Appendix C Simulation Topics

C.1 ROS

The Robot Operating System (ROS) is a set of software libraries and tools that make it possible for the users to build robot applications. It is an open source project created by Willow Garage and maintained by the Open Source Robotics Foundation (OSRF). It is sponsored and supported from a wide range of well-known organisations and corporations such as NASA, DARPA, NSF and Bosch, Qualcomm and Mathworks.

The ultimate goal of using this framework is to enable rapid prototyping in robotics. Similarly the web 2.0 could be cited. By using tools like apache, MySql and python in a Linux environment, it was made possible to move from static web pages to dynamic or user-generated content and the growth of social media. Likewise this example, there is a need for a group of tools, compatible to each other that will boost the development in robotics field. Fortunately these days this is can be realized, by exploiting the work of many individual developers' communities; there are many sophisticated and well supported tools like ROS as a framework, Gazebo for 3D simulations, Stage for 2D simulation, OpenCV for computer vision and many others more specialized in specific domains. All these packages are open sourced and structured in way that can be utilized easily by other platforms such as iOS and android.

ROS offers many advantages to the developers and based on these it has become a popular tool used in labs, classrooms and companies around the world. There is a 62% increase in the number of papers citing the ROS overview paper (Quigley et al., 2009) from August 2103 to July 2014 according to the community metrics report (Conley and Foote, 2014). Some of the strong points that make ROS a dominant solution in robotics are the following:

- It is an easy to use, cross-language (C++, python, Lisp) inter-process communication system that is fairly versatile. It is based on TCP-IP as well as shared memory communications.
- It allows easy integration of a wide range of tools such as visualization of robot kinematics and sensor data, path planning and perception algorithms. In addition, it includes a wide range of low-level drivers for commonly used sensors.
- It is a fully scalable platform that can support anything from simple ARM CPUs to Xeon clusters.
- There is a large international online community that offers support in possible problems and requests, registering a total of 18.144 questions in the duration of one year (Aug.13 – Jul.14). Something that indicates the growing popularity of ROS is the fact that this number of questions is 38% higher compared to the previous year, according to the community metrics report (Conley and Foote, 2014).

Aside from the loud advantages that ROS has, there are also inevitable drawbacks. From my perspective, it has a very steep learning curve; it is not easy for new users to grasp. The users should spend considerable time to install the packages in Linux machines and solve any dependency incompatibilities between the different versions and computer configurations. As a result at the beginning of the process a lot of effort is spent for the setup rather than the design and understanding of the algorithms. Despite it not being clearly stated on the software, having good C++ skills is a strong advantage. Moreover, there are only a few books available to learn it, although the ros.org website offers many tutorials. As a result there are not many teaching alternatives to experimenting with a working ROS installation and starting from the classic turtle simulation, then moving towards more complex structures like manipulators and mobile robots. At the end of the day, it is all about effort and time.

C.2 Gazebo

Gazebo is a very popular tool in robotics simulation. Gazebo is a 3D simulator, while ROS serves as the interface for the robot. Combining both results in a powerful robot simulator. It allows rapid algorithm testing, robot design and regression testing using realistic scenarios. It can be used for both indoor and outdoor scenarios. In addition, it supports simulation of population of robots. Some of the great characteristics of Gazebo are the way by which it facilitates the integration physics engines and its high quality graphics. Furthermore, it offers convenient programmatic and graphical interfaces. Using Gazebo the users have access to a wide range of ready-made robot models and sensors with noise. Having said that, it is also possible for the users to use a specific language called SDF to build their own models and sensors too. Finally, by using Gazebo, users have access to multiple high-performance physics engines, which can easily simulate effects on illumination, gravity, inertia and other physic entities.

All in all, Gazebo is a great tool that every roboticist should have available in their toolbox. It speeds up the testing in difficult or dangerous scenarios without any harm to the robot. Most of the time it is faster to run a simulator instead of starting the entire procedure on a real robot. From my personal experience of using this tool, I concluded that it is that sometimes unstable and gives errors while other times it executes flawlessly without any change. It is computational-resources ‘greedy’ and highly affected by the graphics card’s characteristics and capabilities. It is probably inadvisable to execute Gazebo in machines with low resources.

C.3 Terminology

There is a solid structure that a ROS system should follow independently of the programming language or the nature of the application. This allows us to have a simple standard code structure and to follow a schematic way of work.

Henceforth, there is a concise presentation of the ROS ecosystem (fig.C.1). References taken from (Bohren, 2015) and (ROS.org, 2015)

ROS Core

ROS core is a collection of nodes and programs that are pre-requisites of a ROS-based system. More precisely, it consists of three programs that are necessary for the ROS runtime. These programs are:

- *ROS master* which manages the communication connections. It provides name service in ROS.
- *Parameter Server* which stores persistent configuration parameters and other user-defined data.
- *Rosout* which is a network-based standard output for human readable messages.

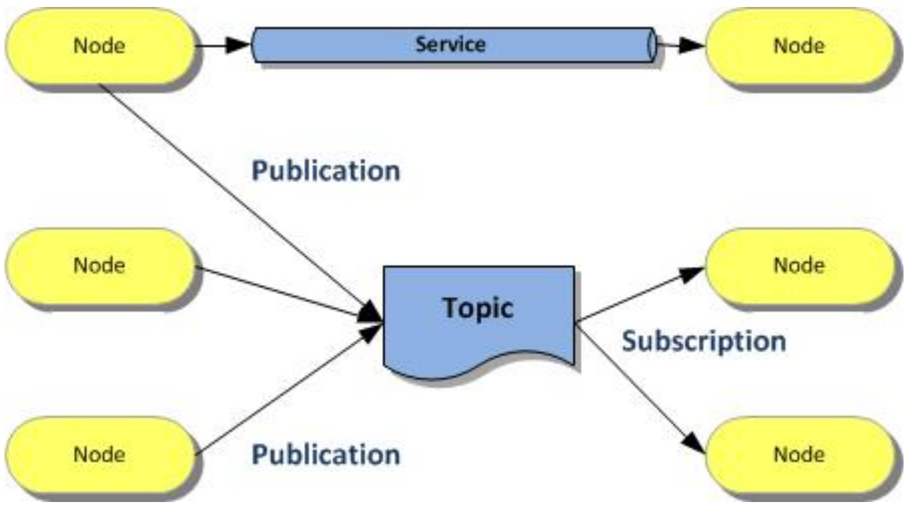

Fig.C.1 ROS structure of the basic elements (image from generationrobots.developpez.com)

Nodes

Nodes are executable files within ROS packages. They represent processes in a ROS network. Nodes use ROS client library to communicate with each other. They can publish or subscribe to a *Topic* and provide or use a *Service*.

Parameters

During runtime there are data related to the configuration and initialisation stored in the Parameter Server called *parameters*.

Topics

Topics can be perceived as data streams, asynchronous many-to-many ? between nodes. It should be used for continuous data streams like sensor data or robot state.

Messages

As computer languages have supported data types, ROS supports *messages* used when subscribing or publishing to a topic.

Services

ROS nodes can exchange data not only using topics but also using *services*. Essentially, they are synchronous one-to-many network-based functions. *Services* should be used for remote procedure calls that terminate quickly. For instance, a *service* can be utilized for querying the state of a node or doing a quick calculation such as inverse kinematics (IK). Not using them for longer running processes is recommended. Furthermore, *services* cannot support processes that might be required to preempt if exceptional situations occur.

Actions

In cases where the service takes a long time to execute, the users might want the ability to cancel the request during execution or get periodic feedback about how the request is progressing. *Actions* provide the means to build specific servers that execute long-running goals that can be preempted. Moreover, it also provides client interface in order to send the related requests. Thus, if a server is executing two action goals simultaneously, for each client a separate state instance can be kept since the goal is uniquely identified by its id.

C.4 ROS development approach

In the "Simulation" Part of the dissertation I have clarified the reason why the ROS framework is going to be employed with the integration of the Gazebo 3D simulator for the project's simulation. On top of them, the Hector quad-rotor will form the basis for my UAV with various sensors, customised as required. Even if the users have reached this stage of decision-making, the way in which they are going to work is not predefined. This is owed mainly to the way that ROS is built, giving the users complete freedom to implement nodes as long as they can conform to the communication protocols.

My first attempt to start developing ROS system nodes for designing simple robotics algorithms was to use Matlab's robotics system toolbox. This is new feature of Matlab (R2015a). Robotics Toolbox provides an interface between MATLAB and Simulink and ROS that enables applications testing and verification on ROS-enabled robots and robot simulators such as Gazebo. It supports C++ code generation. Thus, it enables the generation of a ROS node from a Simulink model and deployment to a ROS network. This is a great addition to the strict ROS terminal based interface. Matlab suite and scripting language is developer friendly and it can boost the research and productivity in higher levels.

The way that Matlab and ROS work is based mainly on the TCP/IP communication (fig.C.2). Although Matlab can autonomously execute ROS master and host the entire ROS ecosystem, most of the time users need to experiment with third-party ROS packages. To host these needs, it is assumed that Matlab is installed in a windows or mac host computer. To execute external ROS master, Matlab directs the users to download an Ubuntu virtual machine. After that, the users can normally execute any package in ROS master in the virtual machine and have access to topics and services from the Matlab in the host computer.

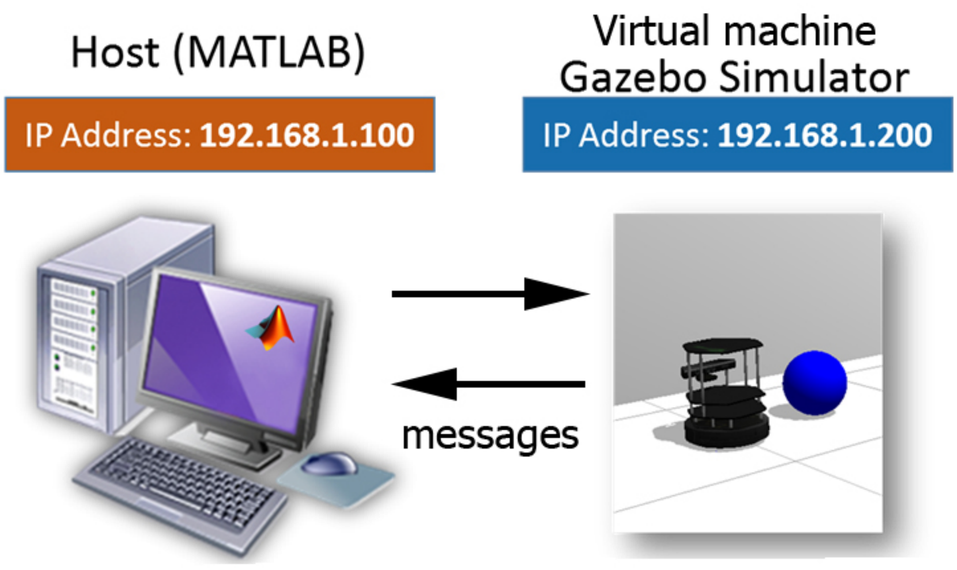

Fig.C.2 Interface between Matlab and ROS (image modified from www.mathworks.com)

The proposed way of work is convenient when there is a need to process data obtained from ROS nodes, transmitted using the topics. However, it is not straightforward to deploy more complex algorithms. From my experience in working with it, although I followed successfully all the learning material from the Matlab knowledge base, using the classic *turtlesim* simulation, when I attempted to get data from the Hector simulator I encountered problems. More precisely, Matlab couldn't receive laser scanning from hector quad-rotor. In addition, the set up of working in two different places at the same time makes the workflow more complicated. In my opinion, the interface between Matlab and ROS is a very strong combination, which can exploit the advantages of both successful platforms. Nevertheless, the implementation was just released and it needs time to mature with bug fixes and optimization.

For my project, I wanted a simpler approach. I preferred a setup implemented with fewer stages in order to eliminate potential point of failures. The solution to that was to work directly on a ROS installation in Ubuntu using the terminal and a simple text editor. In my first attempt I utilized python language, for some people the free counterpart of Matlab. Python is a high level programming language, with enhanced readability. It is an interpreted language rather than compiled like C++. This means that in ROS the users merely have to develop scripts without caring about the settings in the *CMakeList.txt* file which configures the compilation. This is by itself a great advantage since these settings require a degree of experience to make them work correctly and efficiently.

The use of Python in ROS is officially supported and facilitates complex developments making the robotics rapid prototyping a reality. The syntax allows programmers to express concepts in fewer lines of code that it would be possible in languages such as C++. However, this level of abstraction and simplicity comes at a cost. From my experience, although the scripts were interpreted without any errors, the applications did not work. This is because there were errors in my code that were not reported from the lenient python interpreter.

The native ROS development language is C++. Using this language the users can develop everything they want in the ROS ecosystem. It requires having good programing skills in object oriented programming in C++. To compile the code the users should manually edit the settings in the *CMakeList.txt* file and *package.xml* file of the package. It is particularly hard at the beginning and even a simple task could take considerable time to accomplish. When the users acquire the needed experience, everything is simpler. On the other hand, the compiler is very strict compared to the python interpreter. If the users manage to compile the application successfully it is guaranteed that is going to work, unless it has logical errors. In addition, the compile error messages are self-explanatory and direct the users to the specific error. Finally, the syntax might not be so friendly like higher-level languages but it is concise and integrates powerful structures and tools. In short, C++ on a ROS installation was the development method that I adapted since it was the most straightforward and simpler, in terms of configuration, approach I could apply.

C.5 UAV manipulation input devices

By and large, the control of the UAV can be implemented by designing a publisher that transmits messages of type *geometry_msgs/Twist* in the *cmd_vel* topic. A recommended way to teleoperate the UAV is to use a joystick. The hector quad-rotor comes with a node that is configured to utilize the xbox controller for teleoperation. In order to achieve continuous movement, which is disabled for safety reasons, a specific parameter, *autorepeat_rate*, should be enabled and set to the required rate. The edited launch file can be found in (goo.gl/b63Ugr). Further than that, the throttle should also be adjusted to move the UAV smoothly. The required changes in the teleoperation node to achieve this can be found in (goo.gl/K173HW).

There is also a simpler way to control the UAV using just the keyboard. To use the keyboard for the UAV manipulation the users should install an appropriate package that includes the required nodes. One option is to use the ROS package (pr2_apps) developed by Willow Garage for the PR2 robot. Once the package has installed successfully, by executing the *teleop_keyboard.launch* file the users can control the translation on the x and y axis, the yaw angle and the speed.

C.6 System configuration

The proper setup of the working environment in ROS and Gazebo is a very hard task. Factors that need to be considered when the system is configured are the following:

- Operating system
  It is recommended to use Linux. Actually, Linux is the only OS that is officially supported in ROS. The installation debian packages (unix archives) for ROS are available for Ubuntu Linux.
- ROS version
  Following a developing cycle similar to Ubuntu, ROS releases new versions every 6 months to one year. At the time of the project implementation, the latest was Jade (May 2015). However, because of compatibilities issues a hydro version was used. This version was released in September 2013. Making this decision I was restricted to install Ubuntu 12 LTS (Long Term Support) as operating system in order to gain maximum compatibility.
- Gazebo version
  Since ROS Hydro, Gazebo is considered a system package instead of a ROS package. The practical meaning is that one major version of gazebo is selected at the beginning of the ROS release cycle and remains the same during the whole life of the ROS distribution. In my case ROS Hydro is compatible with Gazebo version 1.9.
- Hardware
  Hardware is also a parameter in simulation system. In case that the ROS is combined with Gazebo, there is a need for at least 4GB RAM. Gazebo requires a working graphics card with OpenGL 3D accelerated driver to perform various rendering and image simulation tasks correctly. For the current project, an extensive use of virtual machines was engaged. Particularly, I utilize both VMware Fusion and VirtualBox software supervisors. Moreover, there are available virtual machines with pre-installed ROS distributions, which could be easier for the beginners. All in

all, both virtualisation methods performed well, supporting the ROS ecosystem.

To summarize, my setup configuration based on virtual machines having Ubuntu 12.04 and 14.04 as operating system using ROS Hydro and Indigo respectively.