4
Human Robot Interface

This chapter covers the development of the Human Robot Interface, as the involvedtools on its elaboration. The Human Robot Interface runs under the Android Operating System, it uses also the Robot Operating System as an Interface between the robot and HRI.

4.1
Fundamental Concepts

In this section, the basic concepts are introduced in order to obtain better understanding of the Human Robot Interface and its operation.

4.1.1
Robot Operating System

The Robot Operating System (ROS) is a framework for programming Robot software, contains tools, libraries and some standards for the purpose of simplifying the task of programming a robot [74]. It has the following main features: [75].

- Peer to Peer Communication.
- Multilanguage: ROS has the ability to be programmed in several types of programming languages such as C ++, Python, Java, Perl, JavaScript.
- Tools based: It also has various tools that allow the visualization and treatment of the sensory information and interaction with the actuators of the robots.
- Lightweight: ROS can be installed from a simple computer to a embedded computers.
- Free Software: Tools, packages and ROS are under free software license.

Both the sensory information and the commands for the Robot actuators are handled by topics, which are channels where the information travels.

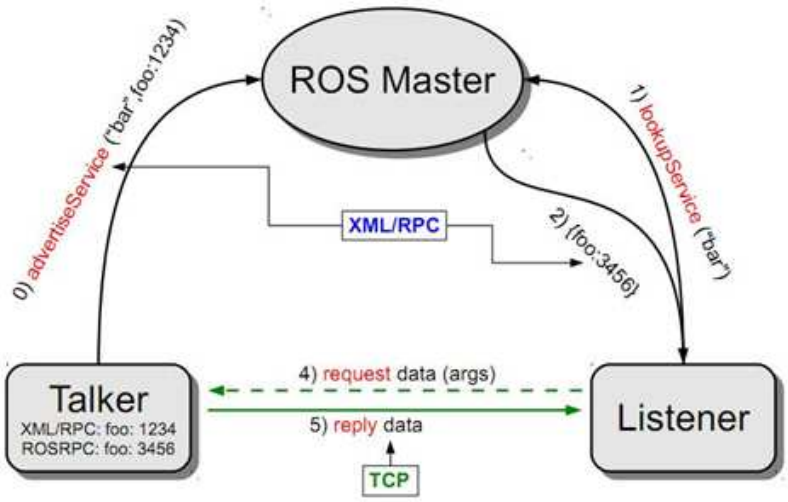

Figure 4.1: Communication scheme for ROS [74]

According to the Figure 4.1, there is a *ROS Master* which is the one that handles the information, every time that a process needs to get information from a topic, the *ROS Master* will take care of to *locate* the topic. There are two nodes (which are the fundamental unit of processing in ROS).

The *Talker* node will be responsible for making sensory information available (or publishing according to the ROS environment) and the *Listener* node will be in charge of requesting the information (or subscribing to ROS terminology).

In order to understand how the architecture of ROS works, and how topics and nodes interacts with sensory information works, the example of the camera will be used. Using the Figure 4.1, the node *Talker* would be in charge of loading the camera's controllers and publishing its information in a topic such as "foo". However, if the requirements of our software need to see the sensory information, we will need a node (*Listener*) to "subscribe" to the "foo" topic and obtain the sensory information and present it in some way.

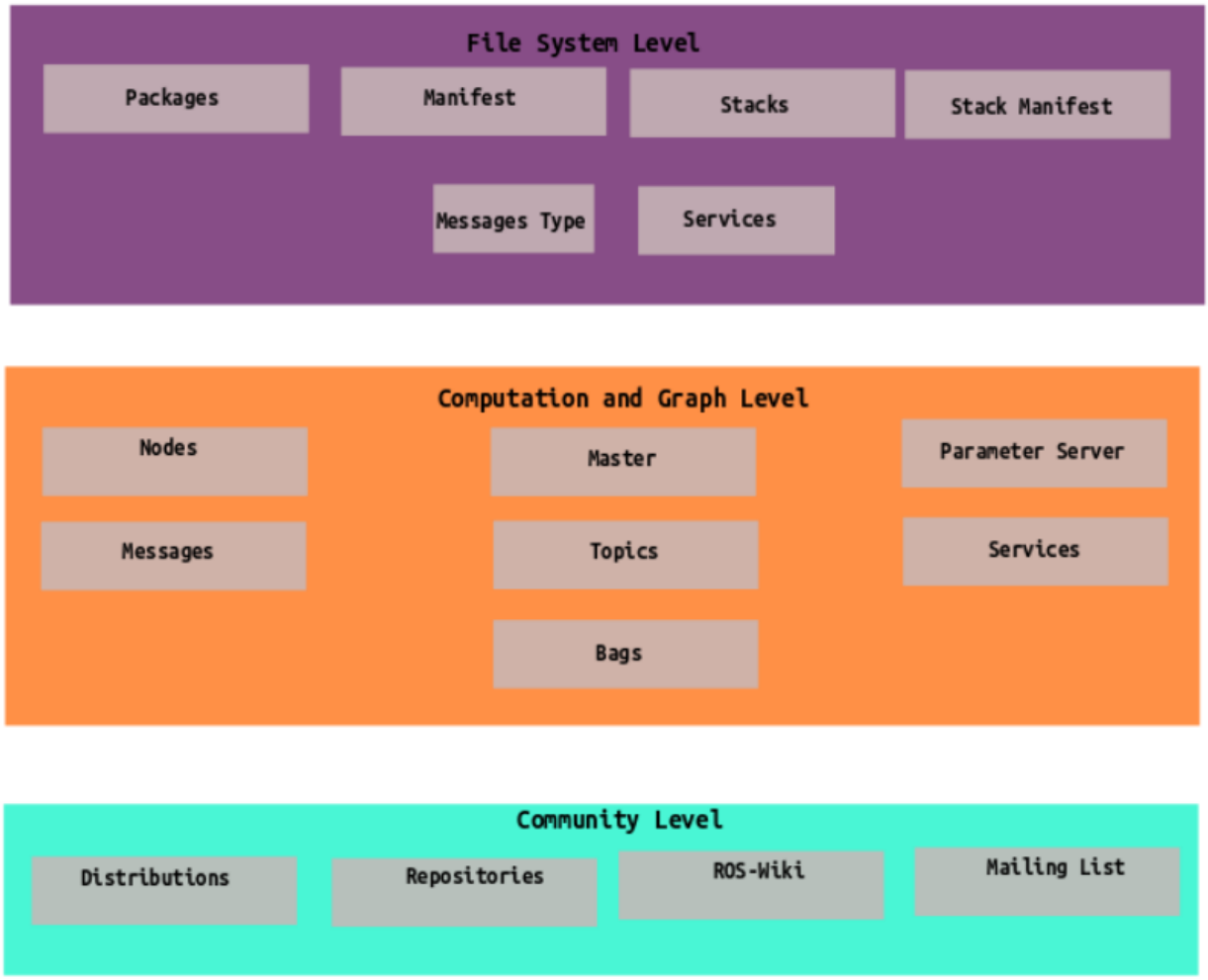

Figure 4.2: ROS Architecture

4.1.2 ROS Architecture

The architecture of ROS has 3 different levels whose components are shown in Figure 4.2, next each level is explained:

1. **FileSystem Level:**

   They are the resources of the ROS at the level of physical storage (hard disk of the computer), it contains all the types of files that ROS generates, generally the XML format is used for each file:

   (a) *Packages*

   It is the smallest unit of software organization in the ROS, for example, if we talk about a particular robot and its software in ROS, the robot will have a package, which can be downloaded from various internet repositories.

(b) *Manifest*

It contains information about each package, such as the author, email, people in charge of maintenance, organization to which he belongs, etc.

(c) *Stacks*

They are collections of several packages, which provide functionality, for example, when it comes to using the Computational Vision software, the packet collection (*Stack*) called *Vision Opencv* is used.

(d) *StackManifest* Contains Metadata about *stack* in the same way as the *Manifest* of a package.

(e) *Message Type*

Description of messages, is saved *mypackage/msg/MyMessageType.msg*, defines the data structures for messages sent in ROS. ROS contains primitive message types, however, when the application that we are developing requires, then is necessary to create new types of messages.

(f) *Service Types*

Description of Services, it is saved *mypackage/srv/MyServiceType.srv*, it defines the data structures for requests and the answers in the services of ROS. When we need an immediate information without needing to use Topics, Services for the request/response communication is used.

2. **Computation and Graph Level**

It defines the communication between ROS processes, it contains the programs generated by the ROS user.

(a) *Nodes*

Processes where computation is done, the ROS is made to be modular, each module can make a part of a whole, for example, the node (module or program) in charge of managing the movement of the wheels of a Mobile Robot.

(b) *Master*

As detailed in the Figure 4.1 provides the name registry and communicates to all the rest of the level *Computation and Graph Level*.

(c) *Parameter Server*

It allows data to be stored in a data center, is a part of the *master*, and contains global variables.

(d) *Messages*

They establish the communication between nodes, it is a data structure of typed fields, it contains primitive data (*integer, floating point, boolean*). Messages can include arbitrarily nested structures and *arrays*

(e) *Topics*

When a node is sending data, it is publishing in a topic, when it is receiving data it is subscribed to a topic, they are like communication buses in which the messages travel.

(f) *Services*

A node offers a service under a name and a client uses that service by sending request messages and waiting for a response.

(g) *Bags*

Format to save and retrieve messages. For example, when data from an experiment is collected and is required to reproduce the data again in the laboratory, these files are used.

3. **Community Level**

(a) *Distributions*

They are collections of *Stacks* which are placed in repositories, each of these distributions are released every year, additionally they are versions of ROS. Currently (2019) the distribution is *Melodic Morenia.*

(b) *Repositories*

Different institutions can develop packages or *stacks* and launch their own versions for their own robots, this software under the Free Software policy is shared in open repositories so that the whole community can use it.

(c) *Ros-Wiki*

It is a web page where the great part of the ROS documentation is located, it contains tutorials, anyone can contribute with this documentation.

(d) *Mailing List*

ROS has an official list of stacks developers where possible bugs are written, next versions, job announcements, etc.

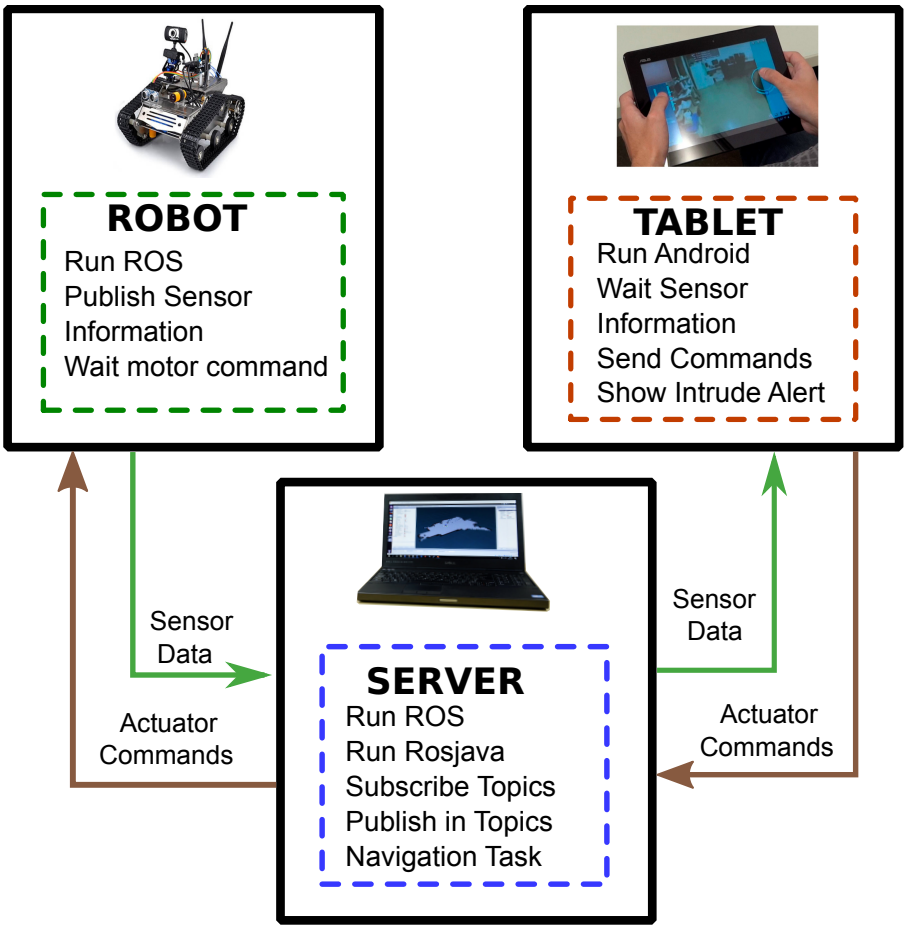

Figure 4.3: System Overview

4.1.3 RosJava

Rosjava provides a client library for ROS communication in java and has the access of all the tools from ROS. Rosjava implements native messages for the ROS and implements a tool to build custom messages as services.

In this work the core communication system is build in java, then, as a consequence, it becomes a requirement of using the same language programming as interface between the Robot and the interface (see Figure 4.3).

4.2 System Overview

In the elaboration of the MUI, we use Robot Operating System (ROS) [75] as a robotic framework, this is embedded in the robot and facilitates the communication with the other components in the system.

In order to communicate the robot and the smartphone interface, we used the communication scheme showed in [54], which includes a wireless network where each component (tablet, server, robot) are connected.

Different components were used as we can see in Figure 4.3, First on the left hand we have the robot that is provided by sensors and actuator, that is published in ROS. Next, this sensor information is sent to a server that plays the role of central of surveillance monitoring, it receives the sensor information, and finally, the server sends to the Android Client (Tablet) on the right side.

4.3 Server

The server, as was mentioned in the previous section plays the role of central of surveillance monitoring, so the server will run ROS and Rosjava which is a ROS client library for Java programming language, it allows an efficient communication with ROS without losing the main features of Java such as object-oriented, distributed, multi-threaded. The server performs different tasks: Publish Sensor Information, Receive Commands and Navigation.

4.3.1 Publish Sensor Information

As was mention before, ROS is being used. In this framework exists a concept *Topic* which always contains sensor information or waits for actuators commands, they are like pipes were always information is running. In order to communicate the topics with RosJava, we can use nodes (programs), there are two forms to program a node: first, the node that consumes information called the subscriber and the node that puts information into a topic called the publisher.

Then, the sensor information arrives at the server, and this information is published as a ROS topic, so every sensor information has its corresponding topic.

Now, having the Sensor Information available, it should be transmitted to the smartphone interface. This process is made by using the Rosjava library first, it obtains the information making a *subscriber node*, the next step is to send this information to the interface for this objective, we send it using the SocketClient library of Java. Is good to remark that the Interface will perform a SocketServer that will wait for the connection of this node.

4.3.2 Receive Commands

As we can see in the Figure 4.3, the interface (Tablet) will send commands to the robot, so the server must be prepared to perform these commands.

In the process of moving the surveillance robot, the first step should be to program a node *publisher*, this node will send the received commands (from the mobile interface) to the robot, the next step is to receive the commands from the interface to make it In this task we execute a *SocketServer* from the Java *Socket* Library, this *socket* will wait for a connection from the interface and will publish the commands received in the corresponding topic.

4.4 Tablet

The tablet is the interface of Smartphone compatible with Android, it is built with Android Studio IDE because it also allows the use of Java features.

In this interface used to teleoperate the robot, according to the review of the state of the art, it was observed that some interfaces present the sensor information and the separate teleoperation interface [47] or only have a teleoperation interface (only to move motors) [53].

Given this in our teleoperation interface, it will have different functions, such as receiving information from the sensor and send commands to the robot, performs two different modes, autonomous and manual.

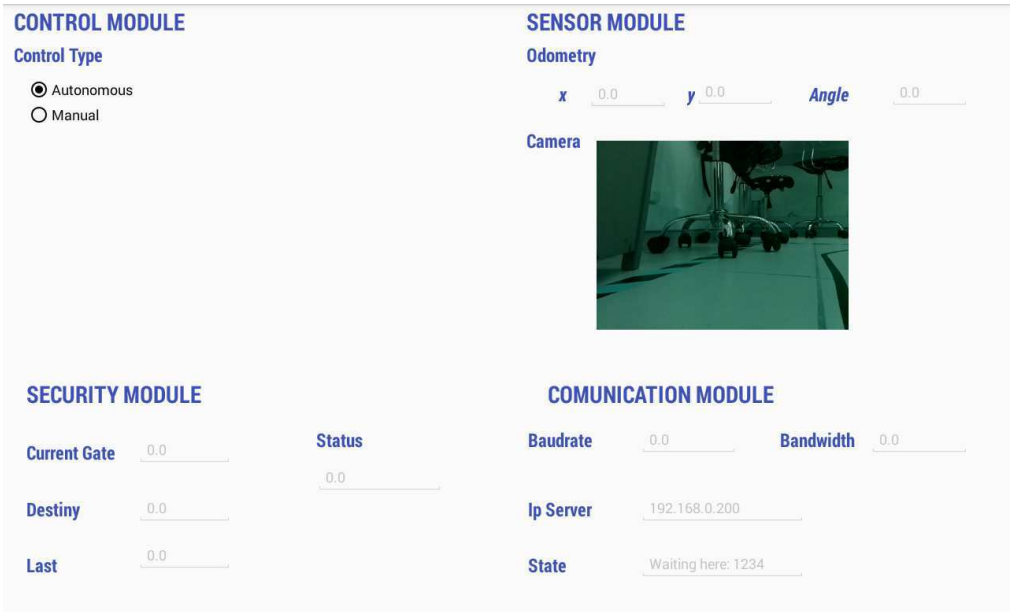

Figure 4.4: Tablet Interface

For this the interface will implement buttons for robot movement, as

shown in the Figure 4.4, also will show the sensory information such as camera, odometry and communication status.

4.4.1 Receive Sensor Information

As we mentioned in section 4.3.1 the server sends sensor information and suppose that there is a server waiting for communication, this server is implemented here. In this subsection we will describe how the user can make a *subscriber node* but in the Android Platform.

The server is using the same mechanism that section 4.3.2 but we add another one which is a Multi-threaded feature from Android. This SocketServer waits for a connection from the server and when it happens, it will show in different ways as can be seen in Figure 4.4.

Here the sensor information is shown such as Encoders information, line sensor, camera. Also will show the perception of the Navigation algorithm whether there is an intruder or not.

4.4.2 Send Commands

Now, as mentioned previously there will be a *SocketServer* waiting for commands, now the client that will send these commands is implemented here. Similar to the previous subsection, we will describe how to implement a node publisher, but on the Android platform.

The Client that sends commands to the server will be implemented using the Java *SocketClient* library, will connect to the server and will publish information on the actuator's topics.

Both processes that receive and send information work with the Java Sockets library, using this library we send a Java object that contains information and commands from the sensor.

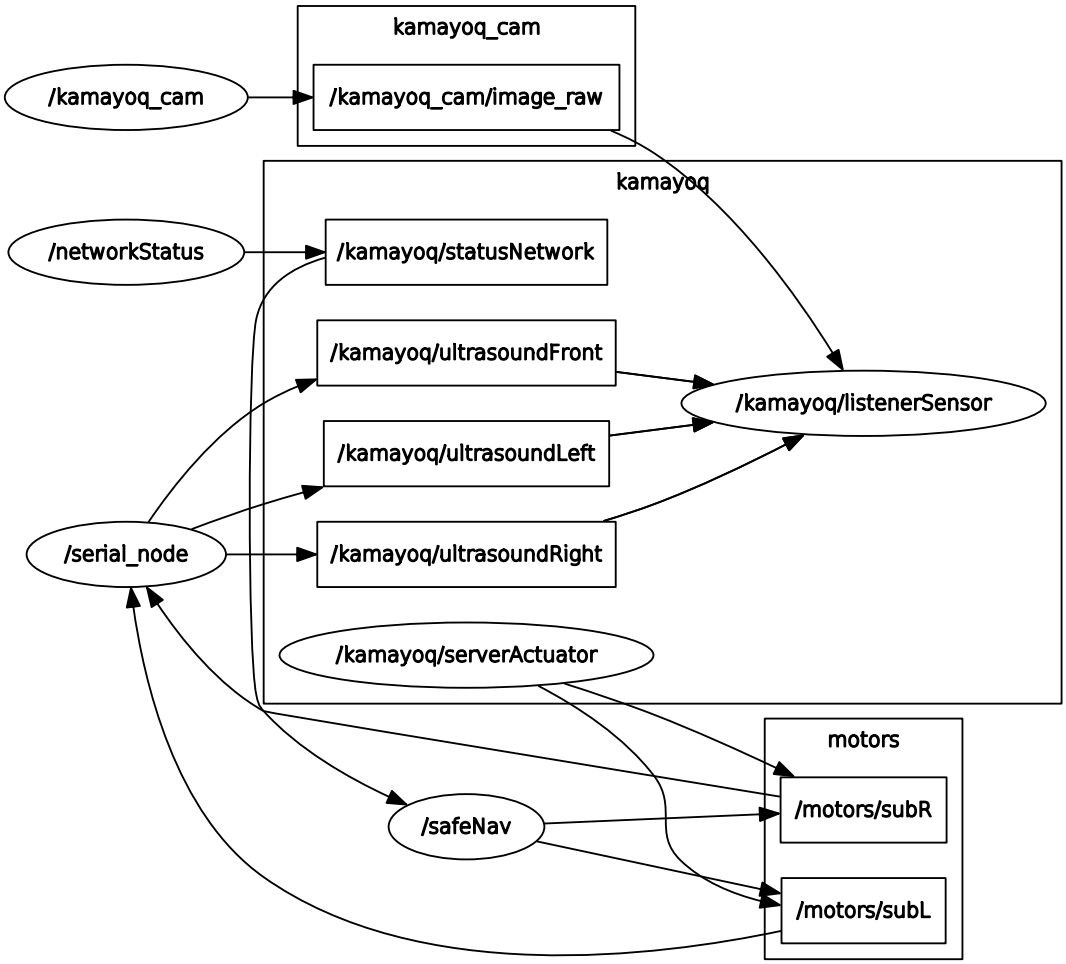

Figure 4.5: Nodes in HRI

4.5 Nodes in the HRI

As was mentioned before Robot Operating System (ROS) works with nodes, to make the teleoperation platform we use several nodes which were programmed and can be appreciated in Figure 4.5. In this Figure also can be seen rectangles and ellipses connected with arrows. An ellipse is a node, a large rectangle is a namespace and a small rectangle is a topic. About connections, a connection that leaves a node to a topic indicates that the topic is a publisher that is to say information to the topic for example, a topic that sends information to the actuators (engines), a connection that leaves a topic to a node indicates that the topic is a subscriber, it means, a topic that publishes information, for example sensory information such as the camera.

Next the main nodes used in teleoperation are explained:

4.5.1 serverActuator

This node is the receiver of commands of the human robot interface, that is, it receives commands from the Tablet and executes them in the robot. That is the reason that Figure 4.5 shows that the node is directly connected to the topics of the engines.

4.5.2 listenerSensor

The function of this node is sending the sensory information to the Tablet, as can be seen in Figure 4.5, to this node comes the information of all topics of sensory information (for example camera and ultrasound).