Un modello per tesi di laurea magistrale al DEIB

Relatore:

PROF. MATTEO MATTEUCCI

Correlatore:

????

Tesi di Laurea Magistrale di:

GIOVANNI BERI
Matricola n. 852984

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

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 vineyArd 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* we used to develop the sofware components of the 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; currently, 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 weaknesses 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.

TF: THE TRANSFORM LIBRARY

tf is a ROS library, which task is very important to understand in order not to get lost in the next sections and chapters. The goal of *tf* is:

> " *[...] provide a standard way to keep track of coordinate frames and transform data within an entire system such that individual component users can be confident that the data is in the coordinate frame that they want without requiring knowledge of all the coordinate frames in the system*" (Foote, *2013*)

The utility of such a component is straightforward, even in quite simple robotic systems. We'll describe here a situation we stumbled upon exactly in the development of the GRAPE project, where of the utility of *tf* is very easy to understand; you'll be able to better contextualize this example after you've read Chapter 6. In this example a LIDAR, mounted on top of the final joint of a robotic arm, acquires data while the arm is moving in order to create a point cloud that will be processed later. To get a meaningful point cloud, it's mandatory to keep track of the movement of the LIDAR with respect to a point with speed equal to zero (*e.g.* the base link of the arm, or the base link of the whole robot), and this gets even more difficult because of the
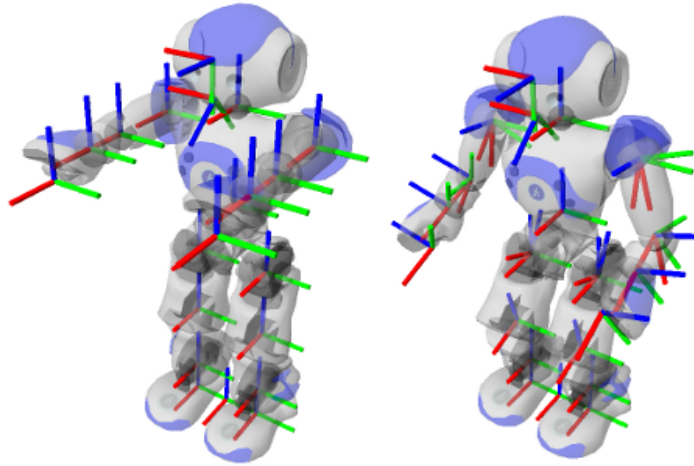
Figure 1.1: *A robot in 2 different positions, with* tf *frames in evidence: x-axis is red, y-axis is green, z-axis is blue. The frames are the same in both configuration, but the transformations (i.e. rototranslations) between them are different*

multiple (6 in our specific case) joints of the arm; but this problem can be easily addressed by means of *tf*, that we are now going to describe with more detail. *tf* implementation relies on ROS topics (see Section 1.1) and achieve the goale mentioned before by building an oriented graph where vertices are reference frames, and edges are transformations (rototranslations) between frames. *tf* does not assume a constant structure and, if a path exists between two reference frames in the graph, the direct transformation between them can be computed by composition of transformation. Since, in general, multiple paths between 2 vertices can exists in a directed graph and this could lead to ambiguity in computing the transformation between two reference frames, the graph is forced to be acyclic. Disconnected subgraphs are allowed, but of course transformation between vertices that belong to different subgraphs cannot be computed. The main components of the library are:

- *tf* **broadcasters**: they are simple software components, that publish a transformation between two reference frames every time an update is available. Different broacasters does not sync together the publishing phase

- *tf* **listeners**: they are more complex components, because they take into account that broadcasters are not synced. Since both transformations and queries to *tf* graph are stamped, listeners make use of queues to store the most recent transformations, and they interpulate old values using SLERP (Spherical Linear intERPolation) to return a transformation for which there is no measured value at the requested timestamp.
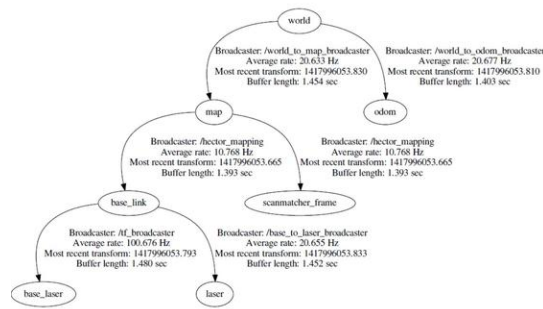
Figure 1.2: *An example of* tf *tree*

In figure 1.1 you can see a graphical representation of the reference frames tracked in a Nao Robot, while figure 1.2 shows an example of *tf* graph visualizad with a the visualization framework *rqt*.

ODOMETRY

Odometry is a very important

- cosa si intende per odometria
- motion models: differential drive e skid steering
- sensor fusion cos'è, e esempio di Robot localization
- localizzazione con amcl

NAVIGATION STACK

SKETCH:

- cos'è navigation stack (immagine http://wiki.ros.org/move_base?action=AttachFile&do=get&tar
- local/global costmap

THE GRAPE PROJECT

The GRAPE project chapter SKETCH:

- descrizione del progetto e i suoi macro goals (navigazione autonoma, monitoraggio della salute della vigna, dispenser dei ferormoni, interfaccia user friendly)

- enti collaborativi (polimi, eurecat, vitirover) con relativi compiti

- difficoltà previste: outdoor, terreno accidentato

- parti su cui ho lavorato io direttamente (design dell'architettura sw, scan motion, parti del deployment)

- qualche foto di vigna/dispensers

GRAPE HARDWARE AND SOFTWARE ARCHITECURE

Grape architecture SKETCH:

- Hardware del robot:
  - scelta dell'husky (adatto per rugged terrain, supporto di pacchetti ROS), foto
  - scelta del braccio, con pro (compatto, 6 dof, power consumption non elevata) e contro (movimenti non precisi, fragile, scarso controllo di collisione)
  - lidar sul braccio per scan
  - lidar davanti per navigazione e eventualmente localizzazione
  - camera montata sul braccio
  - velodyne davanti (a cosa serve?)
  - camera fissa per eventualmente scattare foto alla vigna
  - imu

- foto del robot

- Software del robot
  - descrizione più precisa di quale è nel complesso la procedura: viene dato a mano (interfaccia grafica di Eurecat) una posizione finale, e serie di waypoints intermedi. Procedura di scan che usando PCL trova punti adatti al deployment. Procedura di deploymnet. Waypoint successivo.
  - struttura col coordinatore, che chiama le 3 action (move_base, scan, deploy)
  - di ogni azione, il file che descrive i tipi di goal/result/feedback, con spiegazioni del significato dei parametri

THE ROBÌ PROJECT

The Robì project chapter:

- obiettivo: realizzare un'altra base mobile alternativa all'husky su cui portare il sistema GRAPE

- architettura hw: sensori che ci sono montati a bordo, spiegazione sistema controllo motori tramite CAN usando la board STM

- architettura sw: compatibile con quella di Grape, a meno dei nodi driver dei sensori e attuatori

- spiegazioni e immagini del vario hardware

5

Localization Chapter SKETCH:

- problemi avuti: mappa molto ripetitiva, il robot tenta di passare attraverso i filari, odometria delle ruote poco affidabile dato il terreno, problemi con amcl

- virtual obstacles: cenni (perché non l'ho fatto io) e immagini

- configurazione di robot localization, con i sensori usati (ruote+imu+gps)

- non sappiamo ancora quale sarà la soluzione utilizzata alla fine tra la versione mapless e amcl integrato con robot_localization (o eventuali altre soluzioni che salteranno fuori). Una volta che è stata presa una decisione, presentare quella come soluzione definitiva e spiegare perché l'altra è stata giudicata meno efficace

KINOVA ARM

Kinova Arm chapter: SKETCH

- moveit: cos'è, concetti di planning e execution

- scopo del braccio in GRAPE: scan della pianta, deployment, automa a stati che rappresenta la sequenza di azioni del braccio

- problema 1 dovuto alla scarsa precisione: accenni alle parti di visual servo con marker per compensare questo problema in fase di grasping e deployment (cenni, fatta da Polito e Bascetta).

- problema 2 dovuto ai limiti di execution di moveit: impossibilità di fare un movimento di scansione fluido e completo usando moveit, metodo della jacobiana e la pubblicazione delle velocità

- utilizzo della camera per validazione del grasping e del deployment

- foto del braccio, immagini della nuvola di punti processata dal codice di ferran, alcune immagini dell'image processing nelle fasi di visual servo e di validazione di grasping/deployment

7

EXPERIMENTAL RESULTS

The experimental results chapter:

- TODO: decidere prima della integration week che cosa vogliamo metterci, così sappiamo che dati procurarci

BIBLIOGRAPHY

Foote, Tully (2013). "tf: The transform library." In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. Open-Source Software workshop, pp. 1–6 (cit. on p. 3).

BIBLIOGRAPHY

Foote, Tully (2013). "tf: The transform library." In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. Open-Source Software workshop, pp. 1–6 (cit. on p. 3).