12º WORKSHOP

ROBÓTICA COGNITIVA

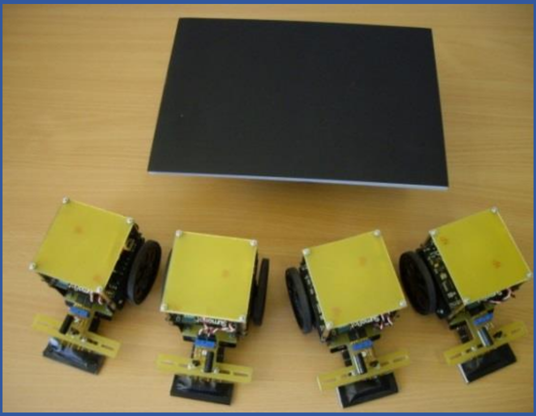
RoboCity2030

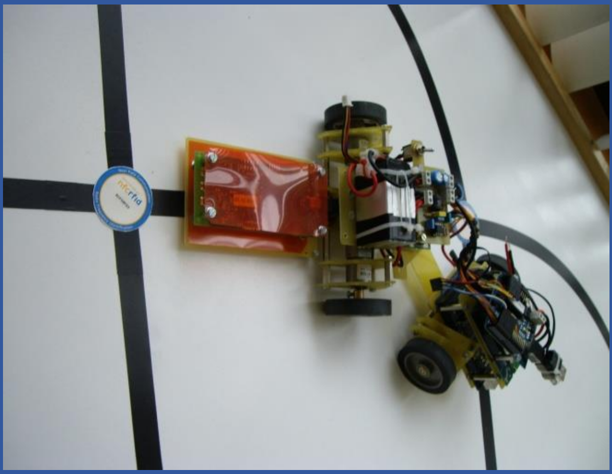
Universidad Nacional de Educación a Distancia

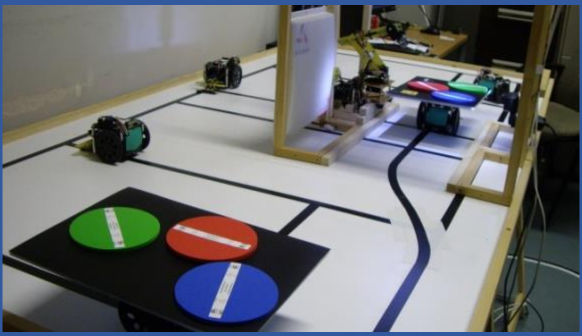
Julio 2013

Editores

Carlos Cerrada
José A. Cerrada
Enrique Valero
Ismael Abad







Robótica Cognitiva

Editores

Carlos Cerrada
José A. Cerrada
Enrique Valero
Ismael Abad

Editado: Universidad Nacional de Educación a Distancia (UNED)

Madrid, Julio de 2013

Edita:
Universidad Nacional de Educación a Distancia (UNED)
Imprime:
Universidad Nacional de Educación a Distancia (UNED)

D.L. M-20260-2013
ISBN: 978-84-695-8175-9

CAPÍTULO 1

RECENT ADVANCES IN THE JDEROBOT FRAMEWORK FOR ROBOT PROGRAMMING

JOSÉ M. CAÑAS[1], MAIKEL GONZÁLEZ[1], ALEJANDRO HERNÁNDEZ[1] and FRANCISCO RIVAS[1]

[1] Universidad Rey Juan Carlos, Spain

Most of the intelligence of cognitive robots lies on their software, the way they manage knowledge and coordinate their sensing and actuation capabilities. In the last years several frameworks have appeared that simplify and speed up the development of robot applications. They favor the code reuse and take benefits from modern software engineering techniques.

This paper presents recent advances in the open source robotics framework Jderobot (http://jderobot.org). It is a distributed and component oriented software architecture which uses explicit interfaces among components and ICE as communication middleware. Jderobot provides many tools for the programming of cognitive robots: a template for controller components, visual Finite State Machine creation, a library for fuzzy controllers, recording and replaying sensor data, etc. It also includes tools to use new RGB-D sensors like Kinect or Xtion and support for the newest release of Gazebo simulator, which is becoming a *de facto* standard. The main new features and tools are described in detail.

Jderobot framework has more than 60000 lines of code and includes more than 30 different components. Two recent improvements are related with a better project management using CMake and simpler installation using Debian packages. This software has been used for more than ten years in research, teaching and commercial applications. More than one hundred fifty different users, mainly robotic students and developers, have taken advantage of it. Lessons learnt using this robot middleware are also summarized in conclusions.

1 Introduction

Most of the robot intelligence lies on its software. Once the robot sensor and actuator devices are set, the robot behavior is fully caused by its software. There are many different ways to program in robotics and none is universally accepted. Some choose to use directly languages at a very low level (assembler) while others opt for high-level languages like C, C++ or Java.

Good programming practices are an emerging field in the software engineer area but also in robotics. Several special issues of robotics journals (ARS Special Issue on Software Development and Integration in Robotics, 2006), books on the topic have been published (Kernighan, 1999) and also specific workshops and tracks have been created inside ICRA and IROS. The Journal of Software Engineering for Robotics (www.joser.org) has been promoting the synergy between Software Engineering and Robotics meanwhile the IEEE Robotics and Automation Society (TC-SOFT) have founded the Technical Committee for Software Engineering for Robotics and Automation.

Compared with other computer science fields, the development of robot applications exhibits some specific requirements. First, liveliness and real-time processing: software has to take decisions within a fast way, for instance in robot navigation or image processing. The second requirement is about the software architecture; robot software has to deal with multiple concurrent sources of activity, and so tends to be multitasking. Third, computing power is usually spread along several connected computers, and so the robotic software may be distributed. Fourth, the robotic software typically deals with heterogeneous hardware. New sensors and actuator devices continually appear in the market and this makes complex the maintenance and portability to new robots or devices. Fifth, the robotic software usually includes a Graphical User Interface (GUI), mainly for debugging purposes. Sixth, the robotic software should be expansible for incremental addition of new functionality and code reuse. Seventh, the simulators are very useful in robotics software debugging.

Mobile robot programming has evolved significantly in recent years, and two approaches are currently found. In the classical approach, the application programs for simple robots obtain readings from sensors and send commands to actuators by directly calling functions from the drivers provided by the seller. In the last years, several frameworks (SDKs) have appeared that simplify and speed up the development of robot applications, both from robotic companies and from research centers, all of them with closed and open source. They favor the portability of applications between different robots and promote code reuse.

First, they offer a simple and more abstract access to sensors and actuators than the operating systems of simple robots. Using the SDK hardware abstraction layer it deals with low level details accessing to sensors and actuators, releasing the robotics programmer from that complexity.

Second, the SDK provides a software architecture for robot applications. It offers a particular way to organize code, handling of code complexity when the robot functionality increases. There are many options: calling to library functions, reading shared variables, invoking object methods, sending messages via the network to servers, etc. Depending on the programming model the robot application can be considered an object collection, a set of modules talking through the network, an iterative process calling to functions, etc.

Third, usually the SDK includes simple libraries, tools and common functionality blocks, such as robust techniques for perception or control, localization, safe local navigation, global navigation, social abilities, map construction, etc. This way SDKs shorten the development time and reduce the programming effort needed to code a robotic application as long as the programmer can build it by reusing the common functionality included in the SDK, keeping themselves focused in the specific aspects of their application. The robot manufacturers sell them separately or include them as additional value with their own SDKs. For example, ERSP includes three packages in the basic architecture: one for interaction, one for navigation and another for vision.

We present our open-source robotic software framework, named Jderobot, which is component oriented, uses ICE as communication middleware and includes several useful tools and libraries. Several sensor and actuator drivers have been programmed or reused from the open-source community. This framework has been widely used in our group for research and teaching for more than ten years. Jderobot has been designed for scenarios with sensors, actuators and intelligent software in between.

The typical scenario is robotics, but also computer vision and home automation.

Why open-source in robotics research? It provides independence on robot manufacturers and so it may support robots from different companies. With the freedom to use and modify the software its final quality does not depend so much on the debugging speed of a single company. The distribution of the source code makes (the distribution) debugging faster. In research, algorithms and results can be replicated and compared. Standard tools (for instance, simulators) and public access to sensor and data repositories (for instance, databases with input data and ground truth for robot localization), etc. help on this. This sharing makes the improvements come more easily and speeds up the advances in the robotics community. In addition, one strong motivation is the feeling of contributing to the community and returning the favor. We have extensively used open-source libraries and tools in our research (OpenCV, Gazebo, GTK, etc.).

Jderobot has enhanced thanks, in a large part, to the robotics community of the Universidad Rey Juan Carlos (Spain). New versions have been releasing including new functionalities. The last version, which is the 5.1, includes new applications, tools to facilitate and improve the management and features to make easier both the use and installation to standard users and also to developers. The new features to highlight on this version are the support to newer versions of the Gazebo simulator. This support is implemented by the gazeboserver driver which acts as an intermediary between the simulator and other applications though ICE interfaces. Another improved component is introrob, a teaching tool successfully used in the robotics subject in the Universidad Rey Juan Carlos.

Tools like CMake (see section 7.1) are also included in the new version of Jderobot. The use of this powerful tool makes the compilation task of all the framework libraries, components and interfaces so simple. In addition, Jderobot has also a set of Debian packages allowing to the user to install each of the components either individually (using atomic packages) or jointly (using virtual packages).

2 Related works

Robotic frameworks can be grouped in two main paradigms, those tightly coupled with a cognitive model in their designs and those designed just from a pure engineering criteria. The first ones force the user to follow a set of rules in order to program certain robotic behavior, while the second ones are just a collection of tools that can flexibly be put together in several ways to accomplish the task.

Cognitive robotic frameworks were popular in the 90s and they were strongly influenced by the Artificial Intelligence (AI), where planning was one of the main keys. Indeed one of the strengths of such frameworks was their planning modules built around a sensed reality. A good example of cognitive frameworks was Saphira (Myers, 1998), based on a behavior-based cognitive model. Even though the underlying cognitive model usually is a good practice guide for programming robots, this hardwired coupling often leads the user to problems difficult to solve when trying to do something the framework is not designed to do.

Key achievements of modern frameworks are the hardware abstraction, hiding the complexity of accessing heterogeneous hardware (sensors and actuators) under standard interfaces, the distributed capabilities that allow to run complex systems spread over a network of computers, the multi-platform and multi-language capabilities that enables the user to run the software in multiple architectures, and the existence of big communities of software that share code and ideas.

Current robotic frameworks focus their design on the requirements that robotics applications need and let the users (the programmers) to choose the organization that better fits with their specific application. Main requirements driving the designs are: multi-tasking, distributed, easy to use and code reusability. Another requirement, probably the main key, is the open source code. That creates a synergy between the user and the developer.

That is why two of the most popular robotic frameworks in the last years are open-source: Player/Stage (Gerkey, 2003) which has been the standard *de facto* in most of the last decade and ROS (Quigley, 2009) which is taking the place currently. As seen in other major software projects as GNU/Linux kernel or the Apache web server, to name but a few, the creation of communities that interact and share code and ideas, could be a great benefit to the robotic community. Main examples of open source modern frameworks are the aforementioned Player/Stage and ROS. Another important example is ORCA (Brooks, 2005). We briefly describe them.

There are other open source frameworks that have had some impact on current the state of the art, like RoboComp (Cintas, 2011) by Universidad de Extremadura, CARMEN (Montemerlo, 2003) by Carnegie Mellon and Miro by University of Ulm. All of them use some component based approach to organize robotic software using ICE, IPC and CORBA, respectively to communicate their modules.

We can find non open source solutions as well, like Microsoft Robotics Studio or ERSP by Evolution Robotics. Those are also very powerful platforms but their main disadvantage is just the open-source advantage: you cannot share code with other robotics groups without a license of the corresponding software.

2.1 Player/Stage

This framework provides a robotics environment dividing it into two sides: Player which is a robot device server and the multiple robots simulator named Stage. To support the development of robotic applications this suit also includes several tools and libraries. Player/Stage has been the most popular framework in the last decade with a big community giving support and also developing a great collection of drivers and algorithms around it. It is completely platform independent and most common programming languages, like C/C++, Python or Java, are supported.

Player provides a network interface to the robot hardware through a collection of standard interfaces that provides a hardware abstraction layer. These standard interfaces are implemented by drivers, one for each different hardware. Following this idea, the user only needs to know the standard way to use of each type of device not every device of every different manufacturers or models. Player exports its devices through a standard TCP network connection (other transport layers are available as well) enabling the user to build distributed systems across a network of computers. Even though, Player was not designed as a component based software, its architecture employs many component based ideas. In addition to hardware drivers, Player has a collection of algorithms like local navigation algorithms, vision related algorithms or localization algorithms.

Stage is a 2D multiple robot simulator that can provide its simulated devices through a Player server. There is a big variety of devices Stage can simulate, like laser rangers, robotic platforms and cameras, to name a few.

The typical organization of a robotic application written in Player/Stage is one or more client programs subscribed to a Player server which are serving the robot hardware. Clients send and receive the data using the known interfaces. So, we can say Player/Stage has a centralized architecture, where Player server assumes the main role.

To implement a new functionality on the server side the user can write a new driver which implements an interface. This is not usual due that developing on this side is a bit harder than on the client side.

2.3 ORCA

Another important example is ORCA which is a component based framework released several years before ROS. ORCA is multi-platform and multi-language as well. The aim of its developers was to increase the software reuse among the robotic community, so they create a component repository for robotics applications and they provide some of the needed glue to connect them. In a previous version, they coded the middleware that enabled components to communicate. Later, they realized that programming a middleware was out of the range of their interests, besides being a complex and time consuming task, so they replaced it with the professional grade middleware Ice from ZeroC (Henning, 2004) that allowed them to focus in the robotic problems. ORCA is the major source of inspiration for the actual version of the Jderobot and some of its core components are closely related to our framework.

2.2 ROS

ROS is one of the biggest frameworks nowadays. It was founded by Willow Garage as an open source initiative. Currently, it has a growing community and its site hosts a great collection of hardware drivers, algorithms and other tools. It is a multi-platform and multi language framework.

The main idea behind ROS is an extremely easy to use middleware that allows connecting several components, implementing the robotic behavior, in a distributed fashion over a network of computers using hybrid architecture. ROS is developed under hybrid architecture by message passing and RPCs. Message passing of typed messages allows components to share information in a decoupled way, where you do not know which component send you a message, and vice versa, you do not know which component or components will receive your message. RPC mechanisms are available as well. Resources can be reached through a well defined naming policy.

The ROS core libraries implement the communication mechanisms and a set of tools to help with tasks as project management (CMake based), system debugging and centralized logging. A set of official libraries implements standard messages (sensors, geometry, action...), well known robotic algorithms for navigation or sensor analysis, and powerful tools as the Rviz 3D visualization environment for robots. A simulation environment is provided through the Gazebo 3D simulator, which started as part of the Player/Stage project, but now is supported by Willow Garage.

A typical application written in ROS is a collection of components interacting with each other (no server/client model) distributed among a network of computational nodes. Often a user can just use some of the components found in the ROS repositories, tweak some parameters and make them interact with his own coded components.

3 Jderobot

The Jderobot platform is a component based framework that uses the powerful object oriented middleware Ice from ZeroC as *glue* between its parts. This important design decision allows Jderobot to run in multiple platforms and to be programmed with the most common programming languages (all the languages supported by Ice indeed). Jderobot components can also be distributed over a network of computational nodes and by extension use all the mechanisms provided by Ice as secure communications, redundancy mechanisms or naming services.

3.1 Design principles

Following current trends in robotic software engineering, we aimed to design a framework with these major requirements:

- Component based.
- Multi-platform and multi-language support.
- Distributed.
- Strongly typed interfaces.
- Open source.

The main unit is the *component*. A component is an independent process which has its own functionality, although it is most common to combine several of these in order to obtain a more complex behavior. There are several types of components, according to the functionality of the component could be classified in:

- Driver-components: offer to developers a HAL (Hardware Abstraction Layer) to communicate with the different devices that are equipped by the robot (sensors and actuators).

- Tools-components: This kind of components uses the driver-components to communicate with the robot (real or simulated) and process the received data. Then the result of this process is sent to the robot through the driver-component.

The communication between Jderobot components occurs through the ICE (Internet Communications Engine) middleware. Using this concept, Jderobot allows directly these components to be distributed as it uses the network as a link. The ICE communication is based on interfaces. This middleware has its own language named *slice* which allows the developer to define their custom interfaces. Those interfaces are compiled using ICE built-in commands, generating a translation of the slice interface to various languages (Java, C++, Python…). This allows communication between components implemented in any of the languages supported by ICE.

The entire configuration needed by the components is provided by its configuration file.

Another important feature of Jderobot is the wide use of third party software and libraries (all of them open source) which greatly extends its functionality. Among them include: OpenCV for image processing; PCL for point cloud processing; OpenNi for the RGB-D support, Gazebo as the main 3D robot simulator and GTK+ for the GUI implementations. Besides, Jderobot provides its own libraries which give to robotics commonly used functionality for the developing of applications under this framework.

3.2 Open source project

As project, Jderobot stands out for being maintained by a community of developers formed largely by the Robotics Group at the Universidad Rey Juan Carlos. As a source of documentations, this framework has its official wiki (http://jderobot.org/index.php/Main_Page) where everyone is able to download the entire project. To complete the support, a mail list is also available to interact with other developers.

Jderobot is a platform that has evolved significantly since its inception and it is currently at 5.1 version. The following sections describe some of the latest developments.

4 VisualHFSM and fuzzy controllers

There are many ways to organize the control and perception code on board a mobile robot. One successful way is the Finite State Machines (FSM). With them the robot behavior is defined by a set of states, each of which performs a particular task. The robot can switch from one state to another through transitions (conditions of stay or change), depending on certain events or conditions, internal or external. A tool has been developed and included in Jderobot to graphically design hierarchies of FSM, insert the specific code of states and transitions, and automatically generate the component source code in C++.

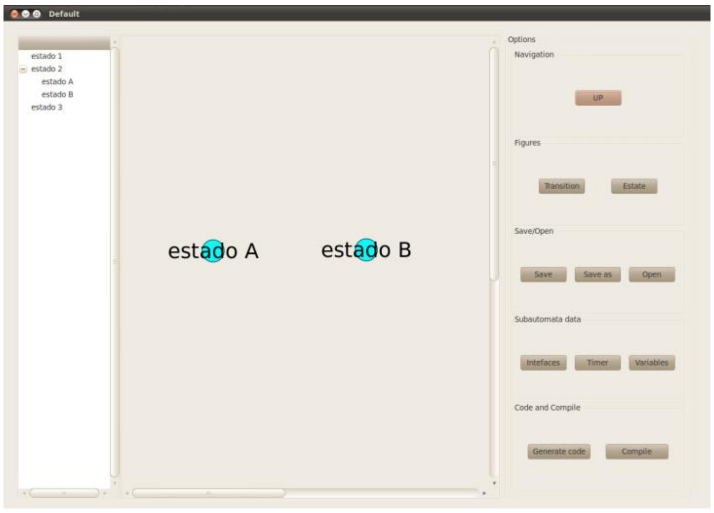


Fig. 1. VisualHSFM Graphic User Interface

Another tool when programming cognitive robots is the fuzzy logic. It allows programming the robot behaviors in terms of simple words and rules which are close to the natural language. In Jderobot we have developed the Fuzzylib library to design and program fuzzy controllers. The control rules are written in a file with simple rules like:

IF (left_obstacle_distance = small)
AND (advance_speed = high)
THEN (rotation_speed = right_high)

The fuzzy labels and variables are also described in such file following a trapezoidal pattern and they are linked in the software to input and output variables of the control program. The controller automatically fuzzyfies the input variables, apply the rules and defuzzifies the output following a Center of Mass combination.

5 Gazebo support

Gazebo[1] (Koening, 2004) is the preferred simulator in Jderobot framework. It is a 3D open source simulator which offers a rich environment to quickly

[1] http://gazebosim.org

test multirobot systems and which simulates several robots and sensors (such as cameras, laser, etc) in a realistic way. All simulated objects have their own mass, velocity, frictions and numerous other attributes that allow them to behave realistically when pulled, knocked over or pushed.

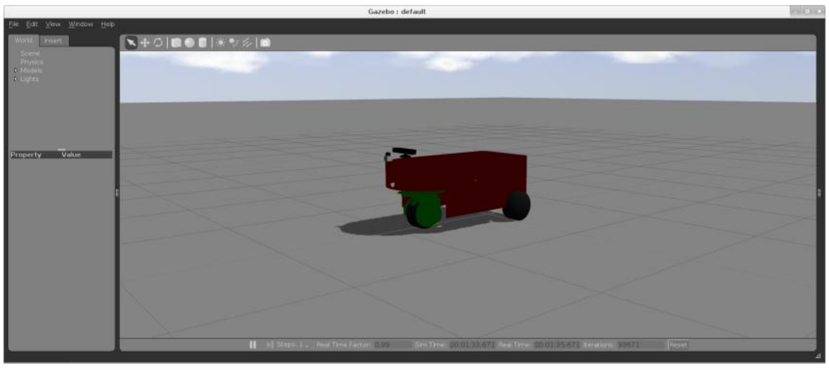


Fig. 2. Gazebo with Autonav robot.

Recently Defense Advanced Research Projects Agency (DARPA) announced its Robotics Challenge for disaster robots. The goal of this challenge is to develop ground robots capable of executing complex task in order to extend aid to victims of natural or man-made disasters and conduct evacuation operations. Selecting Gazebo simulator as the standardized simulation environment and will be provided by the Open Source Robotics Foundation. The teams that do not want or cannot afford to build their own robots will be able to prove themselves using Gazebo simulation environment and later may receive a real robot to use in the competition. The choice of this simulator is successful since it has become a standard in one of the main research project around the world.

Gazebo offers a big variety of sensors, actuators, robots, objects and maps. Also it incorporate tools to design new elements, which integration with the simulator is very simple. Fig. 2 shows a robot integrated in the simulator. This robot is defined with a drive and steering wheel and three sensors such as Kinect, laser and encoders. In the world files is possible to define the simulated world and the different elements that defined the environment. This allows creating a simulated world very close to the real environment where the robot is going to be deployed.

Jderobot 5.1 offers support for the last version of this simulator (Gazebo 1.8). It offers a driver-component call gazeboserver who intermediates between Gazebo and the components in Jderobot 5.1. The architecture of gazeboserver is based on plugins. A plugin is a dynamic library load in execution time when the simulator starts. In each plugin, the functionality of the different devices, what are contained in the robot, is defined. The plugins that the simulator should load must be indicated in the configuration world file.

Plugins are not only connected with the simulator, they also create an ICE interface which allows communicating the simulator with Jderobot components, sending or receiving information about the robot state. Fig.3 shows an example of a component connected with gazeboserver. We can observe that the components of the system are connected to the platform without taking into account if we are using a simulator or the real robot.

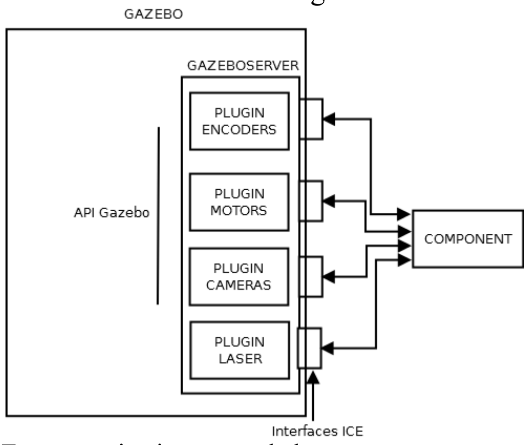


Fig. 3. ICE communication example between a component and Gazebo

6 Introrob component for teaching robotics

Introrob is a teaching tool that Universidad Rey Juan Carlos master students use to develope their practice into the Robotics subject. This component is connected to the Gazebo simulator using gazeboserver. The simulated robot used with introrob is the pioner2dx with a set of sensors (laser ranger, odometry and cameras) and actuators (motors, and cameras Pan&Tilt).

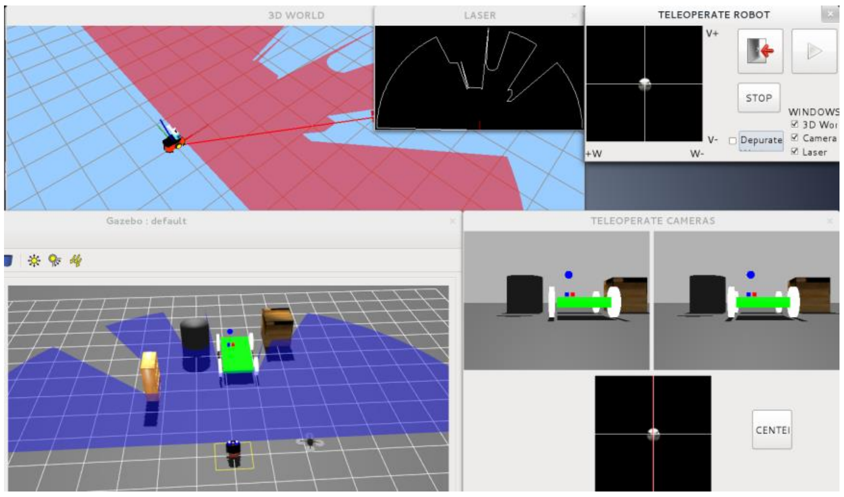


Fig. 4. Introrob running on Gazebo

This component provides to the students a GUI (Fig. 4) where they can find all the data collected from the sensor of the simulated robot. This GUI also includes some features to make easier the debug of their own algorithms. It also gives some functionality to teleoperate the robot using some joysticks changing the linear and angular speed as well as changing the cameras orientation. In addition, introrob shows a 3D viewer based on openGL from which it is possible to perform certain interactions (change the point of view, draw objects, etc…). This GUI also includes a play button to start a certain algorithm developed by the student.

For the algorithm developing, there is available a file called MyAlgorithms.cpp which is launched when pressing the PLAY button. In this way, the student can be abstracted from the rest of the application code and focus only on a single file. Besides, this file also contains a template with examples that allows students to speed up their learning with the tool.

Introrob has also an own API that simplifies even more the developing task to the students offering fully abstract functions. They do not need any prior knowledge about the communications protocol nor data structure. Some examples of this API functions are:

- setV(5): receives as a parameter the speed in mm/s and it is sent to the robot.
- getLaser(): returns a 180 position vector with all the measurements of the laser range scanner.

It also provides some graphical functions to simplify the visualization of their algorithms results.

- drawSphere(x,y,z): receives a 3D point and draws a sphere there in the OpenGL world.
- drawSegment(p1,p2): receives two 3D points and draws a segment that joins them.

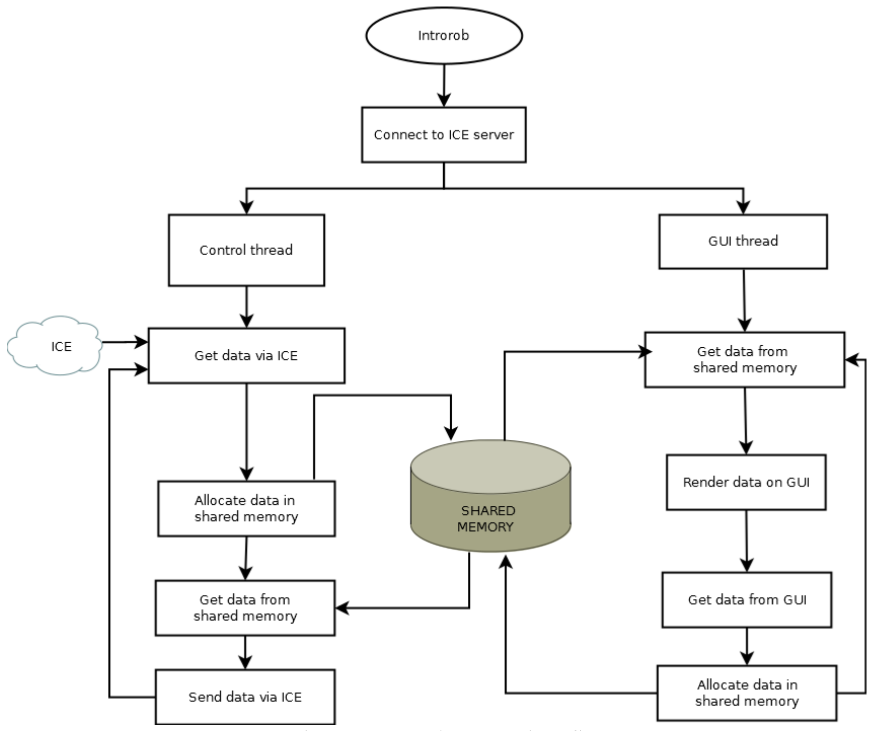


Fig. 5. Introrob execution flow

Internally, the architecture of this component is divided into two main threads: control thread and GUI thread.

- Control thread: this thread asks gazeboserver for the data from the sensors of the simulated robot and locates them in shared memory. Then, it gets the data generated by the GUI thread from shared memory and sends the corresponding orders to gazeboserver.

- GUI thread: this thread is always taking the data allocated by the control thread and displaying it in the GUI. Besides, it puts again in shared memory the data generated, or by the user interaction or their code, to send them to the simulator through the control thread.

Students from Computer Vision (Master degree) use introrob to develope algorithms oriented to the recognition of the environment using cameras carried by a robot. Furthermore, Telematic and Computer Systems (Master degree) students are focused on navigation algorithms with the aim of recognizing and following a line drawn on the floor.

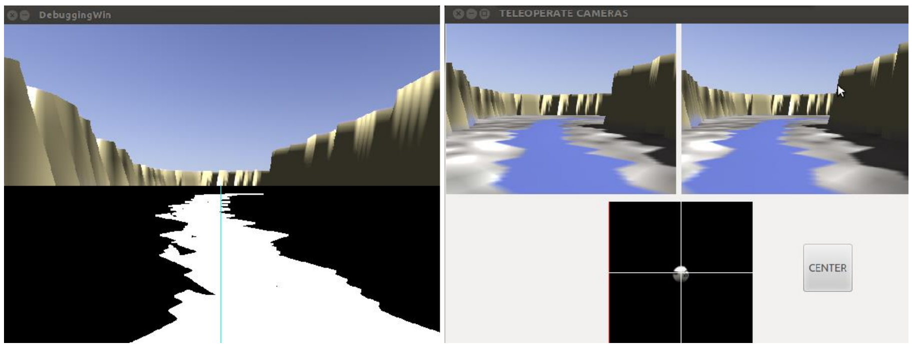


Fig. 6. Student's example algorithm

7 CMake and Debian packages

Jderobot 5.1 has new features that facilitate the management and maintenance of the entire platform and its source code. On the one hand, Jderobot includes the new CMake tool that lets you build simply the entire source code of the platform. On the other hand, it also has a set of Debian packages to simplify the Jderobot installation, both the platform itself and all their third part dependences. These packages are available for the two Linux distributions: Ubuntu 12.04 and Debian Wheezy, both for the 32 bits version.

7.1 CMake

CMake is a build software tool that makes easier the multiplatform code compilation both for users and developers.

At the user level, a feature that highlights is the information it provides about what is happening during the compilation process. This helps the user to resolve any problems that arise. But mostly, the main advantage using CMake is its easiness. To compile a whole project (regardless of the complexity or size) the user has only to execute two commands: *cmake* and *make*.

At developer level, CMake provides agility when adding new items to project and it needs too few configurations to compile and link a lot of software.

The CMake configuration is based on CMakeLists.txt files. These files include everything needed to build the code. The designed solution in Jderobot divides its configuration into two different types of CMakeLists files:

- Type 1 or primary: these files are used directly from the *cmake* command and are the responsible for initiating the process of building. Into them are defined common characteristics for all elements to be compiled and linked (libraries, interfaces and executables).
- Type 2 or secondary: these files are called by other primary files and/or secondary files recursively. They define the relevant data of each element, such as:
  + Source files.
  + Dependences.
  + Location of the dependences (both, headers and libraries).

Following this design, Jderobot 5.1 offers the user three different ways to build the code:

- Compilation from the trunk. The compilation process is initiated by the main CMakeLists.txt located at the top of the trunk project directory. This will build all components, libraries and interfaces of the platform.
- Compilation by components. This second way allows the users not to build all the elements, but those they only need. To do this, the compilation must to start from the main CMakeLists.txt file located in the /build directory of the wanted component. This method requires that the directories hierarchy of the platform has to be respected. This is because the paths used for dependency resolution are defined relatively.

- Independent compilation. This allows the user the ability to download only the source code of a particular component and build it without downloading the entire tree directory of the whole platform. In this case, the prerequisite to build the executable is that the users must have installed on their computer all the libraries on which the component depends.

7.2 Debian packages

Debian packages are compressed files that contain the elements to install on our computer, information on how and where to be installed and information about their dependencies.

There is much heterogeneity among the different systems of students and Jderobot users. This complicates the installation of software consisting of a large number of applications which depends in turn on external software. Debian packages facilitate this task by reducing installation to simple commands allowing the user to have all the necessary software automatically.

For the management and installation of Debian packages, there are tools like *apt* or *aptitude*. Using these commands any user can download and install a package from a repository, resolve its dependencies or uninstall it very easily later.

These tools enhance the user's first experience with the platform, making friendlier their first contact with it.

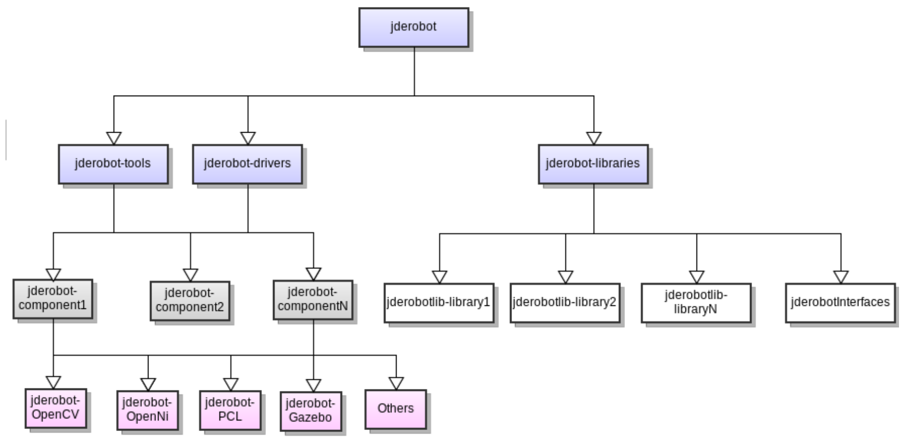


Fig. 7. Jderobot package design

Packet structure designed for Jderobot 5.1 (Fig. 8) is divided into:

- Atomic packages. Jderobot has a package for each of the components (jderobot-componentN) and libraries (jderobot-libraryN) that make up the platform. This allows the users to install, individually, those parts their want to use or develop.

- Third part packages. To make easy the installation, not only the components but the entire external software, Jderobot includes Debian packages for that software used by the components and which is not officially available in the repositories (Ubuntu 12.04 or Debian testing).

- Virtual packages. Virtual packages are packages that do not contain the elements to be installed in our system, but information about which atomic packages should be installed when you install one of them.
Jderobot 5.1 has different virtual packages that encapsulate:

+ Libraries: All Jderobot libraries are grouped into the virtual package jderobot-libraries.
+ Driver-components: jderobot-drivers.
+ Tools-components: jderobot-tools.
+ Components with related functionality, as teaching-robotics which includes introrob, gazeboserver and gazebo.
+ jderobot package. This installs all atomic packages and third party software packages that make up the entire platform.

8 Conclusions

We have presented the Jderobot open-source framework for programming cognitive robots. We have used it in teaching, development of robotic applications and research for more than 10 years. Its software architecture is based on distributed components using the network as communication link. It provides several drivers for accessing different sensors, actuators and robots like Pioneer2DX or Nao humanoid. It also includes a powerful set of tools and libraries that speed up the creation of new robotics applications. In this paper we have described the main new features developed for the Jderobot-5.1 release.

First, some cognitive tools have been incorporated to the framework. The visualHFSM provides a tool for generating robot behaviors using hierarchical Finite State Machines. It lets the programmer to focus on the behavior logic, in terms of states and transitions, more than on implementation details. Most of the final control code is generated automatically by visualHFSM using an automata template and an abstract description of the FSM. Another cognitive tool incorporated is the fuzzy library that eases the programming of fuzzy controllers.

Second, we have improved the connection between Jderobot and the latest releases of the Gazebo simulator. This open source simulator is becoming a *de facto* standard after the ROS developer team chose it as its reference simulator. More recently DARPA chose it for the first stage of the DARPA Robotics Challenge. We have developed a set of new Gazebo plugins that provides the standard Jderobot interfaces to sensors and actuators in the simulated world. The model of a brand new robot has been also created in the new Gazebo.

Third, we have improved the Introrob component that we use in robotics teaching. This Jderobot component is the base for the practices of students from several engineering and master courses. More than two hundred students in the last years have used Jderobot in the last years. They serve as testers giving feedback of the performance of the framework to the developers and providing experimental validation to the system. The specific Introrob component hides some of the complexity of robot programming, and so the students take shorter time to start using the framework, focusing on the robotics algorithms and techniques more than on the platform itself.

And fourth, we have changed two issues of the infrastructure of the whole Jderobot as a software project. We distribute it now as debian packages, both for Debian Linux and Ubuntu LTS. The creation of debian packages makes easier the installation and management of the platform. We have created atomic packages for each Jderobot component, library or tool. We have also prepared several virtual packages that include those atomic packages that make sense together. In addition, the last Jderobot release uses CMake as the project compilation tool, making simpler the inclusion of both new components and libraries. It has been an improvement from the previous compilation tools, automake and autotools, which are more complex.

We are currently working on several lines to extend Jderobot. First, to improve the use of ICE advanced capabilities like Icestorm and Icebox to take benefit of them. Second, on the integration of the new RGB-D commercial sensors (Kinect, Xtion, Primesense sensor) which will boost the developing of reconstruction and localization applications using point cloud information. Third, to give additional functionality to the Gazebo simulator, adding complete models of the Nao humanoid and RGB-D sensors. Fourth, to create tools like ROS's Bags. This kind of applications provides the developers the ability to record logs of scenarios to recreate them later. This feature gives the ability to prove different algorithms on exactly the same data source. Finally, we want Jderobot applications to be easily interactive with regular web browsers and other frameworks. With this concept, human users could easily modulate the applications on real time or see their processing outputs, even in embedded and screenless systems. In addition, Jderobot applications could easily be integrated with other existing frameworks. We are exploring new generation webservices like API-REST for this. Nevertheless these extensions, the framework is stable and with enough functionality. The main efforts will be devoted to extensively use it for research more than to expand it.

References

Brooks A., Kaupp T., Makarenko A., Williams S. and Orebäck A. 2005. Towards component-based robotics. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (págs. 163-168).

Cintas R., Manso L., Pinero L., Bachiller P. and Bustos P. 2011. Robust behavior and perception using hierarchical state machines: A pallet manipulation experiment. *Journal of Physical Agents* , 35-44.

Galvan S., Botturi D., Castellani A. and Fiorini P. 2006. Innovative Robotics Teaching Using Lego Sets. *EEE International Conference on Robotics and Automation Orlando.* Florida.

Gerkey B.P., Vaughan R.T. and Howard A. 2003. The player/stage project: tools for multirobot and distributed sensor systems. *Proceedings of the 11th International Conference on Advanced Robotics ICAR-2003*, (págs. 317-323). Coimbra (Portugal).

Henning, M. 2004. A new approach to object-oriented middleware. *Internet Computing, IEEE* , 66-75.

Hunt A. and Thomas D. 1999. *The pragmatic programmer: from journeyman tomaster.* Boston: Addison-Wesley Longman Publishing Co., Inc.

Kernighan B.W. and Pike R. 1999. *The Practice of Programming.* Addison-Wesley Professional Computing Series.

Koening N. and Howard A. 2004. Design and use paradigms for gazebo, and open-source multi-robot simulator. *In proceedings of 2004 IEE/RSJ International conference on Intelligent Robots and System.* Senday (Japan).

Konolige K. and Myers K.L. 1998. Artificial Intelligence and Mobile Robots: case studies of succesful robot systems. En R. P. In David Kortenkamp, *The Saphira architecture for autonomous mobile robots* (págs. 211-242). MIT Press, AAAI Press.

Montemerlo M., Roy N. and Thrun S. 2003. Perspectives on standardization in mobile robot programming: the Carnegie Mellon navigation (CARMEN) toolkit. *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-03), volume 3*, págs. 2436-2441.

Quigley M., Conley K., Gerkey B., Faust J., Foote T., Leibs J., Wheeler R. and Ng A. 2009. Ros: an open-source robot operating system. *ICRA Workshop on Open Source Software* .