REMOTE CONTROL OF NAO USING GUMSTIX

David Lavy and Alan Belyea

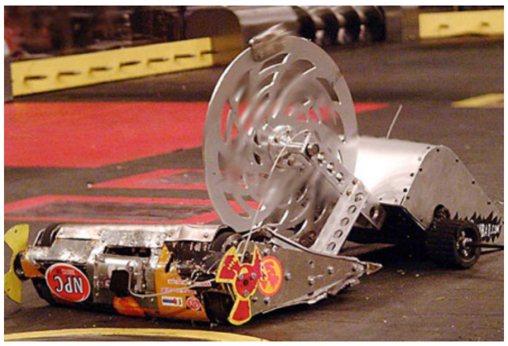
April 2016

Abstract

We introduce the use of a Gumstix board as a remote control for the NAO humanoid robot which will perform 4 different tasks. The proposed system has a GUI on a LCD touchscreen based on Qt which it'll send data via Bluetooth to a server running on a PC. The PC will process the data into readable commands for the robot, which then will send it over WiFi. Finally the robot will receive the commands and execute them one by one.

1 Introduction

Remote control is one of the levels of robotics 'intelligence'. Many robots around the world perform tasks that are difficult for a human to do it, but difficult at the same time for themselves to act autonomously as they don't have a good understanding of the situation and their environment. This is why these robots are operated by a *human brain*. This configuration takes the best of both worlds: strength and expendability of a machine and the brain of a human. One of the most simplest examples is the popular **battlebots** and one of the most complex systems is the **DaVinci robot** (Figure 1). Different applications are used for remote control in robotics. One of the most popular nowadays is navigation in hazardous environments. Another application is in the medical field, where the robot needs to perform extremely precise and fine moves when doing surgery.
For this project we utilize NAO, a complete humanoid robotics platform created by *Aldebaran Robotics*. His extensive capabilities due to a variety of sensors that it has plus its easy-to-use SDK makes him an excellent choice to implement remote control using it's motion dynamics and speaking features.
We implemented our remote control by using the Gumstix board, which by using a graphical interface it will send a series of commands to the NAO robot for execution. We accomplished to successfully operate NAO and send a series of commands from the Gumstix. The PC will queue all the commands and send them sequentially to NAO, which will execute them one by one without crashing or losing any commands.


(a) Battlebots competition

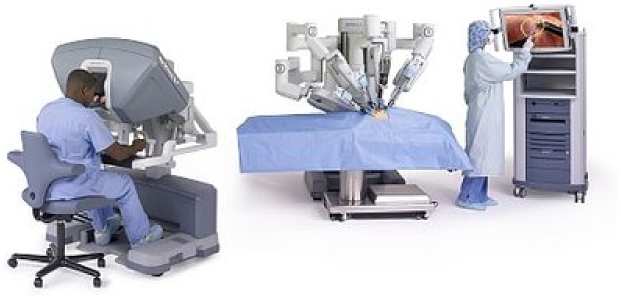

(b) DaVinci robot

Figure 1: Applications of remote control in robotics

2 Design Flow

The project uses a LCD touchscreen that will run a GUI connected to the Gumstix board. The Gumstix transmits user data via Bluetooth to a Python server running on a PC. The Python server receives all the data from the Gumstix and breaks it down into a series of commands and parameters. The commands will be interpreted in a different Python script that will be running the interface between NAO and the PC using a WiFi connection. This Python script uses NAOqi (NAO's SDK) to send the specific action that the robot will *understand* and execute.
The LCD touchscreen display two button groups and 3 edit texts. The first button group displays all the 4 different commands that NAO can execute: Sit, Stand, Speak and Walk. The Walk command uses 3 different parameters that are set in the 3 edit texts: X coordinate, Y coordinate and Theta coordinate (the interpretation of these parameters in the robot are shown in Figure 3) which are set using the second button group to the specified coordinates (in meters and degrees).
The LCD communicates with the Gumstix bluetooth peripheral by calling a bluetooth client located on the Gumstix. The client, which is hardcoded to transmit to the base station, transmits the command data to the base station via Bluetooth. The PC base station has the NAOqi interface running, as well as a Python based bluetooth server which is listening for transmissions from the Gumstix. The bluetooth server application receives data, parses it, and translates the data into NAO commands that are sent over WiFi to the NAO robot. The look and aesthetics of the LCD UI was built by David Lavy, while the functionality of the buttons, error checking, and UI backend were programmed by Alan Belyea. The Bluetooth client software was written by Alan Belyea. The Bluetooth server, and interface between base station (PC) and NAO robot was implemented by David Lavy.

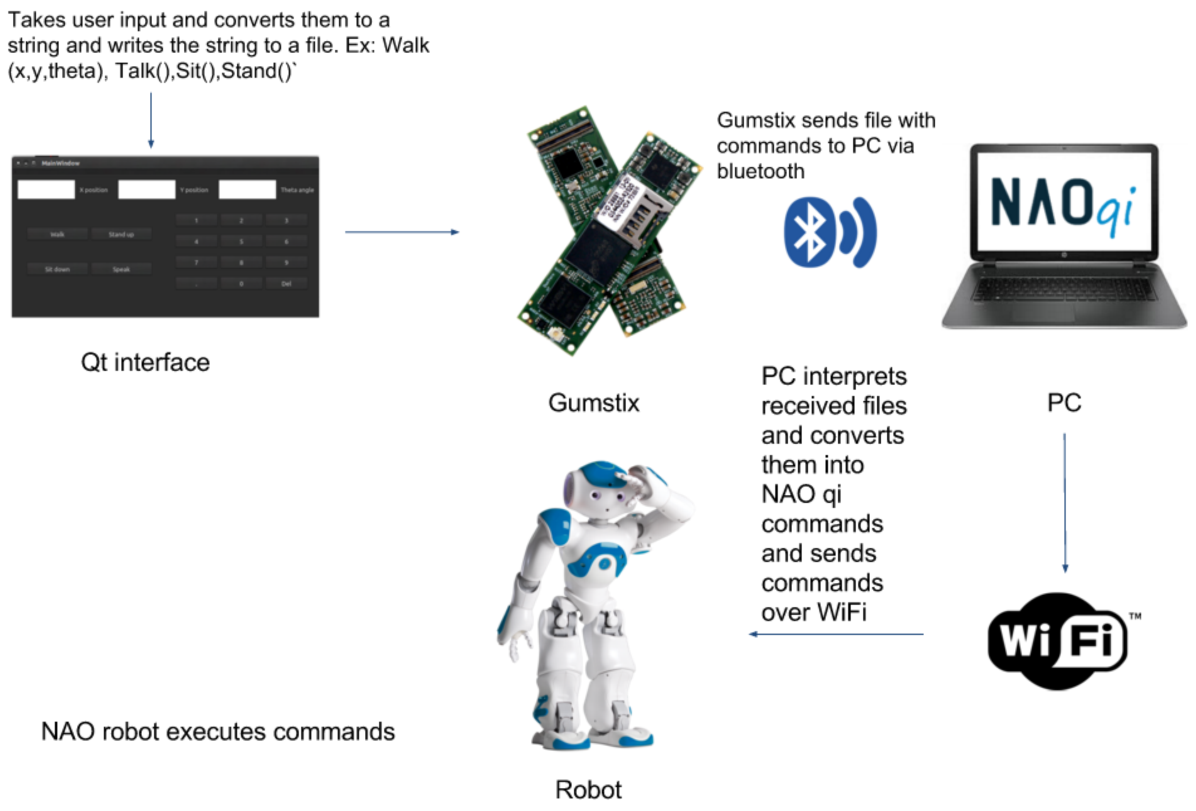

Takes user input and converts them to a string and writes the string to a file. Ex: Walk (x,y,theta), Talk(),Sit(),Stand()`
Qt interface
Gumstix
Gumstix sends file with commands to PC via bluetooth
NAOqi
PC
PC interprets received files and converts them into NAO qi commands and sends commands over WiFi
WiFi
NAO robot executes commands
Robot

Figure 2: Flowchart for the remote control system

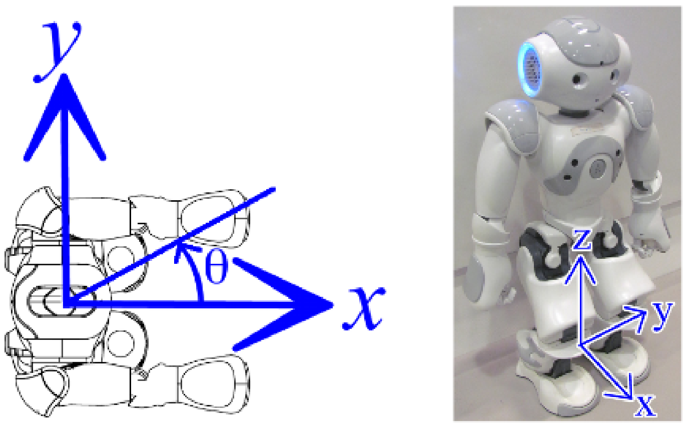


Figure 3: Coordinates' direction for NAO robot

3 Implementation

3.1 Gumstix UI

The Gumstix UI was designed using Qt, and was written in C++. It consists of text edit boxes to input x,y and theta coordinates, as well as a virtual keyboard to enter the numerical data, and a button group for sending final commands to the NAO robot. The user has a list of 4 different commands to execute: Speak, Sit, Stand commands are sent by pressing the appropriate buttons; Walk command needs to first input coordinates using the keypad which will tell the robot walk to a specific destination and finally press the command to send it along with the input parameters.
When a command is executed the UI code (see /UI/mainwindow.cpp) formats all appropriate data that the user input into a comma separated format string, and executes a system call to the bluetooth client with the string as a command line argument. If the walk command is executed with blanks for any of the X, Y or Theta coordinates they are assumed to be 0 and treated as such. If the user enters invalid data, such as a number with multiple decimal points (eg: 1..0 or 1.0.) it is assumed to be invalid input and no system call is executed. If the input has a trailing decimal, and no number after it, it is assumed to be 0 and transmitted (eg 1. Would be assumed to be 1.0). The user is limited to inputting four characters into each coordinate box including decimal points. This is done to conserve space on the LCD by limiting the size of input boxes in an already crowded UI. Commands are sent in sequence one at a time. Multiple commands cannot be input at a time. If more time was available multithreading and queing of commands on the Gumstix side of the project would have been implemented. However, unless a bluetooth connection has timed out, or for some other reason needs to be remade, the transmission time is small making this less of an issue. In addition, the server can queue commands allowing the robot to execute commands in series even if it is not complete an action which can compensate for the lack of queuing on the Gumstix side of the project.

3.2 Bluetooth Client

The Bluetooth client (see /bluetooth/client.c) is written in C. It uses the bluetooth and RFCOMM libraries. Those libraries allow the code to set up a bluetooth socket between the Gumstix and the base station. Currently the MAC address for the base station is hardcoded into the client and would need to be changed if you wanted to use another base station. The code takes an input string determined by the UI code, and sets up a socket. The socket takes the hardcoded MAC address from above, and sets up a socket based on the bluez and RFCOMM library protocols. This socket is treated like a file from the linux perspective. The program then writes to the socket as if it were a file. If the connection cannot be made, or times out the program returns errors and exits without crashing. The client does not support multiple commands being sent at once, and can only send one command each time it is called.

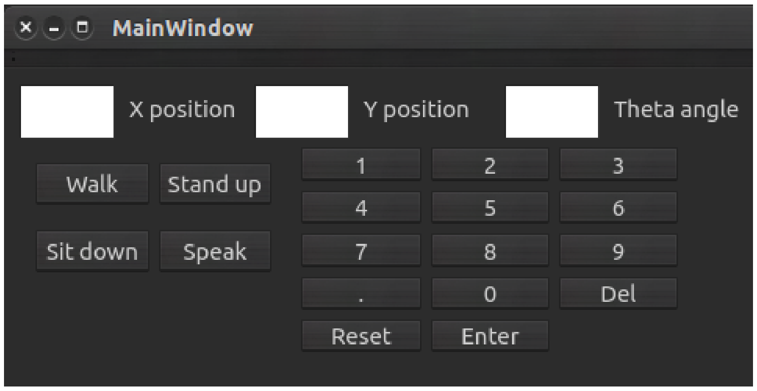


Figure 4: Qt GUI interface

3.3 PC Server

The PC server (see /NAO/BTserver.py) is written in Python. It is similar to the bluetooth client as it uses the same libraries (*bluez*) but in the Python version [1]. It sets up a socket that will be constantly listening to any data that client from the Gumstix will send. As soon as it receives data in form of a string it will open up a file and write the data on it. The server will stack all the received commands in a queue based on that file. A second python script called NAO/naoqiServer.py reads the queue and executes all commands stored in the queue sequentially. If invalid commands are received from the Gumstix, either because of packet loss or invalid commands the server will ignore them without crashing and will continue executing the next command in the queue without crashing.

3.4 NAO Robot

NAO's key components are a 57 cm tall body with 25 degrees of freedom with electric motors and actuators. He owns a whole sensor network of two cameras, four microphones, a sonar rangefinder, two IR emitters and receivers, one inertial board, nine tactile sensors and eight pressure sensors. NAO is also equipped with several communication devices, such as a voice synthesizer, LED lights and two high-fidelity speakers. Moreover, NAO owns two CPUs, one located in the head and the other one located in the torso, which are Intel ATOM 1,6ghz CPUs that run a Linux kernel and support NAOqi, which is Aldebaran's propriety middleware. Figure 5 shows more of its capabilities.
The final implementation was written in Python (see NAO/naoqiServer.py). Although it would be nice to have it in C++ for a much faster compilation and running, it was a challenging task to learn most of the SDK for NAO (NAOqi) just using Python and due to time constraints C++ was not feasible.
The program starts by creating the IP port from which the PC will connect to the robot. The specific IP address is determined by the NAO robot when it boots up. If you press the chest button of the NAO robot when it boots up it will tell you its IP address through built in speakers. The specific IP address can then be coded into the python scripts as needed. Once the connection is set, we create a proxy which will connect the main broker running in the PC to a broker running in the robot (Figure 6). This broker which runs the NAOqi framework will have a direct access to its modules In our case we need three modules: Motion (for walking), Posture (for standing and sitting down) and Speech (for talking). Each module has its own methods from which we will call to make the robot perform certain action. A visual example is shown in Figure 7.
The program interacts with the bluetooth server running already in the PC. Once the server has stacked a command in the file, the program will detect this extract it and parse it, this way it'll know which action needs to send and what parameters it needs to go along with it. The program is set so it will not execute any action if any other action besides the one specified in the GUI are sent. Once it has the command and parameters it uses the corresponding proxy and sends the action through WiFi to NAO. To avoid conflicts in executing multiple commands, that are not compatible, at the same time (for example sitting and standing) the program waits until the action is completed and then attempts to read the file again if any other command

has been sent. This way NAO will respond to commands in a sequentially manner.

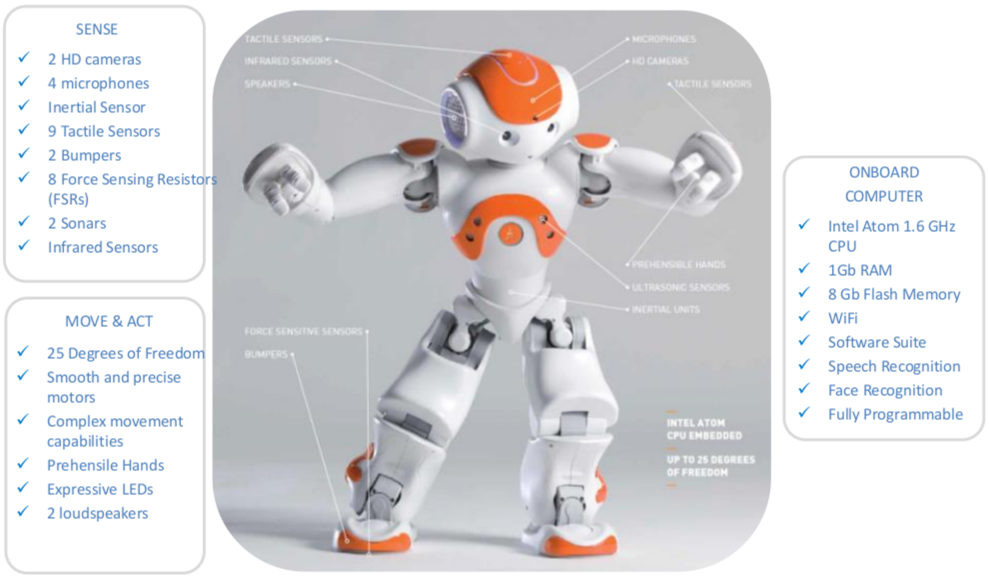


Figure 5: NAO interface [2]

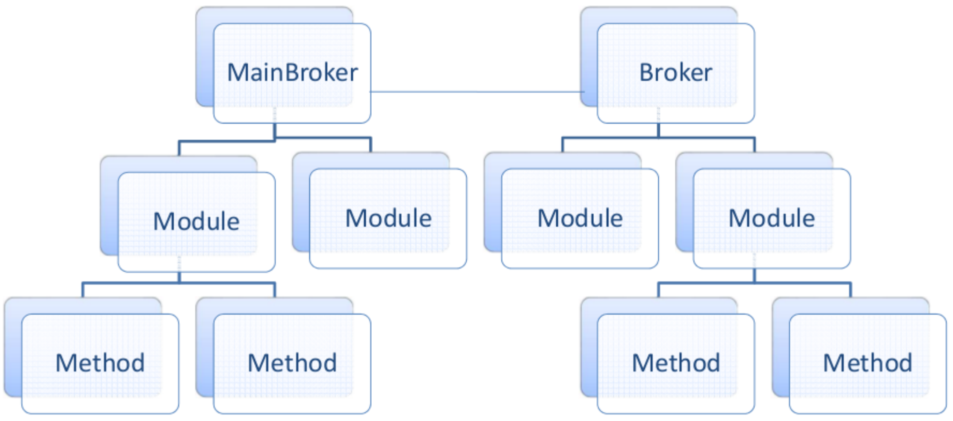


Figure 6: Architecture description between PC and NAO brokers

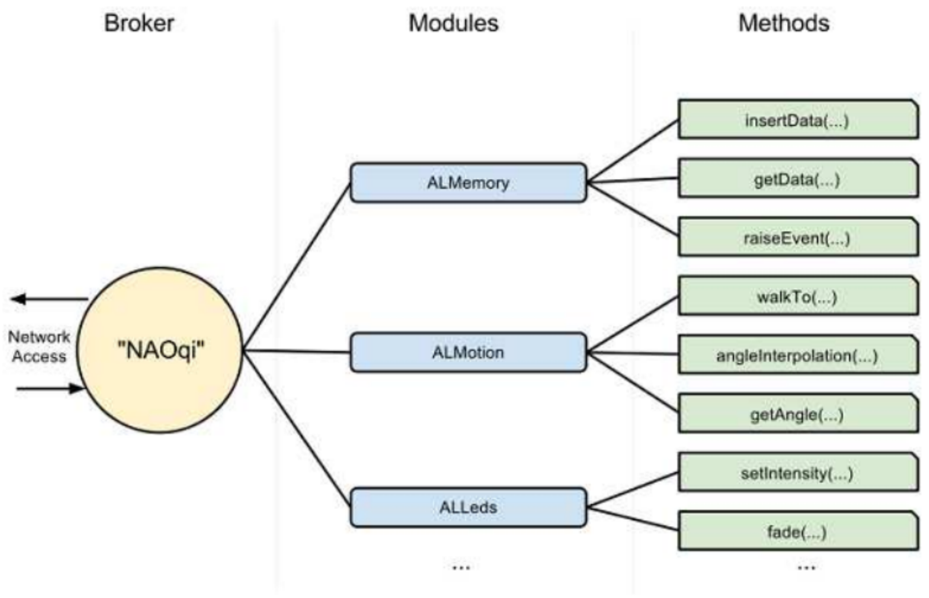


Figure 7: Broker structure

4 Conclusion and Future Work

This project implemented a remote control system which aims to interface with a humanoid robot via a Gumstix board For continued work on this project with Gumstix and NAO, there are several improvements that can be made on the GUI. One of them is to add functionalities on the robot. NAO is a complete robotics platform and it has different functionalities; because of the time constraints we weren't able to dig too much on this capabilities and focused on a few basic ones to have an understanding on how the SDK works. It would be nice to add other functions such a speech recognition or visual recognition of objects being activated from the Gumstix. Another one is to implement a interactive multi-screen menu where we can switch from an initial screen showing all the possible commands to a second screen where we can input the respective parameters per command. Because of the size and poor response of the fingers on the touchscreen it was difficult to easily interact with the screen by setting the parameters because of the size of the buttons. A better distributed menu would improve the experience controlling NAO just like using your phone.

References

[1] Pybluez documentation. `https://github.com/karulis/pybluez`, 2014.

[2] Aldebaran Robotics. Naoqi documentation. `http://doc.aldebaran.com/`, 2005.