“Give me the Red Can”: Assistive Robot Task Creation through Multi-Modal Interaction

Juan G. Victores, Santiago Morante, Alberto Jardón and Carlos Balaguer
All of the authors are members of the Robotics Lab research group within the Department of Systems Engineering and Automation, Universidad Carlos III de Madrid (UC3M).
jcgvicto@ing.uc3m.es

Abstract

In this paper we illustrate our work focusing on bringing advanced robotics closer to everyday domestic users. It will be demonstrated that inexperienced users can be capable of programming the ASIBOT assistive robot platform to perform a specific desired task in a household environment. The process is guided through the robot's Web browsable interface Task Creator Wizard. The robot's open architecture has been developed to enable flexible multi-modal interaction, such as the used touch buttons, voice commands, and Wii Remote™ controller for intuitive robotic movement. The Wizard has been designed to provide enhanced accessibility while taking aesthetic beauty into account, to avoid distraction caused by boredom of the user. The whole concept and implementation has been released as part of the ASIBOT Open Source Code Repository, available online for download and documentation at:
http://roboticslab.sourceforge.net/asibot

1. Introduction

Modern technologies are progressively being incorporated into our everyday lives. We find ourselves surrounded by elements that are composed by advanced embedded electronics: mobile phones, ebooks, netbooks. Inexperienced people and even young children are able to interact with touch screens or buttons, navigating through tabs, menus, and icons [1]. This fact provides the raison d'être of end-user developments (EUD) [2]. With all the possibilities for efficiency and improvement of EUD, some researchers have already begun to see the potential of web applications [3]. In addition to the provided arguments, web interfaces add powerful benefits, such as ubiquitous availability, and public access (if desired).

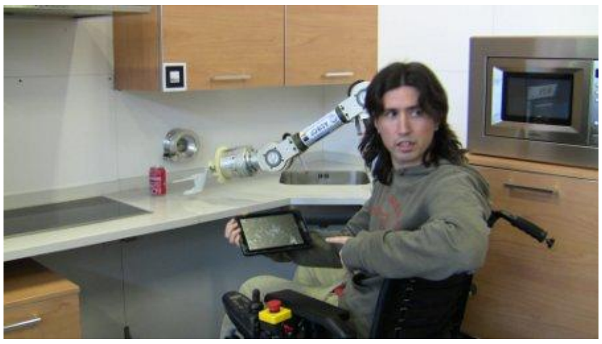


Figure 1. A developer explaining how to use the ASIBOT multi-modal interfaces

All of these advances are also progressively being incorporated in the field of robotics (perhaps at a slower pace). Robotics and automation are fields first developed for industrial environments, with non-friendly engineer-only interfaces. However, recent works such as Baxter [4] are now taking into account the user-oriented point of view to facilitate industrial manipulator programming.

From this steady state of production plants, robotics and automation technology can now be found in retail stores, and ultimately, in our home environment. In its broadest scope, this includes everything from motorized shutters and vacuums to less common advanced robotic manipulators [5]. Current worldwide research focuses on how to introduce dynamic and mobile elements to carry out “housekeeping” and daily chores that require complex manipulation and advanced reasoning skills. These technologies will begin to make our life easier only with the development of human-robot interfaces that provide comfort and satisfaction to the user [6]. In this paper, the authors propose the merger of robotics with technology that everyday users can be familiar with, such as web browsing, voice command control, and video-game controllers, and present proof of concept Open Source implementations and documentation with experimental results.

These developments have taken place using the ASIBOT assistive robot platform (Fig. 1), which is currently located in our ASIBOT assistive living kitchen test environment. The robot and the assistive living environment have been developed by the Robotics Lab research group at UC3M and have been presented by the authors in recent publications [7]. The following is a review of some of the robot's most important features and characteristics.

- Full on-board robot control and communications with no need for an external control cabinet.
- Unlimited workspace through 24V climbing connectors.
- Light-weight symmetrical structure for climbing.
- Tool exchange system for grippers, utensils, sponge, etc.
- Portable and friendly interfaces adapted to different levels of user capabilities and preferences.
- Open architecture for flexible component integration.

The first four cited features are hardware characteristics that allow the robot to overcome many of the robotics issues inherent to the fact that it is a 5 degree-of-freedom manipulator arm (non-redundant for most tasks). The last two features have been exploited by the authors in order to extend the reach of their developments to the hands of everyday home domestic users.

2. Software Infrastructure: Open System Architecture

Through the use of the YARP robotics platform [8] for publisher/subscriber communications, we provide an open architecture that enables flexible multi-modal interaction. In our previous developments we had identified this platform as lightweight enough for our embedded system, and appreciated its multi-lingual and multi-platform support combined with easiness of use for the large range of our developer profiles [9]. Our current implementation also benefits from its `RFModule` (resource location and watchdog thread) and `RateThread` (best-resolution-per-platform periodical threads) classes, multiple carriers (including MJPEG and new custom HTTP carrier), streaming and strict-write-with-acknowledgment ports (each with callback function mapping mechanisms), and the YARP plugin mechanism which allows classes to be used as local libraries or as executables that are remotely accessible through a same class API.

Two collections of libraries that are implemented as YARP devices and can therefore be used through the plugin mechanism have been developed: rlPlugins and rlPlugins2. The rlPlugins library is a small library intended for PC that contains RaveBot (Fig. 2), a simulator class that creates an instance of the OpenRAVE core libraries [10] (qtcoin viewer and ODE physics included). `RaveBot` implements position, velocity, and encoder interfaces, and additionally publishes the stream of images and measurements from every camera and sensor it finds in the environment and robot description XMLs.

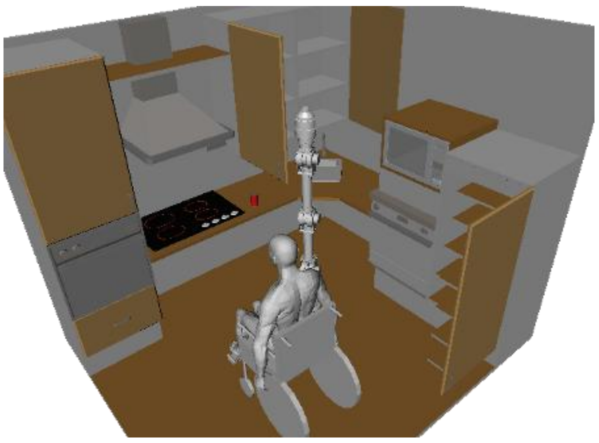


Figure 2. `RaveBot`'s default configuration loads the ASIBOT assistive kitchen model

The rlPlugins2 library is intended to be cross-compiled using a *gnueabi* tool-chain [11] for its use within the robot. It contains `CanBot`, a low-level robot controller class that manages the ASIBOT's internal CAN communications bus for position and velocity control with the appropriate shared resource locks and releases. `CanBot` implements and exposes the same interfaces as `RaveBot`, so `CanBot` and `RaveBot` are identical to external viewers except for port naming. Both rlPlugins and rlPlugins2 additionally provide the CartesianBot plugin, which provides the geometrical closed-form solution of the ASIBOT forward and inverse kinematics. Switching to our generic solver based on recursive methods (`KdlBot` class) can be seamlessly achieved by changing the configuration file or passing it as an argument at instantiation. For either of the two, the

`OrderThreeTraj` class is used to generate trajectories with null initial and final velocities when moving point-to-point in differential kinematic mode. In velocity-controlled streaming mode, the `OrderOneTraj` trajectory generator class is used, in order to assure a constant Cartesian space velocity.

This flexibility is provided through polymorphism: the `OrderThreeTraj` and `OrderOneTraj` objects are allocated on the dynamic memory heap, passing their reference to the `Traj` trajectory class base pointer that is used by the control thread to provide updated references to the low-level controllers. These objects are deleted from the dynamic memory once they are not in use.

The default configuration involves two cartesianServer module instances. Each one instances a `CartesianBot` as a library, setting thread rates and port name prefixes as set in the configuration files. One cartesianServer module is set to use the `RaveBot` as a library and show the simulation on a 2D or 3D screen, and the other instance is set to use the `CanBot` as a library and run on the real robot. These ASIBOT modules can be accessed through any of the following three different methods.

- All of the ASIBOT module ports may be interfaced using the module specific commands of the ASIBOT online documentation through the use of the YARP port class and inherited classes.

- All of the ASIBOT module ports may be interfaced through a series of other mechanisms such as telnet, web browsers, raw sockets, or ROS [12] as documented in the YARP online documentation ("YARP without YARP" and "YARP with ROS" documentation sections).

- We also provide a minimalistic library for communicating remotely with cartesianServer instances, called the CartesianClient library, which has been implemented in native C++, Python, and Java (tested on MATLAB and Simulink too) languages.

3. Components for Task Creation and Multi-Modal Interaction

The ASIBOT Open Source Code Repository is a compendium of C++ and Python programs (named *modules*), libraries, and examples that can connect to each other to create applications that may be useful for users. One of the main objectives of the ASIBOT research and software development of the past years has been to provide integrated modes of Human-Robot Interaction (HRI) through devices with which users can already be previously acquainted with, therefore allowing them to immediately start discovering how to control the ASIBOT robot platform through the interface device instead of using time learning how to control the interface device. Fig. 3 depicts this multi-modal interaction concept.

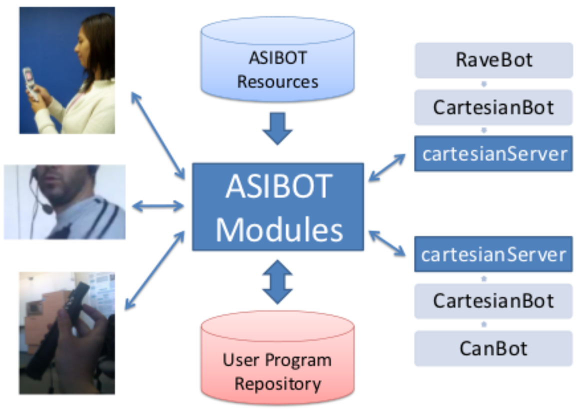


Figure 3. The ASIBOT modules provide multi-modal interfaces for HRI

The ASIBOT modules are intended to a run within the Wireless Local Area Network (WLAN) of the robot, but the user is free to expose the configurable interface sockets for external assistants to collaborate, remotely interacting with the modules from a distant location. The ASIBOT system's open architecture allows the control interfaces and devices to be used simultaneously, and all are managed coherently.

3.1. Touch Buttons

The ASIBOT webInterface module provides a Web browsable interface which is intended for display on devices that support tactile interaction.

The interface is composed by nine functional tabs (Fig. 4, top). A persistent Connection Manager for establishing and terminating communications with the real robot and with the simulator is set to be rendered at the bottom left corner of the browser window. The client side scripts of the webInterface served pages have been optimized to minimize the amount of client-server interactions that take place.

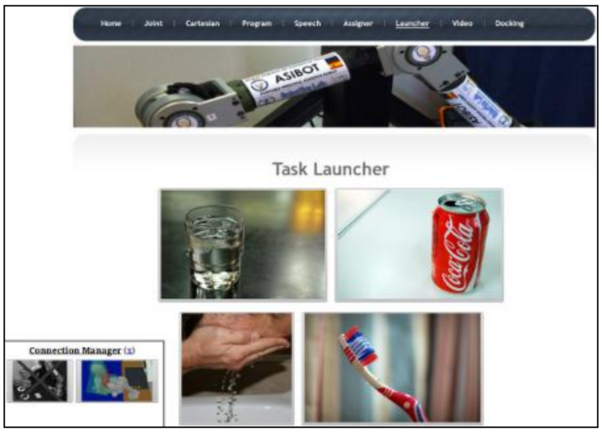


Figure 4. The ASIBOT modules provide touch buttons for HRI

Additionally, through the use of the Asynchronous JavaScript and XML (AJAX) interrelated web development techniques [13], the full page are only loaded when the user changes from one tab to another. Each tab's web content is dynamically changed from within the browser whenever it is required by the user's actions. This bandwidth consumption optimization provides a drastic improvement in performance and allows the user to benefit from page loading time reduction.

3.2. Wii Remote controller integration

A Wii Remote Plus controller interface module has been developed as part of the ASIBOT open architecture components for multi-modal interaction[1]. Upon initializing the wiimoteServer module and connections, the user can move the robot in what we call the Wii space: the robot tip aligns with the Wii-Remote controller pitch (Fig. 5.1), and the robot's base roll is controlled with the controller roll (Fig. 5.2).

A and B buttons control forward and backward translation functionalities respectively, while maintaining both buttons pressed allows Wii space reorientation. This behavior implies the use of a hybrid position/velocity control scheme which is achieved through the use of a floating virtual target point. First, the orientation information is added to the virtual target point. Then, the forward, fixed, or backward translational command component determines the distance between the robot end-effector and the virtual point (positive, null, or negative). The virtual point is sent as a robot target to the cartesianServer modules through the velocity-controlled streaming mode mechanism which has been explained in Section 2.

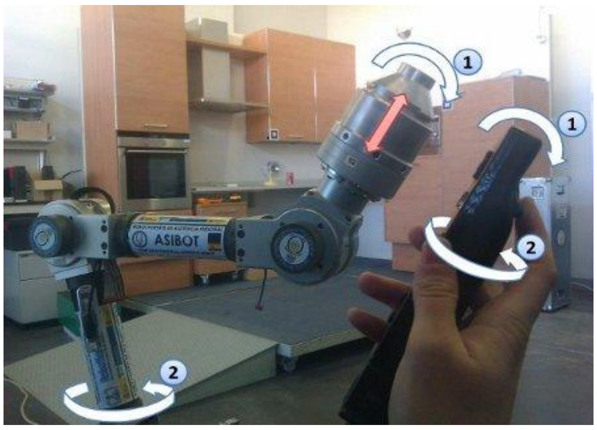


Figure 5. The Wii Remote orientation is tracked with a fixed linear velocity

3.3. Automatic Speech Recognition

ASIBOT's speech recognition has been integrated into the Web browsable interface as a selectable tab. The page served (Fig. 6) contains an automatic speech recognition input field for recording and saving commands which can later be assigned to different tasks.

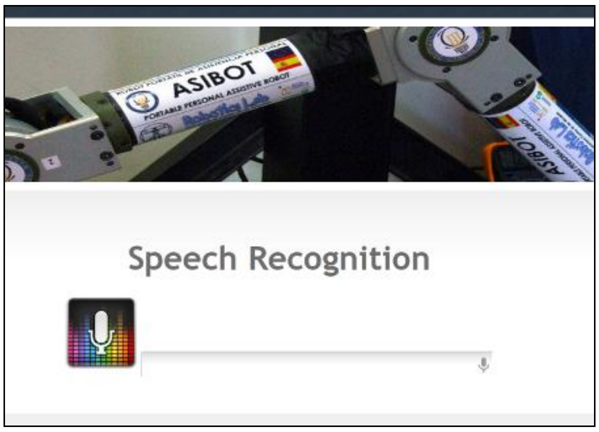


Figure 6. The ASIBOT modules provide speech recognition for HRI

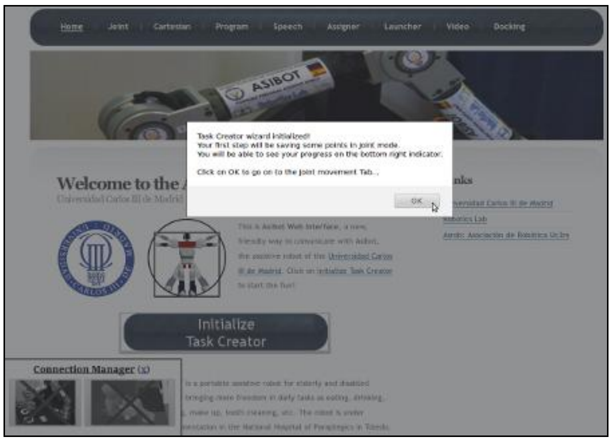
The input field makes use of the x-webkit-speech attribute, which links the field to the Google Inc. implementation of the HTML5 Speech Input API (currently a W3C Editor's Draft [14]). The x-webkit-speech input field attribute is currently recognized by the Google Chrome and Google Chrome for Android

[1] Low-resolution video link: http://youtu.be/S6SKFVUwL9A

web browsers. The Google Inc. implementation of the x-webkit-speech attribute uses Google's service cloud to perform the actual speech recognition, which returns a plain text string that the ASIBOT webInterface module stores in the User Program Repository.

4. A Walk through the Task Creator

The ASIBOT Web browsable Task Creator Wizard has been developed to guide the user through the task creation process from within the ASIBOT Web browsable interface. An ASIBOT task is composed by one or several custom or predefined programs that the user may invoke through the use of one or more of the open architecture's multi-modal interfaces. The Task Creator Wizard is initialized from within the Web browsable interface Home page (Fig. 7, background: Initialize Task Creator). It is set to display useful user guide information in the form of prompts and alerts. The use of the Wizard is, however, not mandatory. The user may instead choose to browse through the tabs manually to develop ASIBOT tasks.


Task Creator wizard initialized!
Your first step will be saving some points in joint mode.
You will be able to see your progress on the bottom right indicator.

Click on OK to go on to the joint movement Tab...

Figure 7. The Task Creator Wizard displays useful user guide information

First, once activated, the Wizard automatically redirects the user to the Joint space movement tab (Fig. 8). The tab is invoked so that a progress bar is displayed on the bottom right corner of the page. It indicates how advanced the user is in the task creation process, and allows the user to jump to each next step throughout the entire creative process.

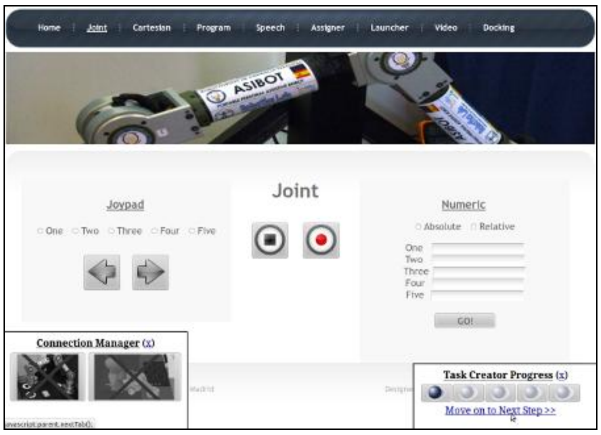


Figure 8. A progress bar guides the user throughout the whole creative process

As has been previously mentioned in Section 3.1, the user can establish connections with the real and simulated robot using the persistent Connection Manager situated at the bottom left of the interface. Once the connections are established, the user can move the selected robots in the Joint space using the correspondent tab buttons.

Additionally, the user can press the capture button (the record icon situated at the center-right of the same Fig. 8) to open a prompt for saving the end-effector point with a custom name. The end-effector point position and orientation information that is stored is computed on the user capture button click event, which may occur even if the robot is in movement. This behavior has also been implemented in the Cartesian space movement tab (Fig. 9), which is the next step the user is guided through.

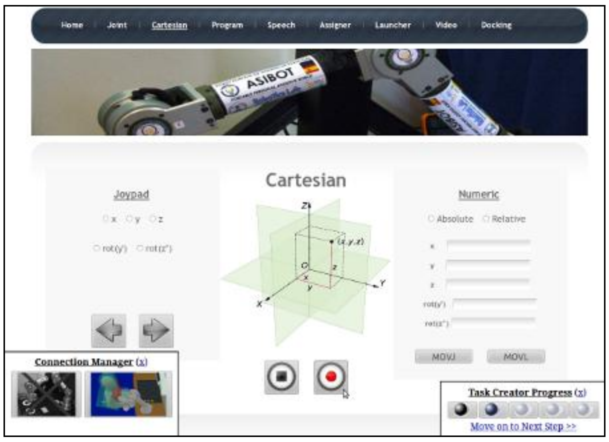
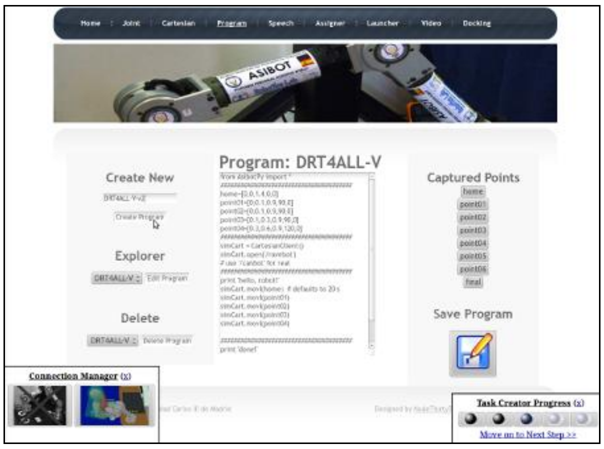


Figure 9. Points may be captured even when the robot is in movement

The capture button of either of these two tabs, namely the Joint space movement tab and the Cartesian Space movement tab, may additionally be used to capture points when the robot is moved by using the Wii Remote Plus controller interface (see Section 3.2). On the completion of this point capturing phase of the Task Creation process, the user is guided by the Wizard to the Program tab (see Fig. 10). Here, the user can create, edit, save and delete ASIBOT modules directly from within the Web browsable interface.



```
from AsibotPy import *
########################################
home=[0,0,1.4,0,0]
point01=[0,0.1,0.9,90,0]
point02=[0,0.1,0.9,90,0]
point03=[0.1,0.3,0.9,90,0]
point04=[0.3,0.6,0.9,120,0]
########################################
simCart = CartesianClient()
simCart.open('/ravebot')
# use '/canbot' for real
########################################
print 'hello, robot!'
simCart.movl(home)  # defaults to 20 s
simCart.movl(point01)
simCart.movl(point02)
simCart.movl(point03)
simCart.movl(point04)

########################################
print 'done!'
```

Figure 10. The ASIBOT Web browsable interface Program Tab

The ASIBOT Web browsable interface Program tab plays the role of an Integrated Development Environment (IDE) for developing ASIBOT user Python modules. The left side panel allows the user to create, explore and delete ASIBOT user Python modules. When the user decides to create a new module, the IDE engine returns a new file with a fragment of default source code. This source code is extracted from a template.py file which is set to load the basic resources for programming the ASIBOT (libraries, initialization routine calls). Additionally, some hint lines of code are added for connecting to a remote instance of the cartesianServer module, performing a robot homing movement, waiting, and closing the module cleanly. The complete template.py file contents can be seen in Listing 1.

The Program tab additionally provides a set of buttons with the captured point names, situated on the top part of its right side panel. Clicking on this type of button inserts two lines of code into the central program text area:

- A Point definition. The definition of the point that was captured and given the name that the button indicates.
- A Cartesian space movement command. The robot is commanded from its current position to an absolute position following a straight line trajectory with null initial and final velocities (a `movl` function call).

```
from AsibotPy import *
########################################
home=[0,0,1.4,0,0]

########################################
simCart = CartesianClient()
simCart.open('/ravebot')
# use '/canbot' for real

########################################
print 'hello, robot!'
simCart.movl(home)  # defaults to 20 s
simCart.wait()      # wait for movement

########################################
print 'done!'
simCart.close()
```

Listing 1. The template.py file contents is copied upon creating modules

An ASIBOT Python point is defined as a native Python list of five doubles that indicate the following: **x**[m], **y**[m], **z**[m], **rot(y')**[degrees], **rot(z'')**[degrees] of the end-effector in absolute base coordinates. The first rotation, **rot(z)**, is given by the `atan2` function of **y** and **x**. This equation completes the Euler ZYZ notation that can be used to define the robot end-effector orientation with respect to its base. Prohibiting the explicit request for the first rotation on the **z** axis helps preventing the request for movements to a great number of points that are unreachable for the ASIBOT, as ASIBOT points may be manually modified or created from scratch, as plain text and the robot is a 5 degree-of-freedom manipulator arm that cannot reach all of the positions and orientations of Cartesian space.

The `movl` member function calls may also be modified and transformed into `movj` function calls. Movements due to `movj` function calls are, generally speaking, faster but less precise (trajectory-wise) than those issued by `movl` commands. This is because `movl` commands involve the computation of a straight linear trajectory, whereas `movj` commands involve trajectory interpolation at single joint level. This nomenclature is commonly found in the context of industrial robots, and the authors have particularly been inspired by the RAPID, an ABB proprietary programming language [15].

Once the user has finished programming, she or he will be prompted to save the program with a custom name by pressing the save button. The Wizard then guides the user to the Speech tab (see Section 3.3). In the Speech tab, the user records and saves words that will be assigned to programs in the Task Creator final step, the Assigner tab (seen in Fig. 11). The Assigner tab is composed by program, recorded word, and icon selectors to generate ASIBOT task files, which are minimalistic scripts that associate these three elements.

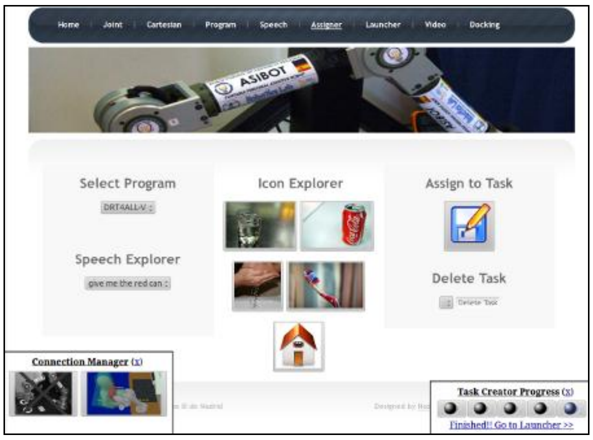


Figure 11. The ASIBOT Web browsable interface Assigner Tab

The Task Creator Wizard leads the user to the ASIBOT Web browsable interface Launcher tab once the assignment has been performed (see Fig. 4). The Launcher parses the task files and presents the selected icons zoomed as touch buttons on screen, awaiting for user tactile interaction or voice commands to execute the tasks that the user has developed through the use of the multi-modal interfaces, with or without the use of the interface's Task Creator Wizard.

5. Experiments

Our experiments were performed, on the technical side, using 300 ms kinematic control cycles, due not to the kinematic calculation times (our kinematic position inversion mean duration is 33 μs), but to not saturate the robot's internal communication CAN bus. The simulator's cartesianServer module was set to directly feed the velocities computed from the trajectory generation described in Section 2 (pure uncompensated feed-forward), as the behavior of the simulated motors is that of a perfect integrator. For the real robot, we added a 0.1 gain on the position error, a classical robotics control approach [16].

On the end-user side, and in order to perform a complete system assessment, we conducted two different tests: one for people that were previously inexperienced with robotics, and one for robotics-related people who were familiar with ASIBOT. The reason for this double-test was to include the not-so-common opinion of developers or technology-skilled users to the common analysis of inexperienced people. The comments and suggestions of technology-skilled users can be useful to assure an easy teaching process, as at a certain point they could actually become the people in charge of training disabled people in handling high-tech adapted devices.

In the first test, the ten healthy inexperienced users were invited to the ASIBOT assistive living kitchen environment and attended a five-day course of two hour sessions. At the end of the course, they were asked to use the developed system for Task Creation through Multi-modal Interaction to perform one simple task: grabbing a red can, a task which we already knew that the robot was capable of achieving, as we had previously performed it by ourselves (Fig. 12).

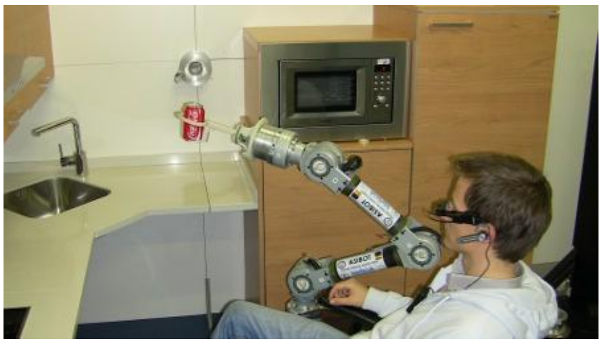


Figure 12. The ASIBOT "Give me the Red Can" task goal achievement

To evaluate the users' experiences, we performed spoken interviews, allowing the users to express their sensations, their pros and cons about the comfortability and easiness-of-use of the applications. The following is a review on the aspects of the implementation and experiments.

- All of the users confirmed they found the use of the proposed multi-modal interfaces (touch buttons, voice commands, Wii Remote controller) very interesting.
- We found devices to fit each of the user's needs so they could all be able to use the Web

browsable interface for comfortably interacting with the robot.

- Each of the users was capable of generating several voice patterns that could be recognized by the automatic speech recognition system.
- At the end of the five-day course, every user had achieved in making the robot grab her or his own red can successfully. Our experience with industrial robot courses indicates that achieving similar tasks with conventional controllers usually takes two weeks if similarly distributed in two hour sessions[2].
- Two users evaluated the use of the Wii Remote controller as an interface device negatively. The main drawback they emphasized on was having to sustain the implemented ``dead man'' buttons while moving the controller to the desired orientation.

In the second test, we asked ten robotics-related people to perform the same task and, in order to measure their satisfaction with the software, we provided them with SUS tests (System Usability Scale). The range of ages of the participants was between 25-35 years old. As a summary of the results:

- The average punctuation was 70.5 ± 9.5 over 100 (where 100 is the best score).
- The best results were obtained in the item: *I think that I would like to use this system frequently*, with an average of 4 ± 0.6 over 5 (where 5 is the best score).
- On the other hand, the worst results were obtained in: *I think that I would need the support of a technical person to be able to use this system*, with an average of 2.6 ± 0.8 over 5 (where 5 is the worst score).

In general, the results were positive and additional feedback was received on how automating the naming mechanism for points, programs, and tasks would be useful, as typing can become tedious and time consuming using certain Web browsing devices.

6. Conclusions

The ASIBOT Open Source Code Repository has gained credibility through the demonstration of how its modules provide flexible Multi-Modal Human-Robot Interaction, and easiness of creating custom user tasks through the use of the developed Web browsable interface's Task Creator Wizard. As researchers, and as implementers, we consider these developments have proved useful in our strive to bring advanced robotics closer to everyday domestic users.

The feedback received from the users has helped us understand how to focus our current and future research efforts, providing continuously improved software revisions and devices for improved user accessibility. In the line of the hardware developments, we have opened a line of research for developing a user accessible device that may be used as a substitute of the Wii Remote controller. This device, of which we already have a working prototype circuit, will be wrist or head wearable. This links this part of our developments more closely to the ASIBOT SULTAN [17] concept and to aiding disabled and elderly people, which has been the ASIBOT's main objective since its origin.

[2] Sample 32 hour course: http://tinyurl.com/roboticTraining2012

7. Acknowledgment

The research leading to these results has received funding from the COMANDER project CCG10-UC3M/DPI-5350 granted by the Comunidad de Madrid and UC3M and the ARCADIA project DPI2010-21047-C02-01 granted by CICYT.

8. References

[1] A. Holzinger, "Finger instead of mouse: Touch screens as a means of enhancing universal access", in Universal Access Theoretical Perspectives, Practice, and Experience, ser. Lecture Notes in Computer Science, N. Carbonell and C. Stephanidis, Eds. Springer Berlin / Heidelberg, 2003, vol. 2615, pp. 387–397.

[2] M. M. Burnett and C. Scaffidi, "End-user development", in Encyclopedia of Human-Computer Interaction, M. Soegaard and R. F. Dam, Eds. Aarhus, Denmark: The Interaction Design Foundation, 2011.

[3] J. Rode, M. Rosson, and M. Qui nones, "End user development of web applications", End User Development, pp. 161–182, 2006.

[4] E. Guizzo and E. Ackerman, "The rise of the robot worker", Spectrum, IEEE, vol. 49, no. 10, pp. 34–41, 2012.

[5] L. Iocchi, J. Ruiz-del Solar, and T. van der Zant, "Domestic service robots in the real world", Journal of Intelligent & Robotic Systems, vol. 66, pp. 183–186, 2012.

[6] M. Kim, K. Oh, J. Choi, J. Jung, and Y. Kim, “User-centered hri: Hri research methodology for designers”, in Mixed Reality and Human-Robot Interaction, ser. Intelligent Systems, Control and Automation: Science and Engineering, X. Wang, Ed. Springer Netherlands, 2011, vol. 1010, pp. 13–33.

[7] A. Jardón, J. Victores, S. Martínez, A. Giménez, and C. Balaguer, “Personal autonomy rehabilitation in home environments by a portable assistive robot”, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 2011.

[8] P. Fitzpatrick, G. Metta, and L. Natale, “Towards long-lived robot genes”, Robotics and Autonomous Systems, vol. 56, no. 1, pp. 29–45, 2008.

[9] J. Victores, “Software engineering techniques applied to assistive robotics: guidelines & tools”, Master’s thesis, Universidad Carlos III de Madrid, Dpto. Ing. Sistemas y Automatica, October 2010.

[10] R. Diankov, “Automated construction of robotic manipulation programs”, Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, August 2010.

[11] G. Sally, “Creating a linux distribution from scratch”, in Pro Linux Embedded Systems. Apress, 2010, pp. 107–141.

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

[13] J. Garrett et al., “Ajax: A new approach to web applications”, Adaptive path, vol. 18, 2005.

[14] S. Sampath and B. Bringert, “Speech input api specification,” W3C Editor’s Draft 18 October 2010. Available: http://www.w3.org/2005/Incubator/htmlspeech/2010/10/google-api-draft.html

[15] ABB, “Rapid reference manual system data types and routines on-line”, 2005, issue: For BaseWare OS 3.1, Article: 3HAC 0966-13.

[16] B. Siciliano, L. Sciavicco, and L. Villani, Robotics: modelling, planning and control. Springer Verlag, 2009.

[17] C. Balaguer, A. Jardón, C. Monje, F. Bonsignorio, M. Stoelen, S. Martínez, and J. Victores, “Sultan: Simultaneous user learning and task execution, and its application in assistive robotics”, in Workshop on New and Emerging Technologies in Assistive Robotics IROS, 2011.