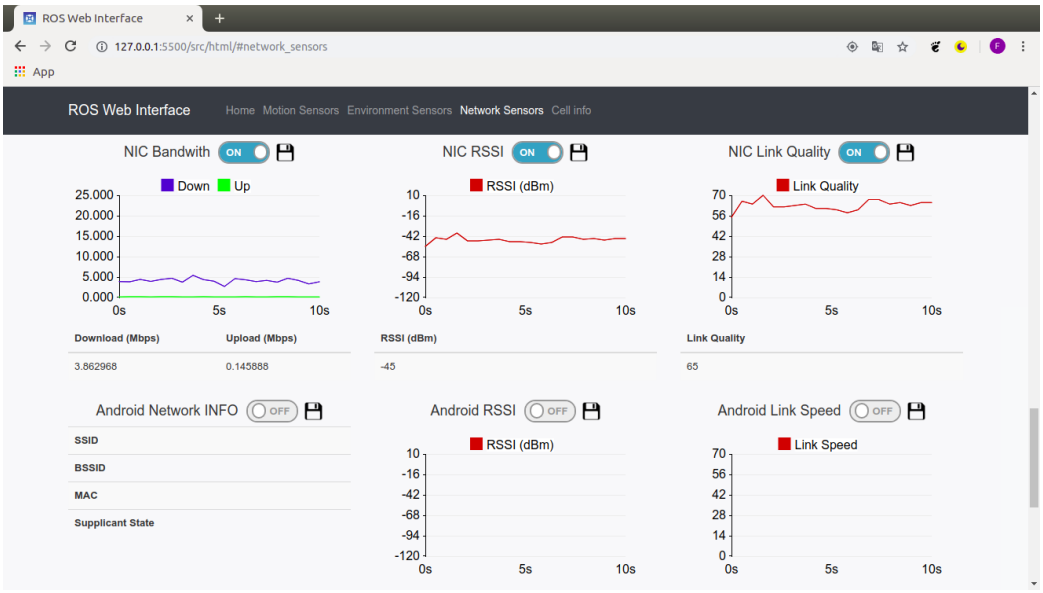


Figura 5.6: ROS Web Interface - Network Sensors

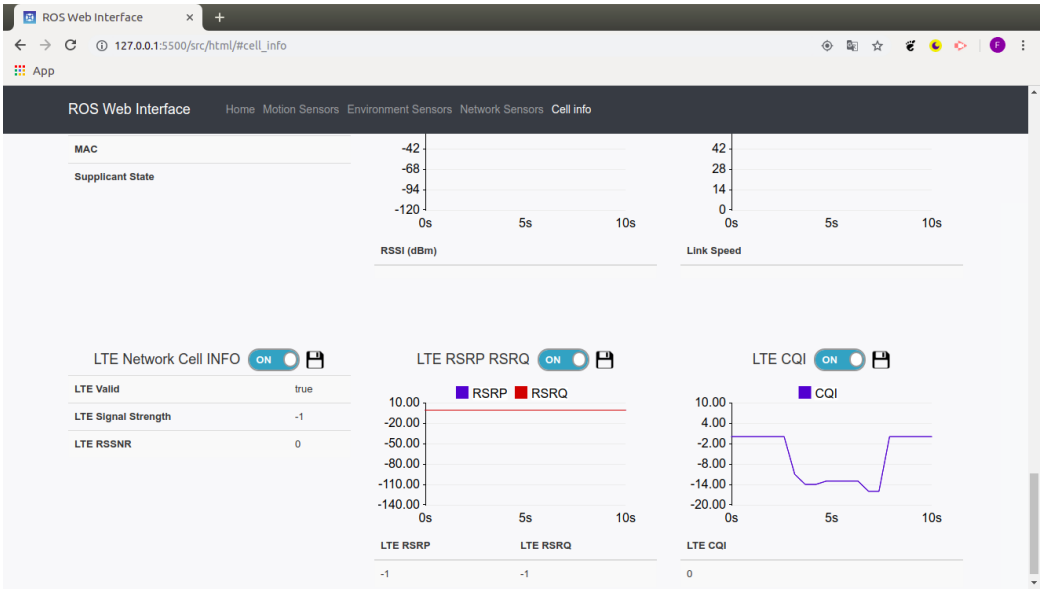


Figura 5.7: ROS Web Interface - Cell Info

I vari sensori, come mostrato nelle Figure 5.5, 5.6 e 5.7, possono essere abilitati utilizzando i relativi tasti di accensione. L'interfaccia web, in particolare, è in grado di rilevare automaticamente la presenza o meno di attuatori

all'interno del dispositivo android. Per esempio, nella Figura 5.5, è possibile osservare l'assenza dei sensori di temperatura e pressione, dai quali non è stato possibile ricavare alcuna informazione. Pertanto, l'interfaccia web nasconde il tasto di selezione per l'accensione dell'apposito sensore. Inoltre, l'interfaccia prevede anche l'analisi dei dati ricavati dalla bussola; tuttavia, tale funzionalità non è stata ancora implementata nell'applicazione android.

È possibile, inoltre, memorizzare all'interno di file JSON le informazioni acquisite, utilizzando l'apposita icona di salvataggio. Una volta salvati i file, l'utente può visualizzare il loro contenuto all'interno di grafici bidimensionali. Questi vengono generati usufruendo di opportuni scripts realizzati in Python. Tali sono contenuti all'interno della directory */scripts* del progetto. Tuttavia, per poter usufruire di tali funzionalità, l'utente deve necessariamente installare dei packages all'interno del proprio sistema operativo. È opportuno, perciò, eseguire da terminale il file bash *install_python_lib.sh*, il quale, automaticamente, importa le dipendenze Python utili per la generazione dei grafici.

L'utente, quindi, può scegliere quali grafici visualizzare, utilizzando i file JSON precedentemente salvati. Questi vengono processati come file di input dagli scripts. Sono stati realizzati secondo le quattro tipologie di sensori analizzati. La scelta, pertanto, deve essere coerente: è necessario utilizzare lo script della stessa tipologia del file JSON. Un esempio di utilizzo degli scripts è mostrato di seguito:

```
$ python graph_motion_sensors.py accelerometer.json
```

Il comando mostrato in precedenza permette di generare un grafico contenente le informazioni relative all'accelerometro. Allo stesso modo, è possibile generare altrettanti grafici, considerando le altre tipologie di sensori. Tuttavia, è importante non modificare il nome assegnato automaticamente ai file JSON generati al momento del salvataggio. Affinché gli scripts siano in grado di generare i grafici, inoltre, è opportuno che i file JSON vengano spostati all'interno della directory */scripts* del progetto. In questo modo, è possibile eseguire la «processazione» dei dati contenuti all'interno dei file. I grafici visualizzati possono essere salvati nel proprio PC usufruendo di vari formati messi a disposizione dal tool utilizzato all'interno degli scripts Python.

5.3 Risultati sperimentali

Le informazioni, acquisite mediante l'utilizzo di un dispositivo android ubicato sul rover, descrivono la qualità del servizio realizzato. Le misurazioni sono state effettuate in un ambiente outdoor, in quanto l'applicazione è stata realizzata per il monitoraggio di ambienti esterni. È stato utilizzato un access point sfruttando la tecnologia *hotspot* di un dispositivo android, al quale sono stati collegati la stazione remota dell'esperimento (personal computer in cui è avviato **ROS Web Interface**), il dispositivo android collocato sul rover (avviando l'applicazione **PIC4SeR Monitoring**) e il rover Clearpath Jackal. Sono stati svolti diversi test, dai quali sono stati dedotti vari aspetti critici. Nell'elaborato vengono descritti alcuni di essi, mediante l'utilizzo di talune tabelle contenenti alcuni valori raccolti, e di grafici bidimensionali, in cui vengono riportati i valori acquisiti in determinati intervalli temporali.

5.3.1 Sensori di rete

I sensori di rete utilizzati per effettuare le varie misurazioni permettono di analizzare l'efficienza della connessione wireless ad una certa distanza. I parametri prestazionali presi in considerazione per tali misurazioni sono: la potenza del segnale di ricezione RSSI, la qualità del canale di comunicazione e la velocità di trasmissione. Vengono considerati due casi: le informazioni raccolte dalla scheda di rete NIC e i dati acquisiti dal dispositivo android. Entrambi i dispositivi sono collegati ad un unico access point fornendo, pertanto, le medesime informazioni di quest'ultimo, ma valori differenti di parametri prestazionali.

Scheda di rete NIC I dati raccolti dalla scheda di rete NIC vengono mostrati usufruendo di opportune tabelle, suddivise per tipologie di parametri prestazionali, e di grafici bidimensionali, realizzati dagli opportuni scripts in Python (come descritto antecedentemente). I parametri prestazionali considerati riguardano: l'occupazione della banda in upload e download, la potenza di ricezione del segnale RSSI e la qualità del canale di comunicazione.

Date	Upload	Download
Feb 13, 2019 16:42:06	0.396184	8.073488
Feb 13, 2019 16:42:09	0.217296	5.76772
Feb 13, 2019 16:42:12	0.275728	7.37788
Feb 13, 2019 16:42:15	0.437296	11.788536
Feb 13, 2019 16:42:18	0.290096	7.482784
Feb 13, 2019 16:42:21	0.227856	6.983984
Feb 13, 2019 16:42:24	0.561424	14.125
Feb 13, 2019 16:42:28	0.337648	10.29576
Feb 13, 2019 16:42:31	0.421056	9.862736
Feb 13, 2019 16:42:34	0.500672	13.990288
Feb 13, 2019 16:42:37	0.478192	13.302552
Feb 13, 2019 16:42:40	0.382016	11.599096
Feb 13, 2019 16:42:43	0.436584	12.897448
Feb 13, 2019 16:42:46	0.461952	12.882568
Feb 13, 2019 16:42:49	0.512216	13.6234
Feb 13, 2019 16:42:52	0.469024	12.858376
Feb 13, 2019 16:42:55	0.284744	7.140584
Feb 13, 2019 16:42:58	0.538336	11.284608
Feb 13, 2019 16:43:02	0.570288	13.716056
Feb 13, 2019 16:43:05	0.471136	11.487592
Feb 13, 2019 16:43:08	0.43884	11.859824
Feb 13, 2019 16:43:11	0.462016	11.974112
Feb 13, 2019 16:43:14	0.42604	12.288168
Feb 13, 2019 16:43:17	0.415856	12.402816
Feb 13, 2019 16:43:20	0.450544	12.630392
Feb 13, 2019 16:43:24	0.330248	9.71676
Feb 13, 2019 16:43:28	0.412848	12.401576
Feb 13, 2019 16:43:32	0.081968	1.3256
Feb 13, 2019 16:43:35	0.036848	0.093312
Feb 13, 2019 16:43:38	0.092688	1.162048
Feb 13, 2019 16:43:42	0.087424	1.036056
Feb 13, 2019 16:43:45	0.076896	0.840784
Feb 13, 2019 16:43:48	0.080464	1.147936
Feb 13, 2019 16:43:51	0.066368	0.692112
Feb 13, 2019 16:43:54	0.104544	0.993232

Tabella 5.7: Valori di banda upload e download - NIC [1]

Date	Upload	Download
Feb 13, 2019 16:43:57	0.090432	1.00824
Feb 13, 2019 16:44:00	0.109232	1.630296
Feb 13, 2019 16:44:03	0.116408	1.73748
Feb 13, 2019 16:44:06	0.123832	2.454168
Feb 13, 2019 16:44:09	0.154664	1.441632
Feb 13, 2019 16:44:12	0.109064	1.849968
Feb 13, 2019 16:44:16	0.234736	4.927128
Feb 13, 2019 16:44:19	0.218488	5.616352
Feb 13, 2019 16:44:22	0.213272	4.269656
Feb 13, 2019 16:44:25	0.129872	1.523976
Feb 13, 2019 16:44:28	0.260312	5.732992
Feb 13, 2019 16:44:32	0.313584	8.873112
Feb 13, 2019 16:44:35	0.257936	7.0682
Feb 13, 2019 16:44:38	0.365472	10.897968
Feb 13, 2019 16:44:41	0.349024	9.62356
Feb 13, 2019 16:44:44	0.381184	9.322992
Feb 13, 2019 16:44:47	0.380896	10.608064
Feb 13, 2019 16:44:50	0.397808	11.964664
Feb 13, 2019 16:44:53	0.336112	9.533192
Feb 13, 2019 16:44:56	0.235936	6.055416
Feb 13, 2019 16:44:59	0.478832	13.600832
Feb 13, 2019 16:45:02	0.338288	9.141248
Feb 13, 2019 16:45:06	0.21056	5.74472
Feb 13, 2019 16:45:09	0.216192	5.822504
Feb 13, 2019 16:45:12	0.127088	3.612424
Feb 13, 2019 16:45:16	0.208672	5.046352
Feb 13, 2019 16:45:19	0.379952	10.491456
Feb 13, 2019 16:45:22	0.187728	5.02648
Feb 13, 2019 16:45:25	0.078208	1.325152
Feb 13, 2019 16:45:29	0.097376	1.97412
Feb 13, 2019 16:45:32	0.084976	2.319056

Tabella 5.8: Valori di banda upload e download - NIC [2]

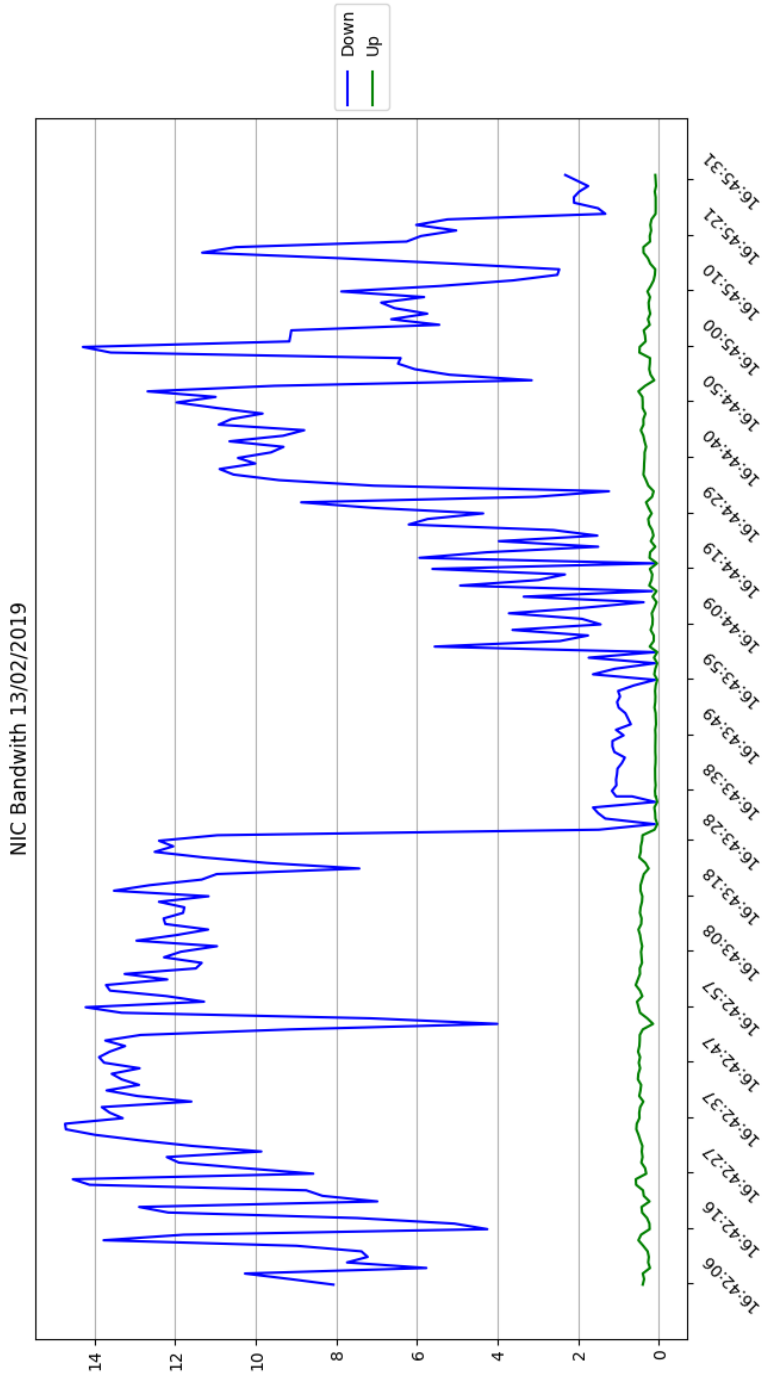


Figura 5.8: Rappresentazione grafica dei valori di banda - NIC

Date	RSSI
Feb 13, 2019 16:42:06	-46
Feb 13, 2019 16:42:09	-38
Feb 13, 2019 16:42:12	-45
Feb 13, 2019 16:42:15	-45
Feb 13, 2019 16:42:18	-42
Feb 13, 2019 16:42:21	-45
Feb 13, 2019 16:42:24	-45
Feb 13, 2019 16:42:28	-44
Feb 13, 2019 16:42:31	-45
Feb 13, 2019 16:42:34	-45
Feb 13, 2019 16:42:37	-45
Feb 13, 2019 16:42:40	-45
Feb 13, 2019 16:42:43	-45
Feb 13, 2019 16:42:46	-46
Feb 13, 2019 16:42:49	-46
Feb 13, 2019 16:42:52	-46
Feb 13, 2019 16:42:55	-46
Feb 13, 2019 16:42:58	-46
Feb 13, 2019 16:43:02	-46
Feb 13, 2019 16:43:05	-46
Feb 13, 2019 16:43:08	-45
Feb 13, 2019 16:43:11	-45
Feb 13, 2019 16:43:14	-46
Feb 13, 2019 16:43:17	-45
Feb 13, 2019 16:43:20	-45
Feb 13, 2019 16:43:24	-44
Feb 13, 2019 16:43:28	-45
Feb 13, 2019 16:43:32	-45
Feb 13, 2019 16:43:35	-45
Feb 13, 2019 16:43:39	-46
Feb 13, 2019 16:43:42	-45
Feb 13, 2019 16:43:45	-45
Feb 13, 2019 16:43:48	-39
Feb 13, 2019 16:43:51	-39
Feb 13, 2019 16:43:54	-39
Feb 13, 2019 16:43:57	-39
Feb 13, 2019 16:44:00	-39
Feb 13, 2019 16:44:03	-38
Feb 13, 2019 16:44:06	-42
Feb 13, 2019 16:44:09	-42
Feb 13, 2019 16:44:12	-44
Feb 13, 2019 16:44:16	-45
Feb 13, 2019 16:44:19	-45
Feb 13, 2019 16:44:22	-45
Feb 13, 2019 16:44:25	-46
Feb 13, 2019 16:44:28	-45
Feb 13, 2019 16:44:32	-45
Feb 13, 2019 16:44:35	-38
Feb 13, 2019 16:44:38	-36
Feb 13, 2019 16:44:41	-36
Feb 13, 2019 16:44:44	-36
Feb 13, 2019 16:44:47	-36
Feb 13, 2019 16:44:50	-36
Feb 13, 2019 16:44:53	-35
Feb 13, 2019 16:44:56	-38
Feb 13, 2019 16:44:59	-42
Feb 13, 2019 16:45:02	-45
Feb 13, 2019 16:45:06	-45
Feb 13, 2019 16:45:09	-45
Feb 13, 2019 16:45:13	-46
Feb 13, 2019 16:45:16	-39
Feb 13, 2019 16:45:19	-38
Feb 13, 2019 16:45:22	-38
Feb 13, 2019 16:45:25	-39
Feb 13, 2019 16:45:29	-39
Feb 13, 2019 16:45:32	-40

Tabella 5.9: Valori di RSSI - NIC

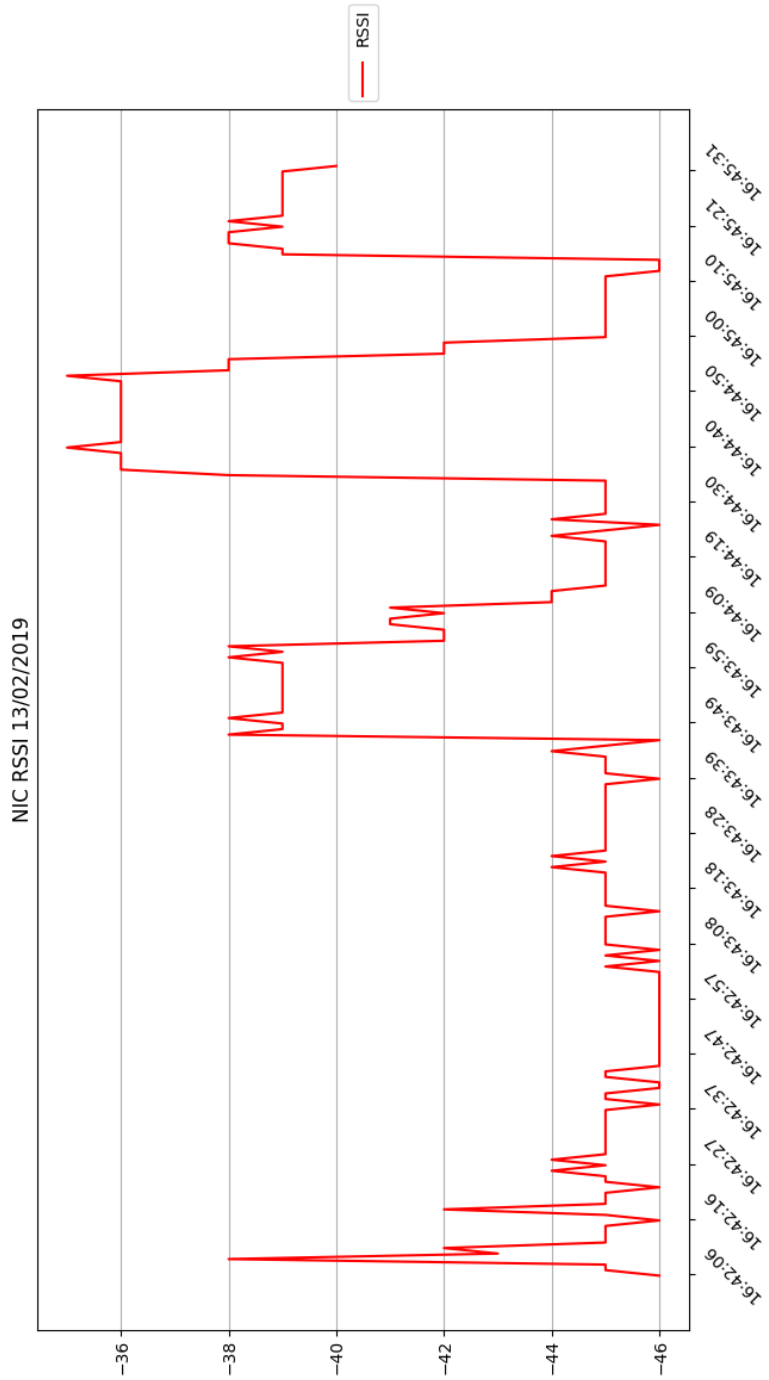


Figura 5.9: Rappresentazione grafica dei valori RSSI - NIC

Date	Quality
Feb 13, 2019 16:42:06	64
Feb 13, 2019 16:42:10	67
Feb 13, 2019 16:42:13	65
Feb 13, 2019 16:42:17	65
Feb 13, 2019 16:42:20	65
Feb 13, 2019 16:42:23	65
Feb 13, 2019 16:42:27	65
Feb 13, 2019 16:42:30	65
Feb 13, 2019 16:42:33	65
Feb 13, 2019 16:42:36	65
Feb 13, 2019 16:42:39	65
Feb 13, 2019 16:42:42	64
Feb 13, 2019 16:42:45	64
Feb 13, 2019 16:42:48	64
Feb 13, 2019 16:42:52	64
Feb 13, 2019 16:42:55	64
Feb 13, 2019 16:42:58	64
Feb 13, 2019 16:43:02	64
Feb 13, 2019 16:43:05	64
Feb 13, 2019 16:43:08	65
Feb 13, 2019 16:43:11	65
Feb 13, 2019 16:43:14	64
Feb 13, 2019 16:43:17	65
Feb 13, 2019 16:43:20	65
Feb 13, 2019 16:43:24	66
Feb 13, 2019 16:43:28	65
Feb 13, 2019 16:43:32	65
Feb 13, 2019 16:43:35	65
Feb 13, 2019 16:43:39	64
Feb 13, 2019 16:43:42	65
Feb 13, 2019 16:43:45	65
Feb 13, 2019 16:43:48	70
Feb 13, 2019 16:43:51	70
Feb 13, 2019 16:43:54	70
Feb 13, 2019 16:43:57	70

Date	Quality
Feb 13, 2019 16:44:00	70
Feb 13, 2019 16:44:03	70
Feb 13, 2019 16:44:06	68
Feb 13, 2019 16:44:09	68
Feb 13, 2019 16:44:12	66
Feb 13, 2019 16:44:15	65
Feb 13, 2019 16:44:18	65
Feb 13, 2019 16:44:21	65
Feb 13, 2019 16:44:24	65
Feb 13, 2019 16:44:27	65
Feb 13, 2019 16:44:31	65
Feb 13, 2019 16:44:34	65
Feb 13, 2019 16:44:37	70
Feb 13, 2019 16:44:40	70
Feb 13, 2019 16:44:43	70
Feb 13, 2019 16:44:46	70
Feb 13, 2019 16:44:49	70
Feb 13, 2019 16:44:52	70
Feb 13, 2019 16:44:55	70
Feb 13, 2019 16:44:58	68
Feb 13, 2019 16:45:02	65
Feb 13, 2019 16:45:06	65
Feb 13, 2019 16:45:09	65
Feb 13, 2019 16:45:13	64
Feb 13, 2019 16:45:16	70
Feb 13, 2019 16:45:19	70
Feb 13, 2019 16:45:22	70
Feb 13, 2019 16:45:25	70
Feb 13, 2019 16:45:29	70
Feb 13, 2019 16:45:32	70

Tabella 5.10: Valori di Link Quality - NIC

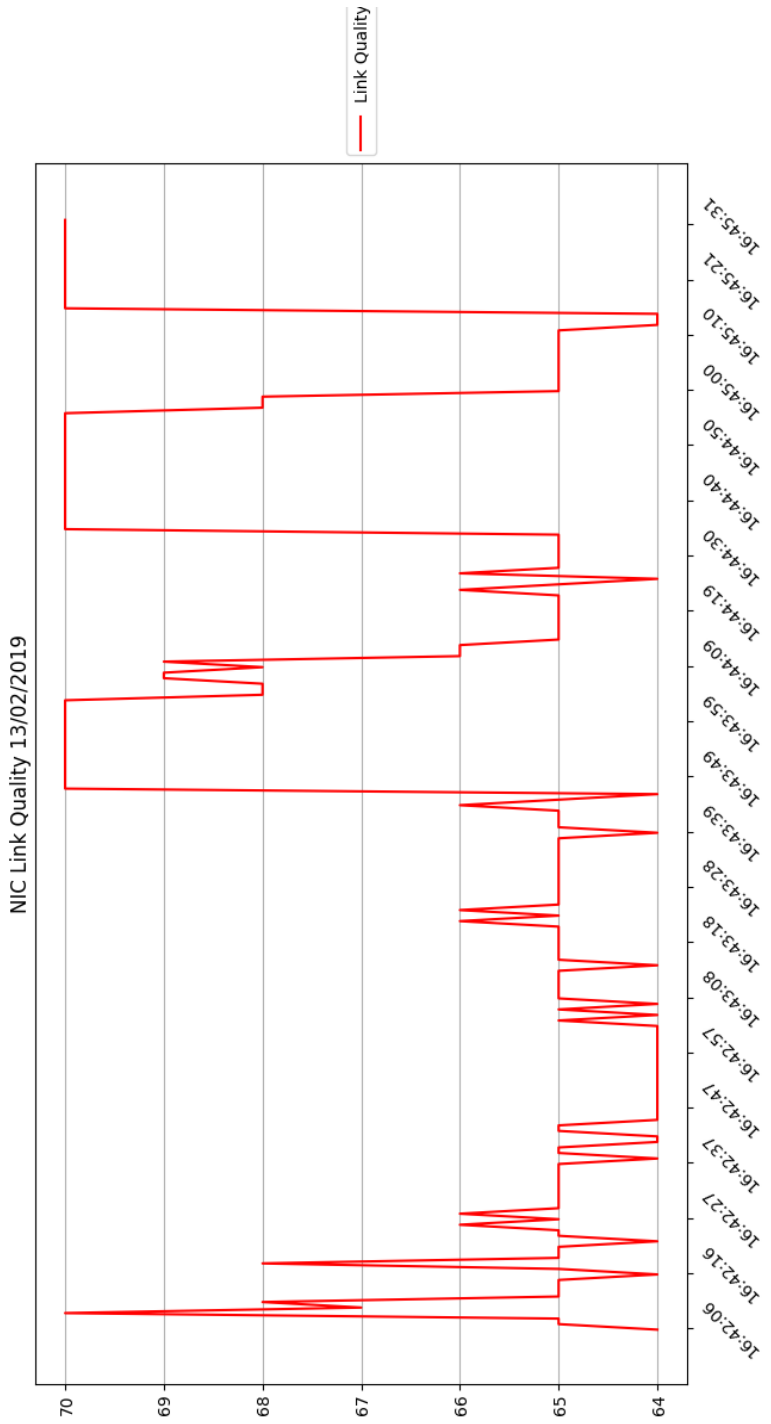


Figura 5.10: Rappresentazione grafica dei valori Link Quality - NIC

I valori contenuti all'interno delle tabelle e mostrati successivamente, all'interno dei grafici, permettono di enunciare le seguenti conclusioni:

- i valori di banda in UPLOAD risultano costanti e minori rispetto a quelli di DOWNLOAD. In particolare, è possibile osservare, nella zona centrale del grafico, dei valori di banda in DOWNLOAD minori rispetto al normale andamento del grafico: tale comportamento è dovuto all'utilizzo del codec video *H264*, il quale permette di occupare una porzione minore di banda;
- i valori di potenza RSSI sono compresi tra -33 dBm e -46 dBm. La prima parte del grafico illustra dei valori più o meno costanti, a differenza della seconda: ciò è dovuto alla distanza tra l'access point utilizzato e la stazione remota;
- i valori di Link Quality sono altamente correlati con quelli relativi alla potenza del segnale ricevuto: una buona qualità del canale di trasmissione fa sì che il valore di potenza del segnale ricevuto RSSI sia altrettanto elevato.

Scheda di rete Android Le informazioni acquisite dalla scheda di rete, presente all'interno dello smartphone, riguardano la potenza del segnale ricevuto RSSI e la velocità di trasmissione del canale. Questi valori vengono illustrati mediante opportune tabelle e grafici bidimensionali. L'interfaccia web, inoltre, è in grado di visualizzare ulteriori informazioni relative all'access point: SSID, BSSID, MAC. Tuttavia, si tratta di informazioni non necessarie per l'analisi della qualità di trasmissione delle informazioni. Di seguito, infatti, sono illustrate esclusivamente le informazioni utili per valutare la qualità della rete.

Date	RSSI
Feb 13, 2019 16:42:07	-72
Feb 13, 2019 16:42:12	-70
Feb 13, 2019 16:42:17	-71
Feb 13, 2019 16:42:22	-68
Feb 13, 2019 16:42:27	-66
Feb 13, 2019 16:42:32	-65
Feb 13, 2019 16:42:37	-70
Feb 13, 2019 16:42:42	-68
Feb 13, 2019 16:42:47	-69
Feb 13, 2019 16:42:52	-67
Feb 13, 2019 16:42:57	-68
Feb 13, 2019 16:43:02	-69
Feb 13, 2019 16:43:07	-71
Feb 13, 2019 16:43:12	-71
Feb 13, 2019 16:43:17	-71
Feb 13, 2019 16:43:22	-71
Feb 13, 2019 16:43:27	-70
Feb 13, 2019 16:43:32	-71
Feb 13, 2019 16:43:37	-69
Feb 13, 2019 16:43:42	-67
Feb 13, 2019 16:43:47	-62
Feb 13, 2019 16:43:52	-68
Feb 13, 2019 16:43:57	-70
Feb 13, 2019 16:44:02	-68
Feb 13, 2019 16:44:07	-67
Feb 13, 2019 16:44:12	-70
Feb 13, 2019 16:44:17	-67
Feb 13, 2019 16:44:22	-70
Feb 13, 2019 16:44:28	-66
Feb 13, 2019 16:44:32	-66
Feb 13, 2019 16:44:38	-65
Feb 13, 2019 16:44:42	-66
Feb 13, 2019 16:44:48	-63
Feb 13, 2019 16:44:52	-67
Feb 13, 2019 16:44:58	-63

Date	RSSI
Feb 13, 2019 16:45:08	-74
Feb 13, 2019 16:45:13	-66
Feb 13, 2019 16:45:18	-70
Feb 13, 2019 16:45:23	-70
Feb 13, 2019 16:45:28	-77

Tabella 5.11: Valori di RSSI - Android

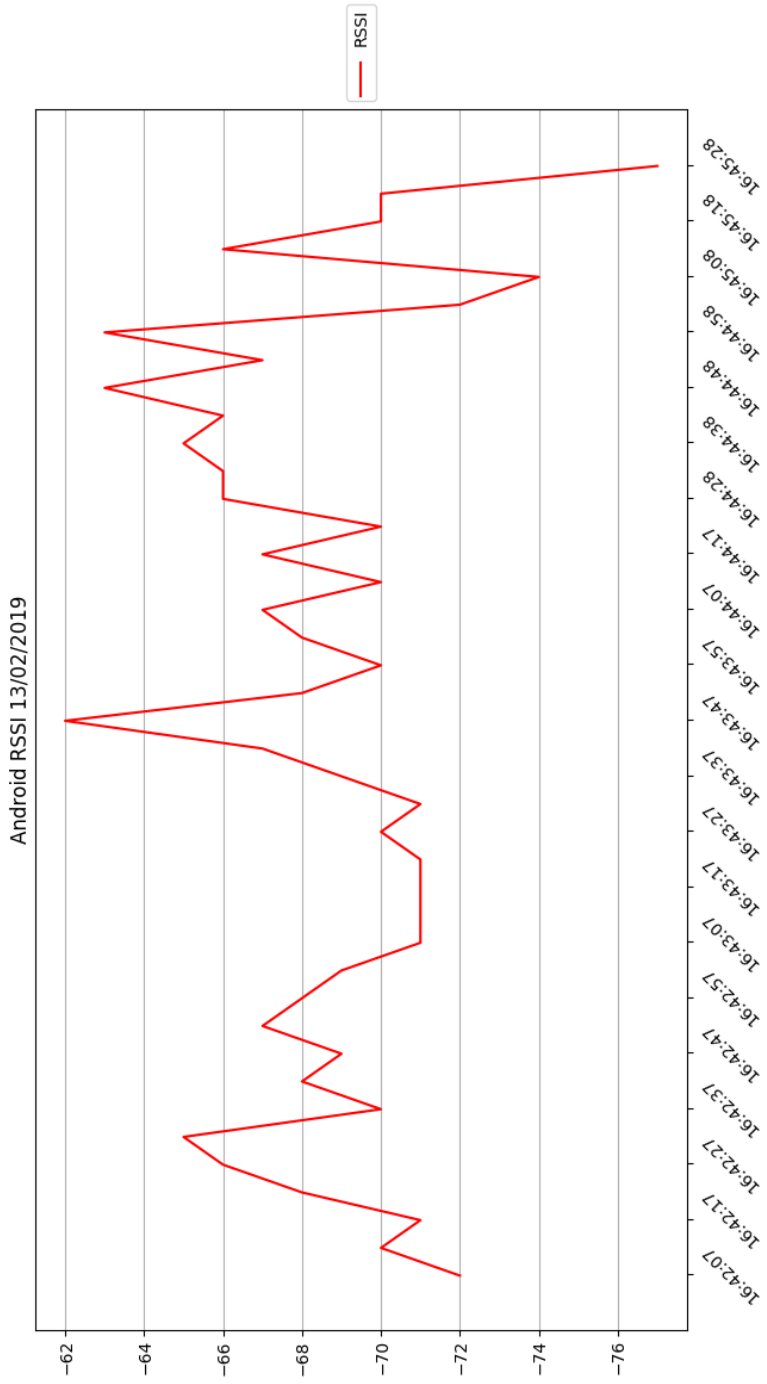


Figura 5.11: Rappresentazione grafica dei valori RSSI - Android

Date	Speed
Feb 13, 2019 16:42:07	42
Feb 13, 2019 16:42:12	42
Feb 13, 2019 16:42:17	42
Feb 13, 2019 16:42:22	50
Feb 13, 2019 16:42:27	56
Feb 13, 2019 16:42:32	53
Feb 13, 2019 16:42:37	50
Feb 13, 2019 16:42:42	14
Feb 13, 2019 16:42:47	56
Feb 13, 2019 16:42:52	25
Feb 13, 2019 16:42:57	25
Feb 13, 2019 16:43:02	58
Feb 13, 2019 16:43:07	70
Feb 13, 2019 16:43:12	56
Feb 13, 2019 16:43:17	42
Feb 13, 2019 16:43:22	50
Feb 13, 2019 16:43:27	47
Feb 13, 2019 16:43:32	53
Feb 13, 2019 16:43:37	56
Feb 13, 2019 16:43:42	60
Feb 13, 2019 16:43:47	70
Feb 13, 2019 16:43:52	42
Feb 13, 2019 16:43:57	30
Feb 13, 2019 16:44:02	21
Feb 13, 2019 16:44:07	43
Feb 13, 2019 16:44:12	51
Feb 13, 2019 16:44:17	65
Feb 13, 2019 16:44:22	70
Feb 13, 2019 16:44:28	55
Feb 13, 2019 16:44:32	47
Feb 13, 2019 16:44:38	51
Feb 13, 2019 16:44:42	34
Feb 13, 2019 16:44:48	56
Feb 13, 2019 16:44:52	62
Feb 13, 2019 16:44:58	48

Date	Speed
Feb 13, 2019 16:45:08	65
Feb 13, 2019 16:45:13	70
Feb 13, 2019 16:45:18	70
Feb 13, 2019 16:45:23	70
Feb 13, 2019 16:45:28	65

Tabella 5.12: Valori di Link Speed - Android

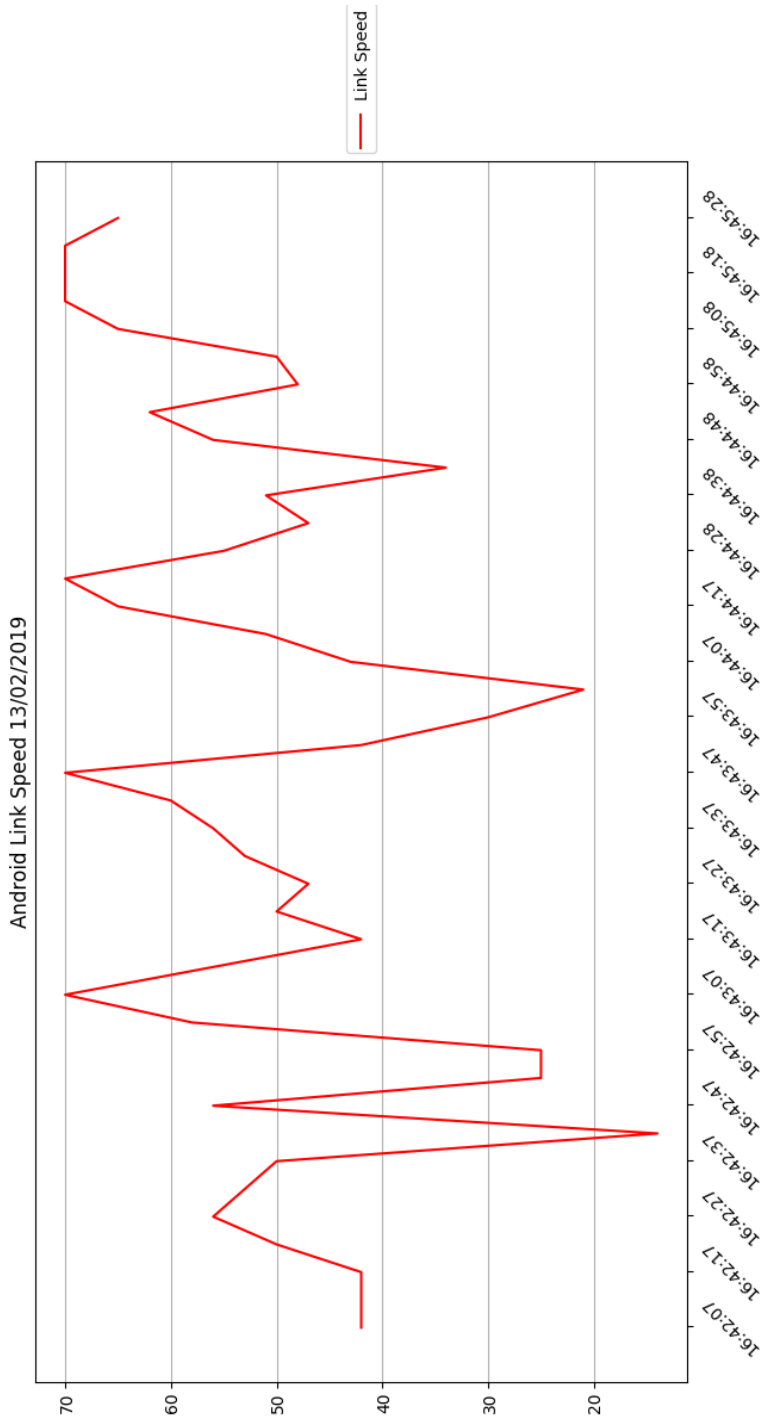


Figura 5.12: Rappresentazione grafica dei valori Link Speed - Android

Le informazioni, sottoscritte dall'applicazione web, sono state visualizzate all'interno di opportuni grafici bidimensionali dai quali è possibile trarre le dovute considerazioni finali:

- i dati raccolti riguardo la potenza del segnale in ricezione RSSI sono compresi tra -62 dBm e -77 dBm. Poiché il dispositivo android è collocato sul rover, i valori di potenza tendono a diminuire non appena la distanza con l'access point di riferimento aumenti;
- i valori della velocità di trasmissione del canale oscillano frequentemente. Tale comportamento è legato al canale di comunicazione instaurato tra il dispositivo android e l'access point; inoltre, i continui spostamenti del rover influenzano in maniera altalenante i valori acquisiti.

5.3.2 Informazioni sulla rete cellulare LTE

L'interfaccia web prevede una sezione dedicata alle informazioni relative alla rete cellulare LTE. Queste permettono di analizzare la qualità della suddetta rete mobile. I parametri prestazionali considerati sono: la potenza del segnale di ricezione RSRP, la qualità del segnale di riferimento ricevuto RSRQ e l'indicatore che riporta le informazioni sulla qualità del canale di comunicazione CQI. Tali informazioni non sono le uniche acquisite dal dispositivo android. Infatti, quest'ultimo è in grado di pubblicare informazioni sulla potenza del segnale LTE e sul rapporto segnale/rumore.

Le informazioni utili acquisite per l'analisi della rete mobile LTE sono contenute all'interno di tabelle e mostrate, di seguito, mediante l'utilizzo di grafici bidimensionali. È possibile, perciò, valutare qualitativamente la copertura del segnale LTE da parte dell'operatore mobile utilizzato.

Date	RSRP	RSRQ
Feb 13, 2019 16:42:07	-95	-17
Feb 13, 2019 16:42:12	-85	-20
Feb 13, 2019 16:42:17	-87	-18
Feb 13, 2019 16:42:22	-82	-16
Feb 13, 2019 16:42:27	-80	-12
Feb 13, 2019 16:42:32	-83	-14
Feb 13, 2019 16:42:37	-87	-14
Feb 13, 2019 16:42:42	-90	-19
Feb 13, 2019 16:42:47	-100	-22
Feb 13, 2019 16:42:52	-98	-19
Feb 13, 2019 16:42:57	-95	-17
Feb 13, 2019 16:43:02	-80	-12
Feb 13, 2019 16:43:07	-85	-13
Feb 13, 2019 16:43:12	-85	-15
Feb 13, 2019 16:43:17	-85	-15
Feb 13, 2019 16:43:22	-81	-10
Feb 13, 2019 16:43:27	-90	-13
Feb 13, 2019 16:43:32	-89	-13
Feb 13, 2019 16:43:37	-87	-13
Feb 13, 2019 16:43:42	-95	-17
Feb 13, 2019 16:43:47	-96	-17
Feb 13, 2019 16:43:52	-92	-16
Feb 13, 2019 16:43:57	-95	-16
Feb 13, 2019 16:44:02	-89	-14
Feb 13, 2019 16:44:07	-93	-15
Feb 13, 2019 16:44:12	-90	-12
Feb 13, 2019 16:44:17	-95	-15
Feb 13, 2019 16:44:22	-86	-10
Feb 13, 2019 16:44:28	-85	-10
Feb 13, 2019 16:44:32	-97	-20
Feb 13, 2019 16:44:38	-92	-18
Feb 13, 2019 16:44:42	-93	-18
Feb 13, 2019 16:44:48	-92	-18

Tabella 5.13: Valori di RSRP e RSRQ - LTE

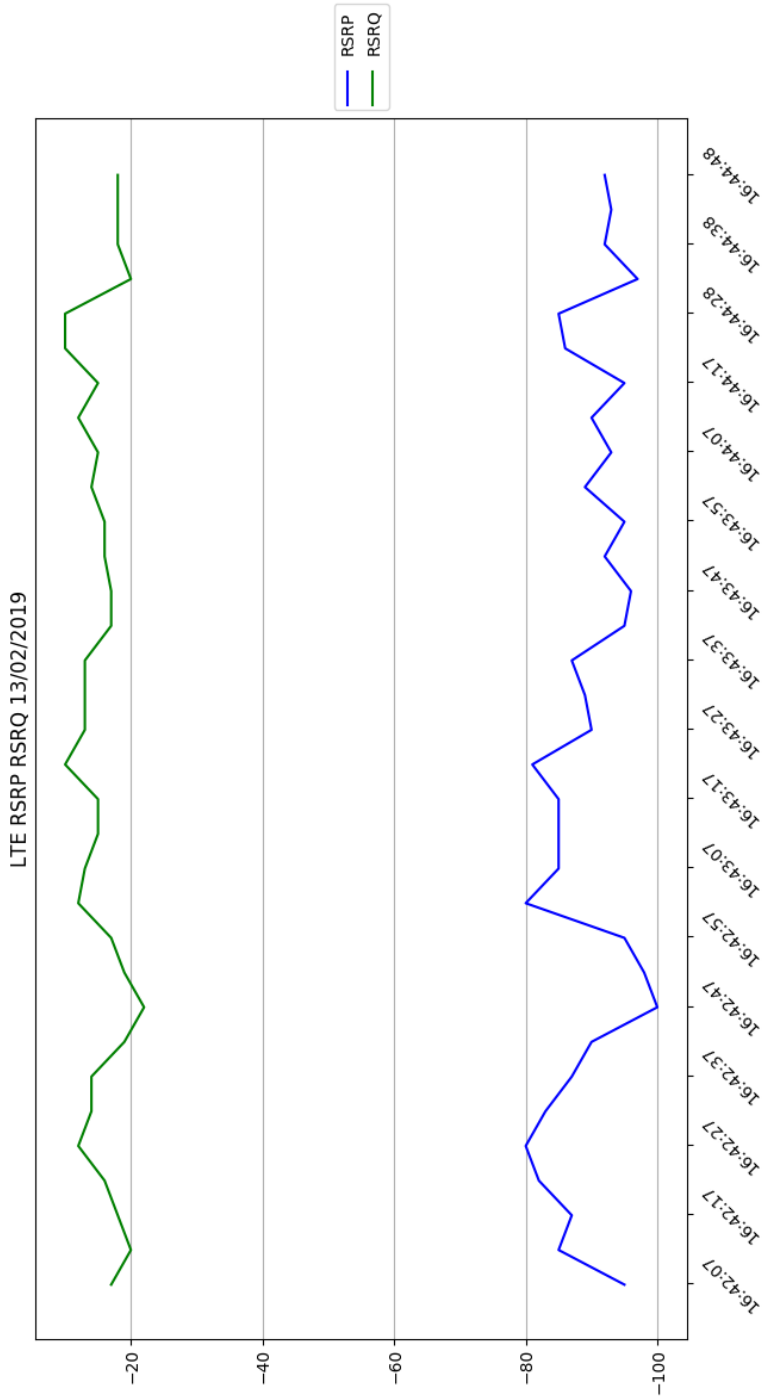


Figura 5.13: Rappresentazione grafica dei valori RSRP e RSRQ - LTE

Date	CQI
Feb 13, 2019 16:42:07	0
Feb 13, 2019 16:42:12	-1
Feb 13, 2019 16:42:17	8
Feb 13, 2019 16:42:22	4
Feb 13, 2019 16:42:27	2
Feb 13, 2019 16:42:32	-2
Feb 13, 2019 16:42:37	3
Feb 13, 2019 16:42:42	6
Feb 13, 2019 16:42:47	5
Feb 13, 2019 16:42:52	4
Feb 13, 2019 16:42:57	-2
Feb 13, 2019 16:43:02	10
Feb 13, 2019 16:43:07	2
Feb 13, 2019 16:43:12	2
Feb 13, 2019 16:43:17	2
Feb 13, 2019 16:43:22	-1
Feb 13, 2019 16:43:27	3
Feb 13, 2019 16:43:32	0
Feb 13, 2019 16:43:37	0
Feb 13, 2019 16:43:42	0
Feb 13, 2019 16:43:47	0
Feb 13, 2019 16:43:52	3
Feb 13, 2019 16:43:57	-2
Feb 13, 2019 16:44:02	0
Feb 13, 2019 16:44:07	0
Feb 13, 2019 16:44:12	0
Feb 13, 2019 16:44:17	-1
Feb 13, 2019 16:44:22	1
Feb 13, 2019 16:44:28	0
Feb 13, 2019 16:44:32	5
Feb 13, 2019 16:44:38	0
Feb 13, 2019 16:44:42	3
Feb 13, 2019 16:44:48	0

Tabella 5.14: Valori di CQI - LTE

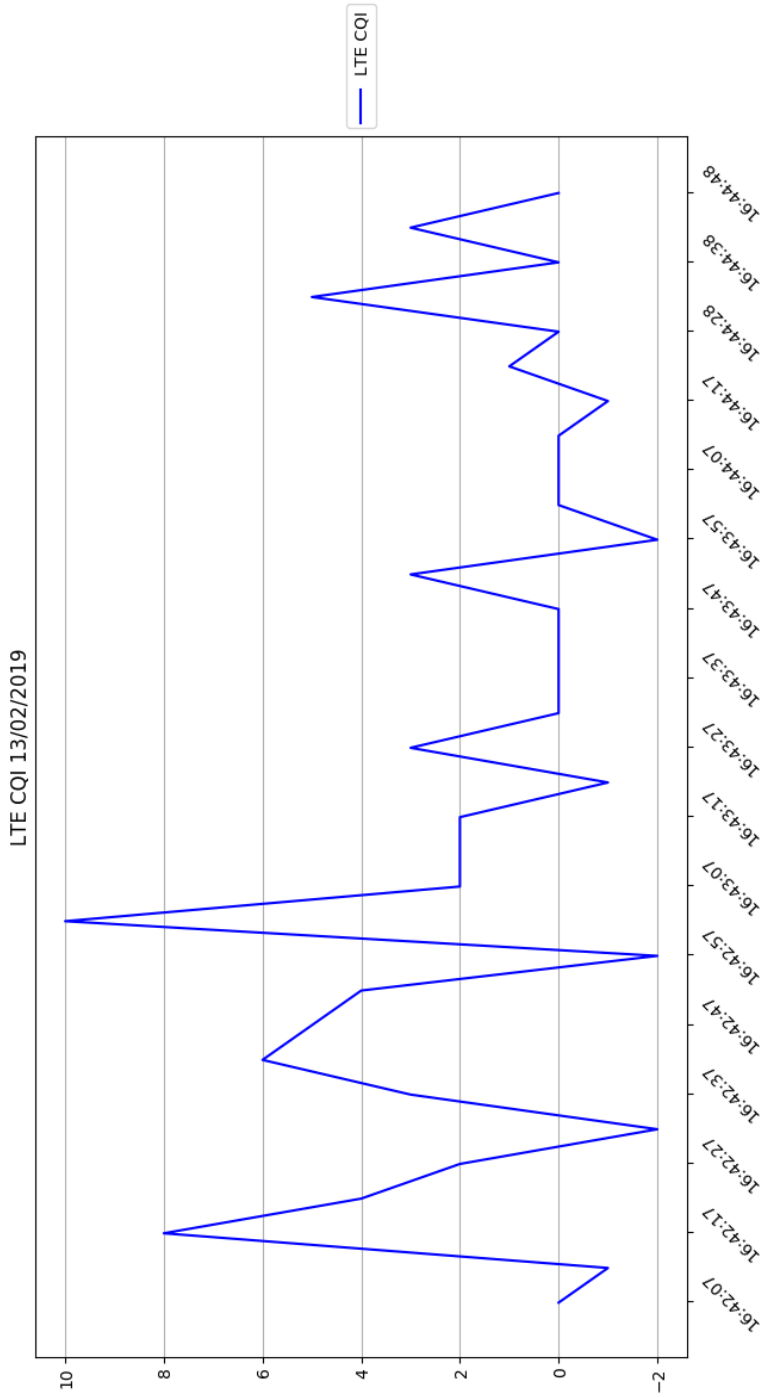


Figura 5.14: Rappresentazione grafica dei valori CQI - LTE

I dati mostrati in precedenza risultano utili per trarre le dovute considerazioni:

- la copertura della rete mobile LTE è presente in maniera costante. In particolare, i valori di RSRP sono tutti compresi tra -80 dBm e -100 dBm, mentre quelli di RSRQ tra -10 dB e -20 dB;
- l'indicatore sulla qualità del canale di comunicazione CQI assume valori prossimi allo 0, tranne in alcuni casi, dove si registrano valori solitamente compresi tra 8 e 2. Tali valori permettono di monitorare la modulazione con la quale i bit vengono trasmessi da parte della rete mobile LTE.

L'analisi effettuata sui sensori di rete e sulle informazioni LTE è stata utile per due ordini di motivi. Nel primo caso, i possibili disturbi di comunicazione dovuti alla distanza dall'access point di riferimento. In seconda istanza, invece, è stato possibile valutare la copertura LTE di un operatore mobile, analizzando opportuni parametri prestazionali. Inoltre, sono state condotte varie analisi riguardo la latenza di trasmissione di queste informazioni, focalizzandoci, in modo particolare, sulla trasmissione delle immagini multimediali. La qualità e la latenza di trasmissione dei frames trasmessi dall'applicazione android, come è stato possibile osservare, è stata condizionata dal canale di comunicazione e dalla potenza del segnale di ricezione. Confrontando le due tipologie di codec, i frames trasmessi secondo la codifica *H264* occupavano una banda minore rispetto alla codifica *jpeg*. Tuttavia, quest'ultima offriva un servizio migliore riguardo la qualità di visualizzazione dei singoli frames in reti condivise.

Capitolo 6

Conclusioni

La realizzazione dell'applicazione web per il monitoraggio e gestione dei dati multimediali è risultata utile in ambito applicativo. L'interfaccia web permette a ciascun utente, anche non esperto del framework ROS, di monitorare l'ambiente circostante. Infatti, è possibile analizzare una moltitudine di parametri, dai quali è possibile trarre una serie di conclusioni. L'utilizzo di un robot e di un dispositivo android, nel quale è installata l'applicazione *PIC4SeR Monitoring*, si è rivelato funzionale al corretto funzionamento di tutto il sistema.

Sfruttando l'interfaccia web, è stato possibile interagire da remoto con il robot, controllando i suoi movimenti e le molteplici potenzialità. L'utilizzo dei sensori presenti all'interno dello smartphone, collocato sul robot, si è rivelato utile alla gestione dei vari dati multimediali acquisiti dal dispositivo android medesimo. Tuttavia, ponendo particolare attenzione ad alcuni di essi, sono state dedotte interessanti considerazioni.

Un precipuo oggetto di studio è rappresentato dall'analisi della rete wireless, alla quale i tre dispositivi (personal computer, dispositivo android, robot) sono connessi. Di fatto, l'utilizzo di un unico access point permette di monitorare la zona solo entro certi limiti. A tal proposito, è possibile osservare, dai grafici presenti nell'interfaccia web, una diminuzione progressiva dei valori di potenza del segnale ricevuto RSSI e del valore della qualità del link. Quest'attenuazione è dovuta, principalmente, all'aumentare della distanza tra i dispositivi interessati e l'access point di riferimento. I grafici, attraverso cui sono state dedotte le dovute considerazioni, riguardano sia quelli relativi alla scheda di rete NIC che quelli correlati alla scheda di rete presente nel dispositivo android.

Analisi ulteriori riguardano specificatamente la rete mobile LTE. In questo caso, è stato possibile osservare l'andamento dei valori RSRP e RSRQ: questi risultano piuttosto costanti, confermando una buona copertura della rete mobile LTE nella zona di monitoraggio. Inoltre, i valori mostrati sul grafico riguardo la qualità del canale, identificata dal parametro prestazionale CQI, permettono di individuare il tipo di modulazione utilizzata per la trasmissione delle informazioni da parte del dispositivo android.

Dai risultati ottenuti dalle varie misurazioni effettuate, si evince l'importanza della tecnica utilizzata per il monitoraggio delle qualità prestazionali a supporto di un qualsiasi contesto mobile. Pertanto, l'obiettivo precipuo è proprio l'espansione degli scenari applicativi, per i quali risulta utile l'utilizzo di tale applicazione. La possibilità di poter monitorare da remoto l'ambiente, nei suoi dettagli associati ai sensori analizzati, è resa concreta dall'utilizzo di pochi mezzi a disposizione. Nei contesti applicativi real time, l'utilizzo di più access point sarebbe, di certo, una delle possibili soluzioni che consentirebbe di monitorare zone più ampie. L'utilizzo di un unico access point, purtroppo, limita il raggio di azione di utilizzo delle soluzioni sviluppate. Minore è la potenza del segnale di ricezione delle informazioni, maggiore sarà il ritardo di trasmissione dei dati, con il rischio di perderne qualcuno.

Le analisi effettuate mediante l'utilizzo di un rover sono state utili per analizzare e valutare l'usabilità dell'interfaccia web. Tuttavia, si pone l'obiettivo di testare tale interfaccia web anche in altri ambiti applicativi, quali: dispositivi in volo. L'utilizzo di aeromobili a pilotaggio remoto, sui quali è possibile collocare un qualsiasi dispositivo android capace di pubblicare specifiche informazioni, sarebbe uno scenario interessante dal punto di vista applicativo. I parametri prestazionali osservati mediante il drone potrebbero assumere valori differenti. Certamente, si suppone che la qualità del canale di comunicazione tra la stazione remota rappresentata dal personal computer e il dispositivo mobile posto sul drone sia inferiore. La distanza tra i dispositivi (stazione base e drone) influirà, inoltre, negativamente sulla trasmissione dei dati e sulla latenza delle immagini acquisite.

Il lavoro di tesi prevede sviluppi futuri su diversi aspetti, poiché l'applicazione web è stata realizzata per inglobare vari campi applicativi. Anzitutto, offrendo un'interfaccia in grado di monitorare molteplici sensori, è possibile spaziare da un ambito applicativo ad un altro. Sicuramente, è possibile monitorare le qualità prestazionali delle reti cellulari, applicando le opportune modifiche. Il monitoraggio delle reti wireless, inoltre, permette di controllare la potenza di trasmissione e ricezione di queste ultime. L'utilizzo dei sensori

d'ambiente può essere interessante per effettuare misurazioni atmosferiche, le quali si mostrano utili nel campo dell'agricoltura di precisione.

Dal punto di vista progettuale, invece, è possibile sfruttare le potenzialità offerte dall'interfaccia web per il miglioramento di tale servizio. In particolare, sarebbe interessante l'interazione di tale applicazione web con più dispositivi android contemporaneamente, in modo tale da analizzare i vari sensori presenti in più dispositivi android nello stesso intervallo di tempo. Inoltre, la realizzazione di una tecnologia *back-end* renderebbe tale applicazione accessibile da parte di qualsiasi utente, mediante l'utilizzo di un indirizzo IP pubblico, utilizzato appositamente per l'applicativo web. Di fatto, sfruttando un server remoto in cui è presente l'architettura ROS, qualsiasi utente potrebbe accedere al servizio web da remoto. Inoltre, i dati potrebbero essere memorizzati all'interno di un database, al quale accedere immettendo opportune credenziali amministrative.

È possibile, inoltre, integrare tale interfaccia web sfruttando ulteriori sensori messi a disposizione dal mondo dell'Internet of Things. La robotica di servizio, pertanto, risulta fondamentale per la realizzazione ed integrazione delle applicazioni pratiche, tra le quali assumono sempre maggior rilievo le applicazioni web.

Bibliografia

[1] Howard A., Koenig N., *Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator*, Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, Robotics Research Labs, University of Southern California, Los Angeles, USA, September 28 - October 2, 2004, pp. 2149-2154

[2] Crick C., DuHadway C., Jay G., Jenkins O. C., Osentoski S., Pitzer B., *Robots as web services: Reproducible experimentation and application development using rosjs*, 2011 IEEE International Conference on Robotics and Automation, Shanghai International Conference Center, Shanghai, China, May 9-13, 2011, pp. 6078-6083

[3] Koubaa A., *ROS As a Service: Web Services for Robot Operating System*, Journal of Software Engineering for Robotics, Prince Sultan University, Saudi Arabia, December 2015, pp. 123-136

[4] Chernova S., Jenkins O. C., Kammerl J., Lee J., Lu D. V., Osentoski S., Toris R., Wills M., *Robot Web Tools: Efficient Messaging for Cloud Robotics*, National Science Foundation, Office of Naval Research, NASA, Ministry of Trade, Industry & Energy (MOTIE, KOREA) Robot Industry Original Technology Development Project

[5] Crick C., Jenkins O. C., Jay G., Osentoski S, Pitzer B., *Rosbridge: ROS for Non-ROS Users*, Brown University, Providence RI, Robert Bosch LLC, Mountain View CA

[6] Lee J., *Web Applications for Robots using rosbridge*, Computer Science Department, Brown University

[7] Luo D., Ma H., Zhou N., *Web-based Mobile Robot Control and Monitoring*, CIS 693 Autonomous Intelligent Robotics Course, Washkewicz College of Engineering, Cleveland State University, USA, May 04, 2018

[8] de Vries M., *WebMap - A ROS web interface*, November 9, 2012

[9] Simpson O., Sun Y., *LTE RSRP, RSRQ, RSSNR and local topography profile dara for RF propagation planning and network optimization in an urban propagation environment*, ELSEVIER, School of Engineering and Technology, University of Hertfordshire, Hatfield AL10 AB, United Kingdom, 31 August 2018

[10] Clearpath Robotics Inc., *JACKAL UNMANNED GROUND VEHICLE USER MANUAL*, Kitchener, Ontario, Canada, Rev. 1.1.7

[11] Clearpath Robotics Inc., *JACKAL UNMANNED GROUND VEHICLE TECHNICAL SPECIFICATIONS*, Kitchener, Ontario, Canada, Rev. 1.1.7

[12] Nenci F., Spinello L., Stachniss C., *Effective Compression of Range Data Streams for Remote Robot Operations using H.264*, 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IRSO 2014), CHICAGO, IL, USA, September 14-18, 2014, pp. 3794-3799

[13] Kautz J., Pece F., Weyrich T., *Adapting Standard Video Codecs for Depth Streaming*, Joint Virtual Reality Conference of EuroVR - EGVE (2011), Department of Computer Science, University College London, UK

[14] Tartaggia G., *Localizzazione e identificazione di persone da robot mobile in ambienti sconosciuti*, Tesi di Laurea Magistrale in Ingegneria Informatica, Università degli Studi di Padova, A.A. 2014/2015

Sitografia

[1] ROS-Industrial is an open-source project that extends the advanced capabilities of ROS software to manufacturing
https://rosindustrial.org/

[2] Robotiko - Robot Operating System
https://www.robotiko.it/robot-operating-system-ros

[3] Dynamic Animation and Robotics Toolkit
https://dartsim.github.io/

[4] Gazebo Robot Simulator
http://gazebosim.org/

[5] Open Dynamics Engine
https://www.ode.org/

[6] ROS Concept
http://wiki.ros.org/ROS/Concepts

[7] Simbody: Multibody Physics API
https://simtk.org/projects/symbody

[8] Turtlebot Gazebo
http://wiki.ros-org/turtlebot_gazebo

[9] How To: Setup Turtlebot Simulator in ROS with Gazebo
http://www.sauravag.com/2016/10/how-to-setup-turtlebot-simulator-in-ros-with-gazebo/

[10] Launching TurtleBot simulator in Gazebo
https://subscription.packtpub.com/book/hardware_and_creative/9781782175193/3/ch03lvl1sec24/launching-turtlebot-simulator-in-gazebo

[11] Simulating Jackal
http://www.clearpathrobotics.com/assets/guides/jackal/simulation.html

[12] Clearpath Jackal Features
https://www.clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/

[13] Ubuntu install of ROS Kinetic
http://wiki.ros.org/kinetic/Installation/Ubuntu

[14] Introducing rqt tools
http://wiki.ros.org/rqt
https://subscription.packtpub.com/book/hardware_and_creative/9781782175193/3/ch03lvl1sec29/introducing-rqt-tools

[15] RQT Image View
http://wiki.ros.org/rqt_image_view

[16] Tutorial Running ROSBridge
http://wiki.ros.org/rosbridge_suite/Tutorials/RunningRosbridge
https://github.com/RobotWebTools/rosbridge_suite/blob/develop/rosbridge_server/launch/rosbridge_websocket.launch

[17] ROS Web Video Server
http://wiki.ros.org/web_video_server
https://github.com/RobotWebTools/web_video_server

[18] ROS Image Transport
http://wiki.ros.org/image_transport

[19] Roslibjs
http://wiki.ros.org/roslibjs
http://robotwebtools.org/jsdoc/roslibjs/current/
https://github.com/RobotWebTools/roslibjs

[20] Ros2djs
http://wiki.ros.org/ros2djs/
http://robotwebtools.org/jsdoc/ros2djs/current/
https://github.com/RobotWebTools/ros2djs

[21] Ros3djs
http://wiki.ros.org/ros3djs/
http://robotwebtools.org/jsdoc/ros3djs/current/
https://github.com/RobotWebTools/ros3djs

[22] Keyboardteleopjs
http://wiki.ros.org/keyboardteleopjs/
http://robotwebtools.org/jsdoc/keyboardteleopjs/
current/Teleop.js.html

[23] ROS Environment Variables
http://wiki.ros.org/ROS/EnvironmentVariables

[24] Creazione di un ROS Package
http://wiki.ros.org/it/ROS/Tutorials/
CreatingPackage

[25] roscreate
http://wiki.ros.org/roscreate

[26] catkin
http://wiki.ros.org/catkin

[27] roscpp
http://wiki.ros.org/roscpp

[28] rospy
http://wiki.ros.org/rospy

[29] std_msgs
http://wiki.ros.org/std_msgs

[30] How to install FFmpeg on Ubuntu 16.04
https://www.ffmpeg.org/
https://linuxize.com/post/how-to-install-ffmpeg-
on-ubuntu-18-04/

[31] Visual Studio Code
https://code.visualstudio.com/

[32] Mapbox Web Applications
https://docs.mapbox.com/help/how-mapbox-works/
web-apps/

[33] std_msgs/Header Message
http://docs.ros.org/kinetic/api/std_msgs/html/msg/Header.html

[34] sensor_msgs/NavsatFix Message
http://docs.ros.org/kinetic/api/sensor_msgs/html/msg/NavSatFix.html

[35] sensor_msgs/Imu Message
http://docs.ros.org/kinetic/api/sensor_msgs/html/msg/Imu.html

[36] sensor_msgs/Illuminance Message
http://docs.ros.org/kinetic/api/sensor_msgs/html/msg/Illuminance.html

[37] sensor_msgs/MagneticField Message
http://docs.ros.org/kinetic/api/sensor_msgs/html/msg/MagneticField.html

[38] sensor_msgs/Temperature Message
http://docs.ros.org/kinetic/api/sensor_msgs/html/msg/Temperature.html

[39] sensor_msgs/FluidPressure Message
http://docs.ros.org/kinetic/api/sensor_msgs/html/msg/FluidPressure.html

[40] CellSignalStrengthLte
https://developer.android.com/reference/android/telephony/CellSignalStrengthLte.html

[41] RSRP and RSRQ
https://wiki.teltonika.lt/view/RSRP_and_RSRQ

[42] LTE Quick Reference
http://www.sharetechnote.com/html/Handbook_LTE_CQI.html

Ringraziamenti

Gli anni universitari al Politecnico di Torino sono stati un momento indelebile i quali mi hanno permesso di confrontarmi e di accrescere il mio bagaglio culturale relativo al percorso di studi intrapreso. È doveroso ringraziare il Prof. Ing. Enrico Masala che mi ha seguito per lo sviluppo della tesi in maniera costante, fornendomi preziosi suggerimenti durante il percorso di lavoro giunto alla stesura di questo elaborato finale.

Vorrei ringraziare tutti i miei amici che mi hanno sempre sostenuto durante questo percorso universitario. Ringrazio gli amici di una vita, i quali mi hanno sempre affiancato nelle scelte intraprese. Ringrazio i miei colleghi universitari con i quali ho trascorso due anni fantastici, con i quali ho avuto la possibilità di confrontarmi e trascorrere molto tempo insieme, anche lontano dallo studio. Ringrazio i miei "fratelli" Cosimo, Marco e Mirko con i quali ho trascorso due anni indimenticabili a Torino, ricordando i vecchi tempi passati, ma augurandoci di rimanere sempre vicini. Un doveroso ringraziamento va a Salvatore, persona con la quale ho avuto modo di confidarmi su alcune decisioni e perplessità.

Ringrazio i miei genitori senza i quali non sarei quello che sono diventato: una persona matura. Li ringrazio per il sostegno ricevuto in questi anni di vita, fondamentali nei momenti di maggiore bisogno. Nonostante la lontananza, mi sono sempre stati vicini. Ringrazio mio fratello Giulio, con il quale ho avuto modo di confrontarmi. Grazie per tutti i momenti trascorsi insieme, siete la mia forza.

Un importante ringraziamento lo devo alla mia ragazza Noemi che è stata in grado di sostenermi nei momenti difficili, in questi due anni trascorsi lontani. La ringrazio per la pazienza nei miei confronti e per l'amore che mi ha dimostrato in questi anni. Grazie di essermi sempre vicino.

Federico Barone