Simulation of a Humanoid Robot in Gazebo Environment

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'Automazione

Author: **Giovanni Porcellato**

Student ID: 10745779
Advisor: Prof. Matteo Matteucci
Co-advisors: Simone Mentasti
Academic Year: 2021-22

Abstract

Here goes the Abstract in English of your thesis followed by a list of keywords. The Abstract is a concise summary of the content of the thesis (single page of text) and a guide to the most important contributions included in your thesis. The Abstract is the very last thing you write. It should be a self-contained text and should be clear to someone who hasn't (yet) read the whole manuscript. The Abstract should contain the answers to the main scientific questions that have been addressed in your thesis. It needs to summarize the adopted motivations and the adopted methodological approach as well as the findings of your work and their relevance and impact. The Abstract is the part appearing in the record of your thesis inside POLITesi, the Digital Archive of PhD and Master Theses (Laurea Magistrale) of Politecnico di Milano. The Abstract will be followed by a list of four to six keywords. Keywords are a tool to help indexers and search engines to find relevant documents. To be relevant and effective, keywords must be chosen carefully. They should represent the content of your work and be specific to your field or sub-field. Keywords may be a single word or two to four words.

Keywords: here, the keywords, of your thesis

Abstract in lingua italiana

Qui va l'Abstract in lingua italiana della tesi seguito dalla lista di parole chiave.

Parole chiave: qui, vanno, le parole chiave, della tesi

Contents

1 | Introduction

Now far from being science fiction, robots are little by little entering our world. The tendency to replace humans with machines has existed for centuries, but it is only during the 20th century that technology has enabled us to achieve astonishing results.
Hence, the robots we know today come in different forms: from robots with human form, to mechanical arms, to robotic heads. In each case, there was extensive research and development before arriving at the results we can enjoy today. Given the initial instability of robotic systems and the danger and cost, the creation of these devices went hand in hand with the development of special simulators.
It is indeed very inconvenient to repeat tests on hardware, for the reasons mentioned above. It is precisely in this circumstance that the need arises to simulate in a virtual environment the movements and actions that the robot would then have to repeat in the real world. It is therefore necessary to set up a testing procedure that has precise specifications and is reliable and repeatable over time.
In particular, the simulated behaviour of the robot must be consistent with the measurements and physical specifications of the robot in order to have a reliable simulation.
From an automation engineering point of view, the main objective is to eliminate human intervention even on the testing procedure, in order to have a semi-autonomous testing software, which returns data that can be easily analysed both for internal development and for supply to third parties.
The operator must in any case monitor the progress of the test in order to obtain useful information on anomalous behaviour.

This work proposes a tool for testing the navigation of a humanoid robot in a simulated environment, offering a faithful and secure solution and obviating the problematic testing on hardware.

In addition, lateral solutions are presented to problems that arose in the testing phase, such as the denoising of the image received from the robot‘s video terminal.

1.1. Oversonic Robotics

This project has been realized as an application for Oversonic Robotics.
Oversonic Robotics is an Italian robotics start-up founded in 2020 in Besana in Brianza (MB) that wants to provide companies with intelligent systems that can assist humans in the most psychologically and physically demanding and exhausting jobs, thus enabling people to devote themselves to tasks that more effectively enhance intelligence. Their first project, Robee (Figure 1.1), is an autonomous humanoid robot, 1.75 m tall with a weight of around 75 Kg. He operationally replicates the mechanical structure of the human body, with 36 movable joints and a complete set of sensors. This enables him to see and navigate the surrounding space. The interaction is managed via a voice interface, capable of carrying out a normal conversation. He has arms and hands that allow him to cover all kinds of tasks. These include the simplest gestures such as pointing or counting, to a solid grip for handling objects. For greater awareness of his surroundings and for better communication with operators, Robee is equipped with a variety of cameras that use facial recognition and object detection.

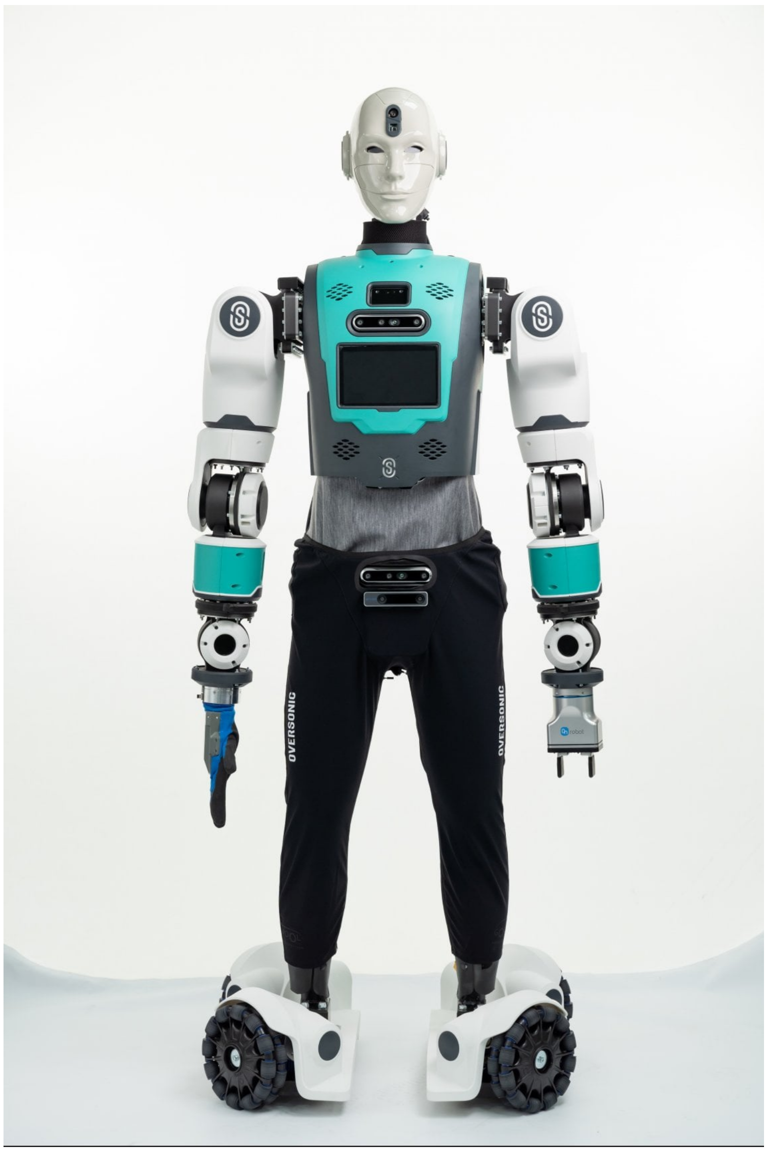


Figure 1.1: "Robee", the humanoid robot developed by Oversonic Robotics Repubblica [3]

1.2. Contribution

The main contributions of this work are as follows:

- Development of a simulated environment to test the robot's new navigation features.
- Development of a module to test the navigation performance of the robot.
- Implementation of a point-cloud filter in order to improve obstacle handling and consequently navigation performance in critical brightness scenarios

1.3. Document Structure

The content is divided into two large sections: the first refers to the **Theoretical and Technological Background**, where the state of the art and the technologies used are introduced, in order to fully understand the system; the second one contains the **Contribution** of the thesis and reports the architecture of the system, its implementation and results. The thesis is composed of six chapters, below we list the content of each of them to give the reader an overview of the work done.

- Chapter 2 provides an overview of the software platform ROS, Robot Operating System, explaining its characteristic and philosophy that highlight why it is used as common platform to manage robots' operations, tasks, motions.
- Chapter 3 first introduces robots providing a brief historical overview. It is then described Oversonic robot, Robee, describing its components, both software and hardware.
- Chapter 4 aims to introduce the literature survey of the various techniques used for mobile robot navigation. Navigation and obstacle avoidance are one of the fundamental problems in mobile robotics, here are described two type of control global path planning and local motion control.
- Chapter 5 represents the main work of this thesis. It introduces first Gazebo simulator, afterwards it goes step by step through the building of the sim. robot and environment. It then encompasses the measurement module implemented. Results and issues are presented.
- Chapter 6 introduces point-cloud. This section proposes to solve an issue encountered while testing on the physical robot.

Theoretical background

From now on some basic knowledge needed to comprehend the reasoning of this work's study object is described and explained. This section consists of:

- **Chapter 2:** ROS
- **Chapter 3:** Robot

2 | ROS

2.1. Introduction

In the field of robotics, platforms are of increasing importance. A platform is divided into a software platform and a hardware platform. There are a variety of features that make up robot software platforms, including low-level device control, SLAM (Simultaneous Localization and Mapping), navigation, manipulation, recognition of objects or people, sensing and package management, debugging and development tools, which are mostly used in the industrial sector where robot software platforms are currently primarily employed. Robot hardware platforms not only study platforms such as mobile robots, drones, and humanoids, but also commercial products. Hence, robot researchers from around the world are collaborating to discover a platform that is intuitive and open source. The most popular robot software platform is ROS, that means Robot Operating System. ROS, the Robot Operating System, is an open source framework to manage robots' operations, tasks, motions, and other things. ROS is intended to serve as a software platform for those who build and use robots daily, but at the same time for people who are starting to use robots no long ago. This common platform allows newcomers to be increasingly inclined to read more and more and it is very easy to use. The design of the ROS platform allows the use of code and information shared by other programmers, which means you do not have to write all code in order to move the robots.

2.2. History of ROS

"In May 2007, ROS was started by borrowing the early opensource robotic software frameworks including switchyard, which is developed by Dr. Morgan Quigley by the Stanford Artificial Intelligence Laboratory in support of the Stanford AI Robot STAIR (Stanford AI Robot) project. Dr. Morgan Quigley is one of the founders and software development manager of Open Robotics (formerly the Open Source Robotics Foundation, OSRF), which is responsible for the development and management of ROS. Switchyard is a program created for the development of artificial intelligence robots used in the AI lab's projects at the time and it is the predecessor of ROS. In addition, Dr. Brian Gerkey, the developer of the Player/Stage Project and 2D Stage simulator, later influence the growth of 3D simulator Gazebo, which was developed since 2000 and has had a major impact on ROS's networking program. He is the co-founder of Open Robotics. In November 2007, U.S. robot company Willow Garage succeeded the development of ROS. Willow Garage is a well-known company in the field of personal robots and service robots." [2]
Robots are computer-controlled electromechanical devices. First dedicated robot programming languages arose in the 1970's. There were robot-centric data types and some robot function libraries. They did not allow neither hardware abstractio, nor multi-robot interaction nor integrated simulation. There did not exist code reuse or standardization. The efforts to build robot programming system continued through 80's, 90's and especially in the 2000's were there was a high push to standardize robot components, their interfaces and basic functions.
Hence, ROS as it is known today was initially developed in 2007 at the Standford Artificial Intelligence Laboratory. Since 2013 it is managed by OSRF and nowadays it is used by many robots, universities and companies, becoming the de facto standard for robot programming.

Figure 2.2: Willow Garage PR2 robot
]

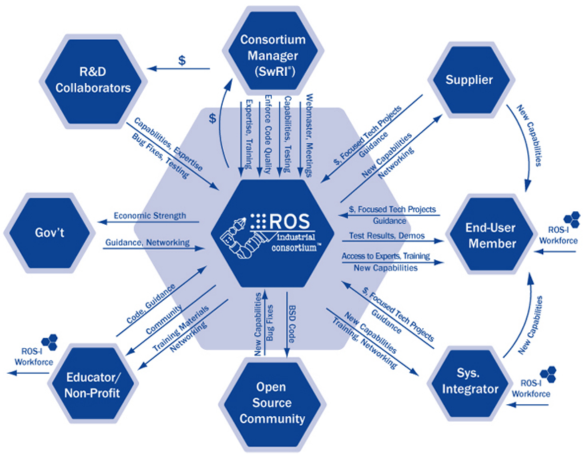


Figure 2.1: ROS Industrial Consortium

2.3. Meta-Operating System

ROS is an open source, meta operating framework for robots, hence it is nothing else than a middleware.
A middleware can be defined as a piece of software that gives an extra level of abstraction to the developer through a layer between the operating system and the applications. It basically sits in the middle of software components and facilitates their interaction.
Its purpose is to provide an abstraction model for functions and at the same time provides the low-level implementation. Every middleware must provide:

- Portability: common programming model regardless the programming language and system architecture
- Complexity management: low-level aspects are managed by libraries and drivers inside the middleware itslef
- Reliability: middleware allows robot developer to discard low level details
- abstraction from sensors/actuators hardware;
- communication protocol for data transport

As a result, they play an essential role in the development of complex applications that rely on a number of hardware and software tools.
While they are still under active development, they are not yet capable of providing a complete set of functions for general purpose robots.

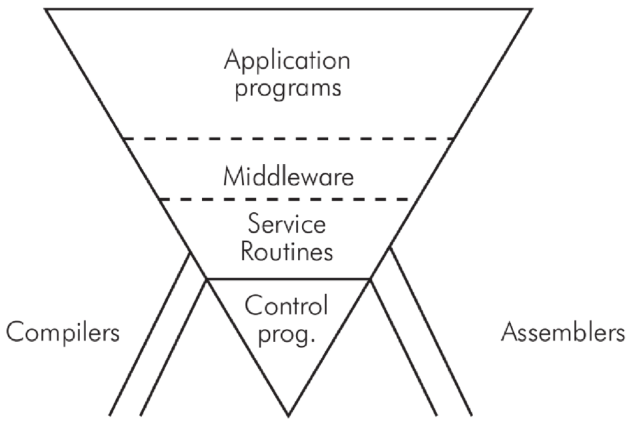


Figure 2.3: Meta Operating System

Several robotic middlewares have been proposed in the past years (OROCOS, ORCA, YARP, BRICS etc.) and eventually came ROS.

2.3.1. Phylosophy of ROS

The following paragraphs describe some philosophical aspects of ROS:

- *Peer to peer*: ROS systems consist of a small number of computer programs that are linked to one another and continuously exchange messages. These messages travel directly from one program to another. Although this makes the system more complex, the result is a system that balances better as the number of data increases.

- *Distributed*: Programs can be run on multiple computers and comunicate over the network.
- *Multilingual*: ROS chose a multilingual approach. ROS software modules can be written in any language for which a client library has been written. At the time of writing, client libraries exist for C++, Python, LISP, Java, JavaScript, MATLAB and others.

- *Thin*: the ROS conventions encourage contributors to create standalone libraries and then wrap those libraries, so that they can send and receive messages to and from other ROS modules. This extra layer is proposed to allow the reuse of software outside of ROS for other applications, and it greatly simplifies the creation of automated tests using standard continuous integration tools.

- *Free and open source*: the core of ROS is released under the permissive BSD license, which allows both commercial and non-commercial use. ROS foresees data exchange between modules using inter-process communication (IPC), which means that systems built using ROS can have fine-grained licensing of their various components.

2.4. ROS Architecture

ROS is based on a graph architecture where processing takes place in nodes, which communicate with each other by exchanging messages asynchronously through the use of topics to which they can subscribe to and/or on which they can publish, and synchronously with the calling of services, similar to RPCs. Structurally, ROS is developed on 3 conceptual levels, File-system Level, Computational Level and Community Level, of which we are going to examine the constituent elements and their role in the architecture.

2.4.1. File-system Level

The Filesystem Level includes all resources used in ROS, in particular

- Packages
- Metapackages
- Manifest
- Message types
- Service types

Packages
Packages are the main structure for organising ROS Wiki [9]. The processes, libraries, configuration files, datasets, and all the files used by the system at runtime are stored in these files. They are the structure that can be found within a ROS-based system. At the filesystem level, the package is represented by a directory. The structure within it includes some subfolders to manage the elements In order to facilitate its development, in particular:

- *include/packagename*: C++ include headers (make sure to export in the CMakeLists.txt)

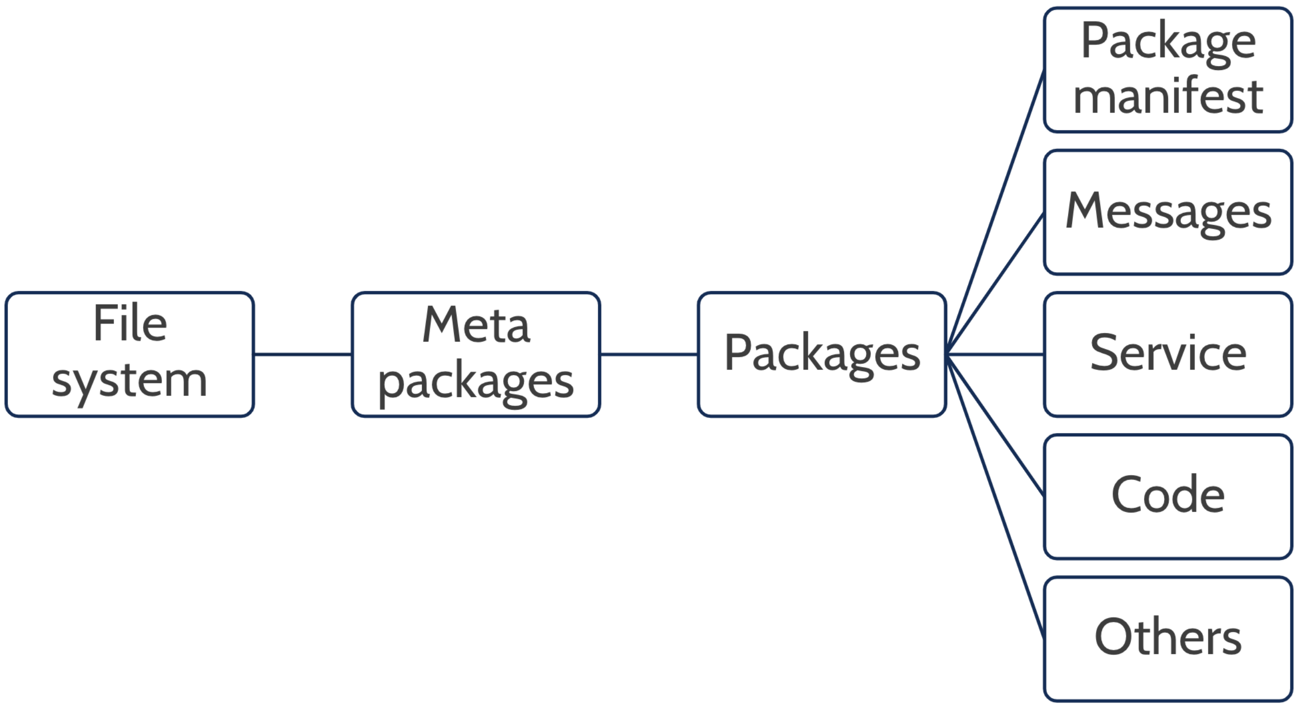


Figure 2.4: File System level representation

- *msg*:Folder containing Message (msg) types
- *src/packagename*: Source files, especially Python source that are exported to other packages.
- *srv/*: Folder containing Service (srv) types
- *scripts*: executable scripts
- *CMakeLists.txt*: CMake build file
- *package.xml*:XML file containing package structure
- *CHANGELOG.rst*: Many packages will define a changelog which can be automatically injected into binary packaging and into the wiki page for the package

Metapackages

Metapackages are specialised structures whose only task is to represent a group of packages that have common characteristics with each other. The metapackages that are created in the context of older versions of ROS and later updated may also result from the conversion of older stacks that perform similar functions in the context of older versions of ROS Wiki [6].

Manifest

A package manifest consists of an XML file named package.xml that must be included in the root folder of any catkin-compliant package. It contains information about the package, including its name, version number, authors, maintainers, and dependencies on other catkin packages. There is a strong similarity between this concept and the manifest.xml file used in the legacy rosbuild build system. System package dependencies are declared in package.xml Wiki [5].

There are a minimal set of tags that need to be nested within the <package> tag to make the package manifest complete.

- *<name>*: The name of the package
- *<version>*: The version number of the package;
- *<description>*: A description of the package contents;
- *<maintainer>*: The name of the person(s) that is/are maintaining the package;
- *<license*: The software license under which the code is released.

Message types
Message types define the structure of messages sent by ROS Wiki [7]. Each file, with extension .msg, represents a different type of message. Within the file each line represents a message field. Each line, in turn, contains two columns: the first one for the data type of the field (Int32/int (C++/Phyton), bool, string, time, etc.), the second for the name. It is possible to assign values to the fields within these le, in this case we speak of constants. Example of msg file (C++):

Service types
Service type are files that define the structure of request/response for ROS services Wiki [12]. These are directly built upon the msg format to enable communication between nodes. They are stored in dedicated .srv files in the srv/ subdirectory of a package. Example of srv file (C++):

```
bool add(beginner_tutorials::AddTwoInts::Request  &req,
           beginner_tutorials::AddTwoInts::Response &res)
  {
    res.sum = req.a + req.b;
    ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
    ROS_INFO("sending back response: [%ld]", (long int)res.sum);
   return true;
  }
```

2.4.2. ROS Computational Graph Level

The Computation Graph is the peer-to-peer network of ROS processes that are processing data together. The basic Computation Graph concepts of ROS are nodes, Master, Parameter Server, messages, services, topics, and bags, all of which provide data to the Graph in different ways Wiki [4].

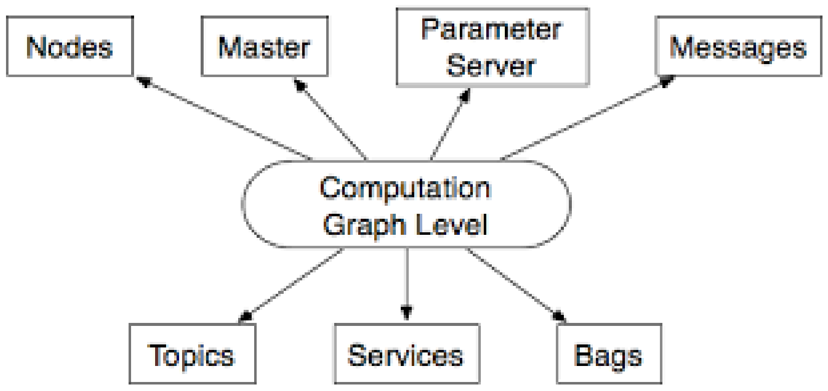


Figure 2.5: Computation Graph

Nodes

A node is a process that performs computation. Nodes are combined together into a graph and communicate with one another using streaming topics, RPC services, and the Parameter Server Wiki [8]. Following the concept of modularity of the system, each node will be related to only one specific functionality.
ROS in fact discourages the creation of 'omnipotent' nodes that perform many functions in order to make the system more maintainable, reusable and clear.
The use of nodes in ROS provides several benefits to the overall system. There is additional fault tolerance as crashes are isolated to individual nodes. Code complexity is reduced in comparison to monolithic systems.

Topics
Topics are buses identified by a proper and unique name that allow messages to be exchanged between nodes. They implement a mechanism of publishing and subscribing: nodes can be publishers and/or subscribers if they are set up to send or receive messages. The division between data producer and data user is clear and separated by anonymity policies between the nodes. Each topic may have a maximum number of messages to be kept in the queue in case they accumulate, those in excess are not added to the queue and lost Wiki [13].

Services
Services are a two-way communication tool between nodes. It is a mechanism that extends that of messages with the possibility not only to send commands to a specific node but also to remain in listening and receiving a structured response from it. Each service is first described in an .srv file where the parameters and type of service are indicated in addition to the name of the service. of the service, as well as the parameters and the type of return data (see Service type). Within the server node, the service is represented by a function which takes as input two pointers to objects of the server class: one one will include the function parameters (Request), the other will collect the return value. will collect the return value Wiki [11]

Messages
The nodes in the graph communicate by exchanging messages. These may be simple and of a primitive type (integer oat, string, char, etc.) or arrays or even more complex, with structures similar to those seen in C.

Bags
Bags represent the system by which ROS saves logs and keeps track of all messages exchanged within a topic. The rosbag tool, once associated with a topic, saves each message exchanged within a related file with the extension .bag. It is very useful for storing data from the sensors as it allows the developer to create a sort of "black box" of the robot. black box" of the robot. ROS also provides a playback tool that allows the visualise and playback the collected data via a graphical interface.

Master

The master in ROS first takes care of registering new nodes within the network, then managing the connection between the nodes in the graph, routing messages and allowing access by one node to the services of another. It is the heart of the software and can only be active one master at a time. It can be started via the roscore command or launched automatically at the start of a node through the correct implementation of the file.

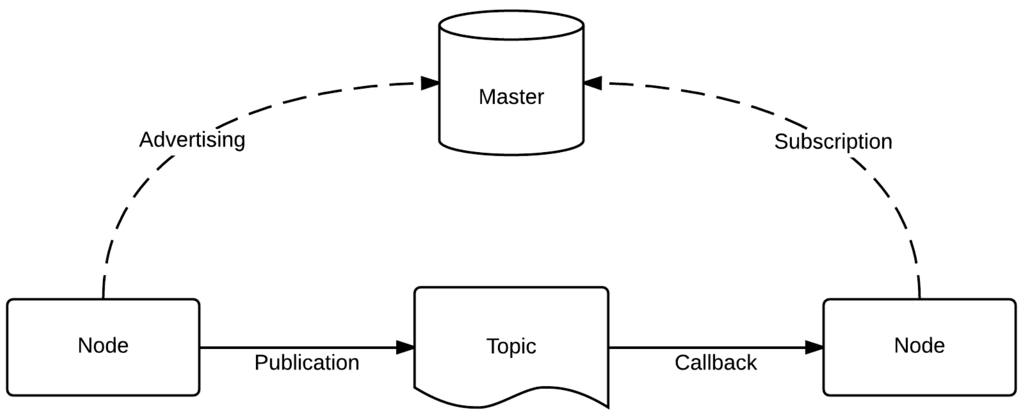


Figure 2.6: Visualization of Master-Node-Topic relationship

Parameter server

The parameter server is basically a component of the master, allowing certain configurations accessible via the network API to be shared publicly with all nodes. Although not an extremely high-performing it is nevertheless useful for the testing phase of the software. Parameters are named using the normal ROS naming convention. This means that ROS parameters have a hierarchy that matches the namespaces used for topics and nodes, Wiki [10].

Coordinate Frames and Transforms

Typically, a robot has many 3D reference systems that change over time. The ROS tf package keeps track of all these coordinate systems in a tree structure. This concept is also necessary to understand later on how URDF files handle the various parent and child links. The tf package therefore, keeps track of all existing relationships between the co-ordinate frames of points and calculates the transforms between them. The transform tree can also be viewed by developers for debugging purposes using the command view frames.

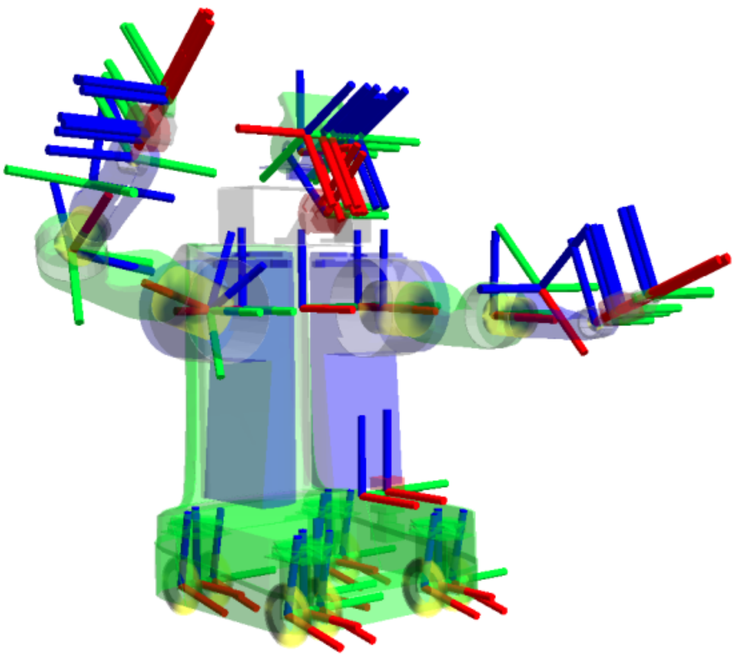


Figure 2.7: Transform Frames of a robot

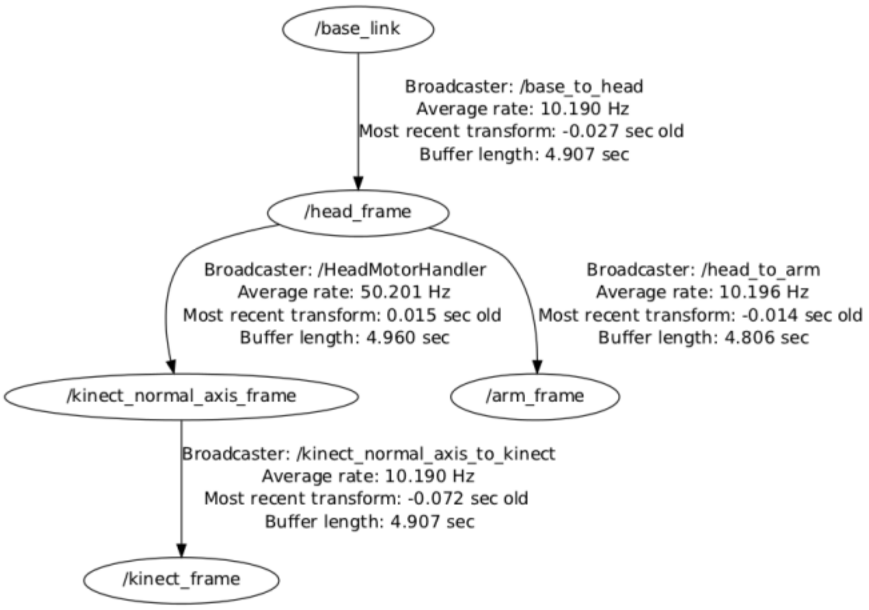

/base_link
Broadcaster: /base_to_head
Average rate: 10.190 Hz
Most recent transform: -0.027 sec old
Buffer length: 4.907 sec
/head_frame
Broadcaster: /HeadMotorHandler
Average rate: 50.201 Hz
Most recent transform: 0.015 sec old
Buffer length: 4.960 sec
Broadcaster: /head_to_arm
Average rate: 10.196 Hz
Most recent transform: -0.014 sec old
Buffer length: 4.806 sec
/kinect_normal_axis_frame
/arm_frame
Broadcaster: /kinect_normal_axis_to_kinect
Average rate: 10.190 Hz
Most recent transform: -0.072 sec old
Buffer length: 4.907 sec
/kinect_frame

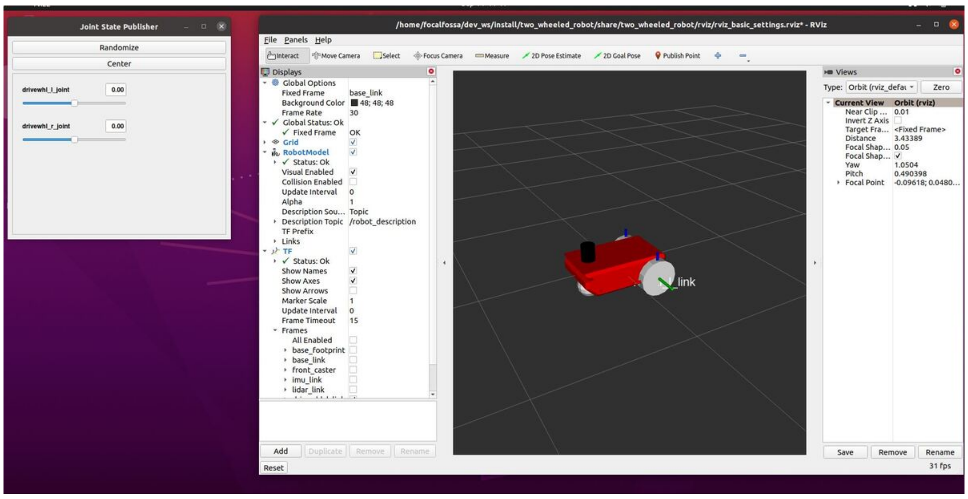
Figure 2.8: Transform Frames Tree of a robot

2.5. ROS Tools and Simulators

ROS provides built-in tools that come to hand when developing robotic applications or when we want to investigate deeper in some issues. The tools described below, although others exist, are the one that were used the most during Oversonic project: RViz (3D visualization tool), Rqt (framework that enables GUI tools: Rqt graph, that displays correlation between nodes and messages in graph form, and Rqt plot) and finally, special emphasis will be placed on Gazebo, a 3D simulator that has long been used in the course of this project.

2.5.1. RViz: 3D Visualization Tool

RViz is a 3D visualization built-in tool of ROS. Its main purpose is to visualize ROS messages and topics in three dimensions, helping the user out to display data and to understand how our system behaves. When opening a new RViz window from scratch, what appears is a black 3D scene. The user can indeed build the environment by customising global options (e.g. the fixed frame that provides a static reference view) and grid settings. It is possible in fact to select which features we want to visualize, ticking them directly from the left panel (picture 2.9). RViz can visualize topic from camera sensor, showing the images on a dedicated box. This feature applies to any kind of sensor that communicates via ROS to our robot, e.g. Lidar, tracking camera, RGB camera etc.



```
drivewhl_l_joint   0.00
drivewhl_r_joint   0.00
```

```
Global Options
  Fixed Frame            base_link
  Background Color       48; 48; 48
  Frame Rate             30
✓ Global Status: Ok
  ✓ Fixed Frame          OK
Grid                     ✓
RobotModel               ✓
  ✓ Status: Ok
  Visual Enabled         ✓
  Collision Enabled      
  Update Interval        0
  Alpha                  1
  Description Sou...     Topic
  Description Topic      /robot_description
  TF Prefix
  Links
TF                       ✓
  ✓ Status: Ok
  Show Names             ✓
  Show Axes              ✓
  Show Arrows            
  Marker Scale           1
  Update Interval        0
  Frame Timeout          15
  Frames
    All Enabled          
    base_footprint       
    base_link            
    front_caster         
    imu_link             
    lidar_link           
```

```
Type: Orbit (rviz_defa
Current View    Orbit (rviz)
  Near Clip ...   0.01
  Invert Z Axis   
  Target Fra...   <Fixed Frame>
  Distance        3.43389
  Focal Shap...   0.05
  Focal Shap...   ✓
  Yaw             1.0504
  Pitch           0.490398
  Focal Point     -0.09618; 0.0480...
```

Figure 2.9: RViz GUI

2.5.2. Rqt: ROS GUI Development Tool

Rqt is a software framework of ROS that implements various GUI tools in form of plugins. In particular, particularly useful is rqt graph. The main goal of this tool is to let visualize ROS nodes, topics and messages in order to facilitate debugging and understanding of the system. In fact, when using ROS it is useful to display the on-going graph to better understand how the various nodes are communicating and how messages are being exchanged. Rqt graph thus results useful in:

- Having a global overview of the system
- Debugging code in case there exist issues in nodes communication (e.g. two nodes are not connected in reality or too many nodes publish on the same topic)

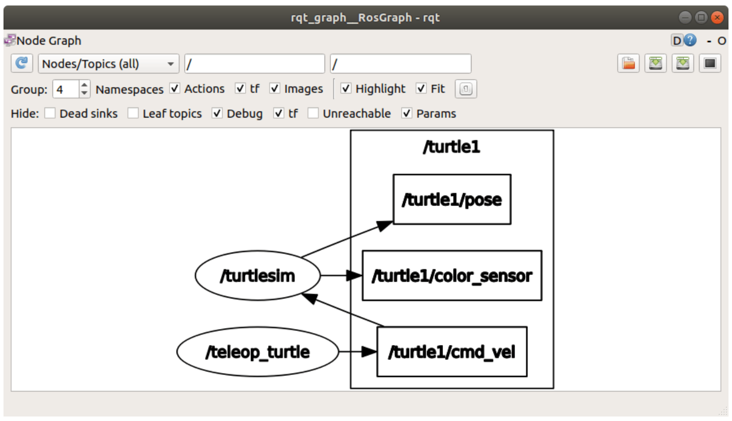
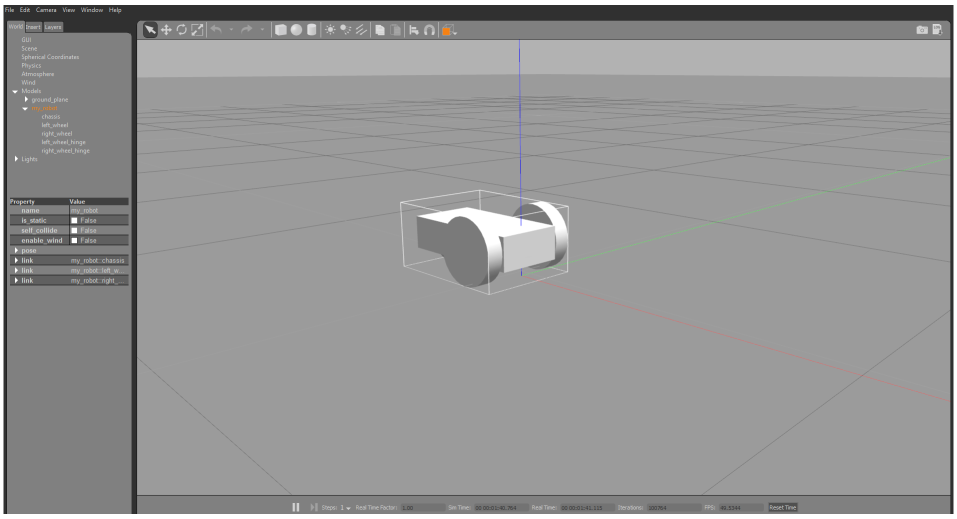


Figure 2.10: Graph from turtlesim

In figure 2.10 two nodes have been initialized and from the graph we can see the path of the messages.

2.5.3. Gazebo Simulator

Dealing with real robots means using physics labs, charging batteries, calibrating sensors and many other small tasks involving hardware. In real-life cases, even the best robots break down periodically due to human error, wear and tear or structural defects. This is where simulators come in: in simulation, we can faithfully simulate the actions that the robot will perform, without the disadvantages listed above. It is also possible to model actuators and sensors either as ideal devices or by adding varying degrees of distortion, error or failure. Thus, the solution of simulating robots in a virtual environment is efficient and cost effective. The problem of SLAM (simultaneous localisation and mapping) has always been one of the most important research topics in the community. In response to this need, 'Stages', for example, with high computational capabilities and different kinematic configurations, built-in sensors have been developed over the years. In the context of this paper, it was decided to use Gazebo, a 3D simulator that provides various models of robots, sensors, environments, offering faithful and reliable simulations thanks to its physics engine. Gazebo is in fact one of the best-known simulators used in open source robotics.



Property	Value
name	my_robot
is_static	False
self_collide	False
enable_wind	False
pose	
link	my_robot::chassis
link	my_robot::left_w...
link	my_robot::right_...

Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:01:40.764 Real Time: 00 00:01:41.115 Iterations: 100764 FPS: 49.5344

Figure 2.11: Gazebo GUI

To use the simulator, it is either possible to download a pre-fabricated model, available among the Gazebo robots (e.g. TurtleBot, PR2, Pioneer2 and other known robots) or to build your own robot model (we'll talk later on about SDF and URDF). As far as robots are concerned, a wide variety of sensors can be applied to your model: stereo camera, RGB camera, tracking, contact sensors. It is also possible to add the noise model to the sensors. ROS integrates very closely with Gazebo, thanks to the gazebo ros package. This provides a plugin module for the simulator that enables bi-unique communication between Gazebo and ROS. Simulated sensors and physical simulation data are thus transmitted bi-directionally between the two platforms.

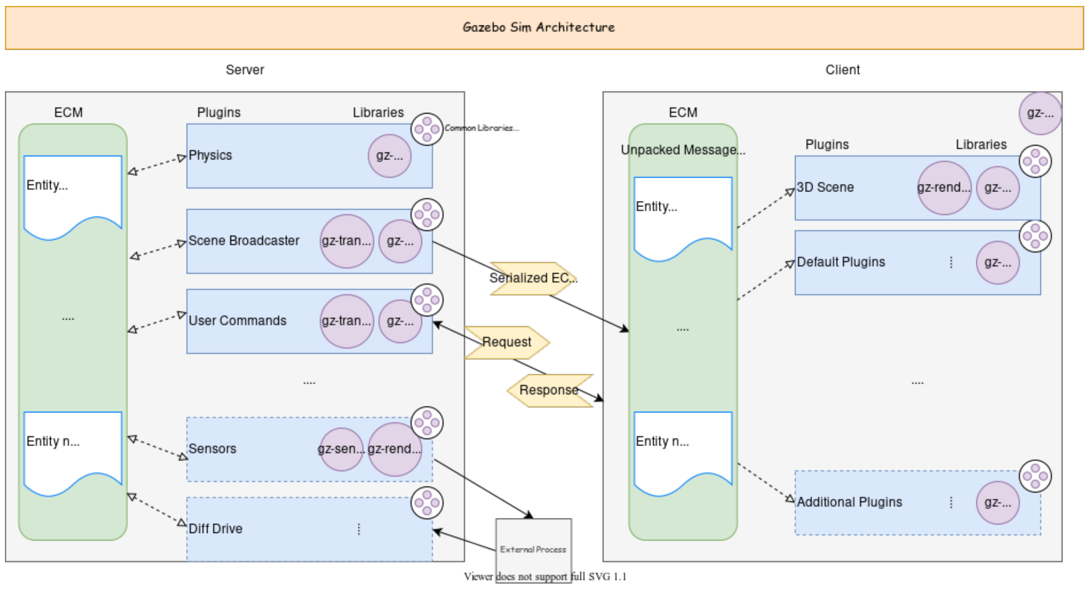


Figure 2.12: Gazebo Sim Architecture

Robot Modelling Formats

As mentioned above, one of the possible ways of describing a robot model is through the URDF. The unified robot description format is a package containing a file in XML format. A URDF file is written in such a way that each link of the robot is a child of some other parent link, with joints connecting each link. In turn, the joints are defined by an offset from the reference frame of the parent link and their axis of rotation. This creates a complete kinematics model. Below is reported a simple sample of the creation of a base link:

```
  <link name="base_link">
   <visual>
     <geometry>
       <cylinder length="0.6" radius="0.2"/>
     </geometry>
     <material name="blue"/>
   </visual>
   <collision>
     <geometry>
       <cylinder length="0.6" radius="0.2"/>
     </geometry>
   </collision>
 </link>
```

When we are developing complex robot models, it can happen that the notation within the XML file becomes verbose and prone to confusion. It is precisely for this reason that the XACRO (Macros XML) model was created, which is nothing more than an XML that allows the design phase of the model to be divided into sub-parts, which are then imported one by one into the main file. This results in cleaner code reading and facilitates debugging.

Below is an example of importing arguments from an external file:

```
    <xacro:property name=    robotname     value=    R001      />
    <link name=    ${robotname}s_leg   />
```

In order to use the model created within the Gazebo environment (which, as we have already mentioned, works with SDFormat files), it is necessary to convert first from XACRO to URDF and finally from URDF to SDF (Simulation Description Format). In fact, files with this format include a description of the world in which the robot will be placed, a series of features related to the simulated physical world (static and dynamic objects, lighting, terrain and even physics), and plug-in additions. Below is an example of SDF file defining a model from scratch:

```
<?xml version='1.0'?>
<sdf version='1.9'>
  <model name='my_model'>
     ...
  </model>
</sdf>
```

3 | Robot

«In the twenty-first century the robot will take the place which slave labor occupied in ancient civilization. There is no reason at all why most of this should not come to pass in less than a century, freeing mankind to pursue its higher aspirations.»
Nikola Tesla (1856 - 1943)

«Robots of the world!
The power of man has fallen!
A new world has arisen:
the Rule of the Robots! March!»
Rossum's Universal Robot (1920)

Man has always spent his life working. Dangerous and degrading work has been the cause of death for many people for centuries. In this sense, there has always been a tendency to try to relieve man of the heaviest jobs by looking for machines or automatic systems to replace him. In a sense, with the advent of the industrial revolution, we witnessed the first real process of robotizing in history. On the other hand, with the evolution of discoveries in the medical field, the desire and curiosity arose in man to try to clone himself, artificially constructing his own like. It is here that these two needs and tendencies come together in what we now call humanoid robots. Indeed, humanoid robots are designed and built to replace humans in the most physical and repetitive tasks, in order to ensure greater well-being.

3.1. Introduction

In recent years, the general public has become increasingly interested in robots and robotics research. New developments, e.g. robotic competitions, which "[push] beyond the boundaries of current technological systems" (such as Defense Advanced Research Projects Agency (DARPA) in the United States), especially in the area of robotics, have promised and delivered fully integrated systems, Lima et al. [1]

4 | Conclusions and future developments

A final chapter containing the main conclusions of your research/study and possible future developments of your work have to be inserted in this chapter.

Bibliography

[1] P. Lima, D. Nardi, G. Kraetzschmar, J. Berghofer, M. Matteucci, and G. Buchanan. Rockin innovation through robot competitions [competitions]. *Robotics Automation Magazine, IEEE*, 21:8–12, 06 2014. doi: 10.1109/MRA.2014.2314015.

[2] B. G. Morgan Quigley and W. D. Smart. *Programming Robots with ROS*. O'Reilly Media, 1 edition, 2015 2015.

[3] Repubblica, 2022. URL `https://www.repstatic.it/content/contenthub/img/2022/05/24/114226177-5cef6076-990d-4f67-b7ae-03d1cc734ee3.jpg?webp`.

[4] R. Wiki. Concepts, . URL `http://wiki.ros.org/ROS/Concepts`.

[5] R. Wiki. Manifest, . URL `http://wiki.ros.org/Manifest`.

[6] R. Wiki. Metapackages, . URL `http://wiki.ros.org/Metapackages`.

[7] R. Wiki. msg, . URL `http://wiki.ros.org/msg`.

[8] R. Wiki. Nodes, . URL `http://wiki.ros.org/Nodes`.

[9] R. Wiki. Packages, . URL `http://wiki.ros.org/Packages`.

[10] R. Wiki. Parameter server, . URL `http://wiki.ros.org/Parameter20Server`.

[11] R. Wiki. Services, . URL `http://wiki.ros.org/Services`.

[12] R. Wiki. srv, . URL `http://wiki.ros.org/srv`.

[13] R. Wiki. Topics, . URL `http://wiki.ros.org/Topics`.

A | Appendix A

If you need to include an appendix to support the research in your thesis, you can place it at the end of the manuscript. An appendix contains supplementary material (figures, tables, data, codes, mathematical proofs, surveys, . . .) which supplement the main results contained in the previous chapters.

B | Appendix B

It may be necessary to include another appendix to better organize the presentation of supplementary material.

List of Figures

List of Tables

List of Symbols

Variable	**Description**	**SI unit**
$\boldsymbol{u}$	solid displacement	m
$\boldsymbol{u}_f$	fluid displacement	m

Acknowledgements

Here you might want to acknowledge someone.