POLITECNICO DI MILANO

Facoltà di Ingegneria
Scuola di Ingegneria Industriale e dell'Informazione
Dipartimento di Elettronica, Informazione e Bioingegneria

Corso di Laurea Magistrale in
Ingegneria Informatica

Un modello per tesi di laurea magistrale al DEIB

Relatore:

PROF. MATTEO MATTEUCCI

Correlatore:

DR. PAUL KRUGMAN

Tesi di Laurea Magistrale di:

GIOVANNI BERI
Matricola n. 852984

Anno Accademico 2016-2017

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

GRAPE Ground Robot for vineyArd Monitoring and ProtEction

ROS Robot Operating System

LIDAR Laser Imaging Detection and Ranging

1 BACKGROUND AND TOOLS

In this chapter we are going to describe the general concepts this thesis deals with, together with the main tools we used to address the project. Since this thesis is in the frame of Ground Robot for viney Ard Monitoring and ProtEction (GRAPE) project (see Chapter 2), most of them are typical of the robotic field and, more specifically, of the agricultural robotics. This last is a part of the so-called *E-agriculture*:

ROBOT OPERATING SYSTEM

Robot Operating System (ROS) is the *robotic middleware* used for the development of the whole system described in this thesis. We decide to use it because of its great modularity, the availability of a very large number of packages, well documented APIs and an active community. Moreover, ROS is a very widespread system, so its power and versatility are well known in the field of software development for robotics. Citing words from its offical website[1], these are ROS main features:

> *It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers*

ROS is actually a *meta-operating system*, that is, it's not an operating system in the traditional sense (it requires to be run on top of an another operating system; the only officially supported OS is Linux Ubuntu), but it provides a peer-to-peer network that processes can use to create and process data together. This network is implemented through TCP, and it's called *Computation Graph*. In this section, we're going to describe ROS with more detail, with particular emphasis on the different tecniques that nodes can use to communicate among them.

ROS MASTER Even if the Computation Graph is a peer-to-peer network, a central process, called **ROS Master**, is required to exist, to provide naming and registration services to all the user processes In this. Once the processes have located each other through the services offered by the Master, they can communicate peer-to-peer without involving a central entity;

1 http://wiki.ros.org/ROS/Introduction

NODES The processes that are in the Computation Graph are called **nodes**, and they are the atomic units of the computational graph. The ROS API are available in C++, Python and Lisp, but C++ is the most widely used. One of the aims of ROS is to be modular at a fine-grained scale, so a complex task should be achieved through cooperation of several different nodes, each with quite narrow tasks, rather than one large node that include all the functionalities. Nodes can use different techniques for communication, depending whether the message is a part of data stream or it is a request message (*i.e.* a response message is expected) and, in this last case, on the (expected) duration and complexity of the computation of the response.

TOPICS Topics implements a *publish-subscribe* paradigm, are they the easiest way that nodes can use to communicate with each other, and basically are named channels, characterized by the type of the messages that are sent through it. When a node *publish* a message on a certain topic, the message is read from all the nodes that previously *subscribed* to that topic, interfacing with the Master. Note that:

- this technique leads to a strong decoupling between publishers and subscribers to a topic, because a publisher node is, from an high-level perspective[2] , not even aware of the presence of subscribers, and viceversa.
- the relationship between publishers and subscribers is *many-to-many*, *i.e.* multiple nodes can publish on a topic, and multiple nodes can subscribe to a topic

We can easily conclude that this method is very suitable for passing streams of data (*e.g.* the handler of a Laser Imaging Detection and Ranging (LIDAR) streams its measurements over the network, or a node publish the velocities commands for the wheels of a robot), but there is no notion of a *response* to a message, so it's not suitable for *request-response* communication.

SERVICES Services are defined by a name, and a couple of message types that describe the *request* type and the *response* type. Each service is offered by a Service server to any Service client that perform a call. So, Services implement an inter-node communication that is very similar to traditional function calling in most common programming languages (*e.g.* C++, Java), in the sense that:

- Service calls are blocking
- using Services, the inter-node communication is *one-to-one*

2 Actually, publisher nodes always know the list of nodes subscribed to their topics. But this is only used in connection phase, and to avoid a situation where a node publish on a topic with no subscribers, for the sake of efficiency.

These properties make Services suitable for punctual (in opposition to data stream) inter-node communication, such as: request of parameters values to another node, ask a node that handles a camera to take a picture, ask a node that perform navigation task to clear the current map.

ACTIONS While Services, with their resemblance to traditional function calls, can address pretty well the problem of *one-to-one* inter-node communication, they can be quite unsatisfying if the computation required to produce the response is demanding in term of execution time (*e.g.*, navigation of a robot from one point to another in an environment), because the caller is stuck at the line with the Service invocation until the end of the procedure. Services show their weakness also in situations where it could be useful to observe the intermediate results of the computation triggered by the request (*e.g.*, a very complex manipulation procedure). **Actions** are very suitable in this context because, at the cost of a more complex implementation, provide an asynchronous and fully preemptable remote procedure call, with the possibility of monitoring intermediate results if needed. Differently from Topics and Services, Actions are not native in ROS, and their functionalities are built on top of the other ROS messagging systems. Asynchronicity is provided by the use of callbacks.

ODOMETRY

SENSOR FUSION

NAVIGATION STACK

2 THE GRAPE PROJECT

The GRAPE project chapter

3

THE ROBÌ PROJECT

The Robì project chapter

4 GRAPE SOFTWARE ARCHITECURE

Grape sw architecture

5 LOCALIZATION

Localization Chapter

6 KINOVA ARM

Kinova Arm chapter

7 EXPERIMENTAL RESULTS

The experimental results chapter