Universitat de Girona
Escola Politècnica Superior

Treball Final de Grau/Carrera

Realitzat i defensat a Heriot Watt University de Regne Unit

Estudi: Grau en Eng. en Tecnologies Industrials Pla 2010

Títol: Development of a teaching assistance application for SoftBank Pepper

Document: Treball Final de Grau

Alumne: Ignasi Lleonsí Carrillo

Director/Tutor: Marc Carreras

Departament: Arquitectura i Tecnologia de Computadors

Convocatòria (mes/any): Juny/2017

Development of a teaching assistance application for SoftBank Pepper

Ignasi Lleonsí Carrillo

Heriot Watt University

A Thesis Submitted for the Degree in
Engineering in Industrial Technologies
2017

Abstract

During this project, we are going to develop an application to support teaching assistance (presentations) on the Softbank Pepper robot. One of the many problems when working with robots is flexibility and comfort for the end user. This project wants to create a simple application that uses the regular user's inputs to feed information to the robot, such as images to show, dialog, etc.

This application will be built from scratch and should be easily extendable by people familiar with the robot's official software. That is one of the main objectives of the project.

This project provides all the necessary tools to learn everything needed to develop the application further, or to just use it as is.

Contents

List of figures

1 Introduction

1.1 Motivation

The Pepper robot from Softbank Robotics focuses on interaction and understanding people's emotions, making it an already powerful addition to many different shops and businesses that's welcoming, informing and entertaining their customers, very present especially in Japan. Some people have even adopted a Pepper robot in their homes.

In robotics, interaction is a great field to explore right now, because as robots become cheaper to obtain and easier to program, it's more appealing to acquire one. This is very interesting for businesses or private owners, since they easily keep people entertained and interested in anything the robot has to say or do.

Teaching assistance is a very interesting (and broad) branch of interaction that has not been as developed as other areas. There's various examples of AI usage for teaching assistance (especially on a PC) but not many on actual robots. This is why the author felt inclined to work on this subject.

The hardware used will be the Softbank Pepper robot, and the software is going to be a combination of Choregraphe, the NAOqi Python SDK, and a GUI building tool for Python.

The author of the project has never worked with robotics before, and has only very basic knowledge of programming. His interest in the subject resulted in the idea of this project and had to learn everything from scratch. This is why this project is going to go through all the tools needed to learn and understand everything, and develop it further.

1.2 Objective

As stated before, this project's objective is to build an application that easily allows the end user (teacher or anyone needing to do a presentation) to use a simple application on their Windows OS that communicates with the robot and is instantly ready to support them for teaching or doing any presentation.

The basic features of the application (on the robot) will be:

- Showing a PowerPoint.
- Saying what the user wants on each slide.
- Navigating the PowerPoint with keywords like: "next", "previous", "start/stop presentation", etc.
- User Manual available for use.

There are other, optional features that the author would like to add given there is enough time in the duration of the project, those being:

- Asking questions to the audience and assess if the answers are right.
- Personalizing gestures.
- Extending the possibilities of voice recognition and keywords.

The application will always be extendable, meaning anyone with enough knowledge and the needed software can add or edit any features of the interactive presentation.

1.3 Scope and specifications

The main objective is to allow the robot to make an interactive presentation without any need for human support. The end user should be able to upload the presentation, write the text Pepper needs to say and upload everything to the robot in less than 15 minutes.

The development of the project has two parts: behaviour creation and application creation.

Behaviour creation includes the programming with NaoQI, using the robot's OS methods and programming language to build a program that allows us to do the following things:

- Show the images (presentation) stored inside the robot on its tablet.
- Whenever it shows a slide, recover the text from the respective text file and read it out loud.
- After showing a slide and speaking, begin speech recognition and allow the user to navigate the presentation with voice commands.

Application creation includes python programming and creating a Windows executable that does the following:

- Prompt the end user for the directory their presentation is in, saved as .PNG images.
- Prompt the user for the text that they want the robot to say to the audience in each slide.
- Upload the presentation and the text to the robot so the behaviour can access the new presentation, making it flexible.

1.4 Document structure

This project aims to give all the tools needed to understand and work with the project as the author has, so anyone can edit or extend the application according to their needs.

Still, the author recommends reading the official documentation for all the software used during the project development, since it contains more detailed information.

Therefore, it will be structured in two main chapters: *Project Documentation* and *Project Development*.

Project Documentation explains everything that was needed (software, settings) to develop this project, and Appendix A provides the necessary tools to install and/or learn.

Project Development focuses on the actual programming and design choices for the project, explaining everything as detailed as possible. It will be structured in the two parts mentioned before: Behaviour creation and application creation.

Appendix A is a User Manual to the application, that will have all the information needed for the end user to prepare their presentation.

1.5 Vocabulary clarification

- Application: Executable file that the end user will use.
- End user: Any person that uses the application to make a presentation with Pepper.
- User Manual: Annexed document that the author recommends the end user checks while using the application.

2 Project documentation

2.1 SoftBank Pepper

2.1.1 Developer

SoftBank Robotics (formerly Aldebaran Robotics) is a robotics company, among the world leaders in the field of humanoid robotics, especially in the professional sphere.

The company has developed several models of robots, including the humanoids NAO, Romeo, and Pepper in collaboration with the Japanese SoftBank. This "family of robots" has several objectives: NAO is geared towards programming, teaching and research, Roméo is intended to assist elderly people and Pepper to customer relations or users.

According to the official communication, the company's vision is centred on the will to make robots for the good of men, with the ultimate objective of marketing its robots to the public as a "new species that is kind to humans".

2.1.2 Robot

Pepper is a robot created by Aldebaran Robotics and Softbank Robotics, and in their words, it is “the first robot that can read emotions”.

It is designed to work in customer service, welcoming, informing and amusing customers in a kind and familiar manner. Pepper is particularly popular in Japan, where it’s present in hundreds of stores, and it is starting to be used in stores all around the world. SoftBank claims that Pepper has helped some stores in the US grow in sales and foot traffic.[1]

[1] As seen in this article https://www.recode.net/2017/1/4/14171436/softbank-robot-pepper-sales-brick-and-mortar-retail-ces

SoftBank claims there are over 10,000 Pepper robots in use all around the world.

Anyone can participate in Pepper development if they own a robot with a legal license, for that reason SoftBank have set up a website[2] to share with other developers.

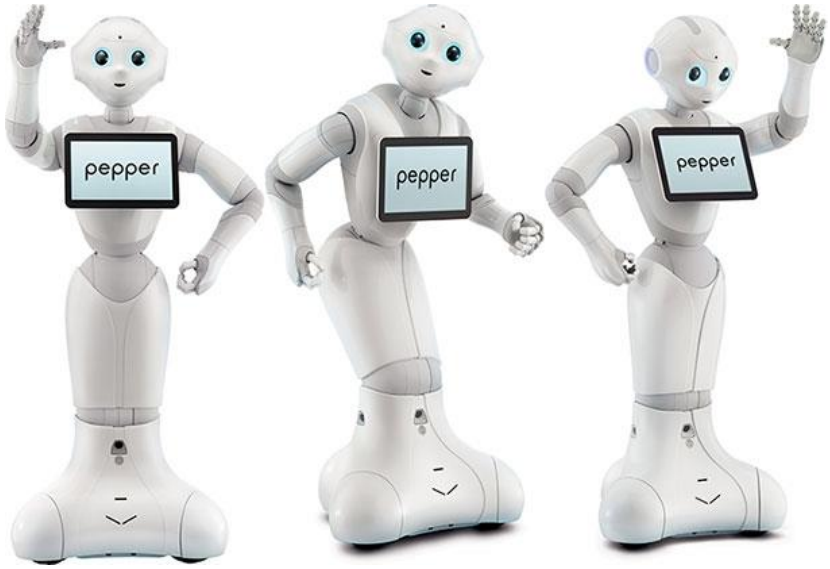


Figure 1. SoftBank Pepper seen in different postures

Technical specifications

Pepper	
Dimensions (mm)	Height: 1210; Width: 480; Depth: 425
Weight	28 kg / 62 lb
Battery	Lithium-ion, 30 Ah / 795 Wh / 12 hours
Tablet	LG CNS Tablet; 10.1 inch
Head	4x Microphones; 2x RGB Cameras; 1x 3D Sensor; 3x Tactile Sensors
Chest	1x Gyrometer; 1x Accelerometer
Hands	2x Tactile Sensors
Feet	6x Laser Sensors; 2x Infra-red Sensors; 1x Sonar; 3x Bumpers
Moving parts	20 Motors
Moving speed	Up to 3 km/h
Networking	1x Ethernet (1xRJ45 - 10/100/1000 base T); 1x Wifi (IEEE 802.11 a/b/g/n)
Operating System	NaoQI OS

Table 1. Pepper technical specifications

For more detailed specifications, check Aldebaran's Documentation.[3]

[2] The mentioned website is the following https://developer.softbankrobotics.com/
[3] http://doc.aldebaran.com/2-4/family/pepper_technical/index_pep.html

2.1.3 NAOqi OS

NAOqi OS is the Operating System of the robot. It's a GNU/Linux distribution based on Gentoo. It's an embedded distribution specifically developed to fit the Aldebaran robot needs.

It provides and runs numbers of programs and libraries, among these, all the required one by NAOqi, the piece of software giving life to the robot.

The robot is controlled using its APIs. Those are either used in Choregraphe Suite or in any of the SDKs (Python, C++, Java, Javascript, etc.) provided by SoftBank Robotics.[4] The main APIs are the following:

- NAOqi Core
- NAOqi Interaction engines
- NAOqi Motion
- NAOqi Audio
- NAOqi Vision
- NAOqi People Perception
- NAOqi Sensors & LEDs
- DCM

Figure 2. NAOqi OS

[4] To install and learn everything needed to work with Pepper (Choregraphe, SDKs), the author recommends the official documentation, found at http://doc.aldebaran.com/2-4/index_dev_guide.html

2.1.4 Choregraphe Suite

Choregraphe Suite is a multi-platform desktop application that allows the user or developer to work with Pepper in an effortless way, reducing the programming skills needed with a friendly interface and easy to understand behaviour creation.

It allows the user to do the following:

- Create animations, behaviours and dialogs,
- Test them on a simulated robot, or directly on a real one,
- Monitor and control you robot,
- Enrich Choregraphe behaviours with Python code.

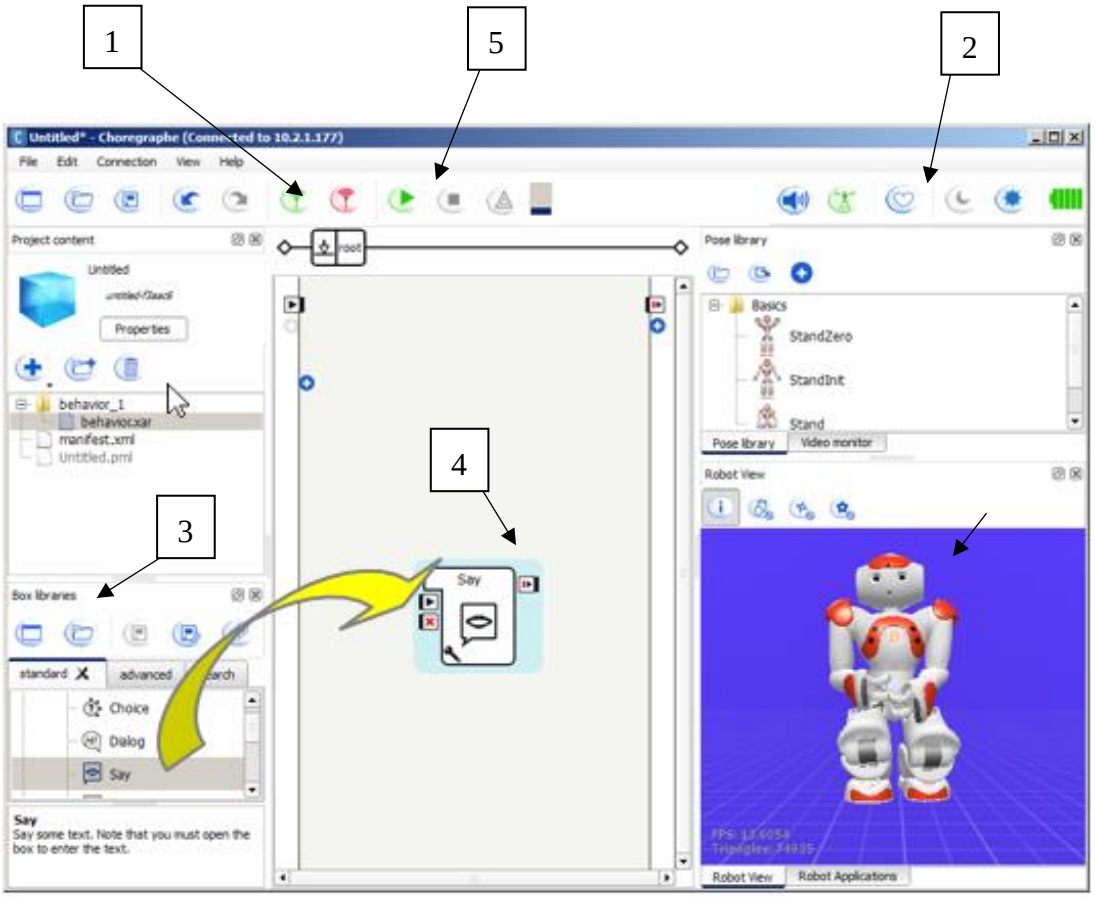


Figure 3. Behaviour creation in Choregraphe Suite

1. **Connection**
   Allows the user to connect to either a virtual robot (as shown in Figure 3) or to a real robot using the network. To connect to the real robot, you must be connected to the same network as the robot.

2. **Autonomous Life On/Off**
   Autonomous Life is the application that "gives life" to the Robot. The Robot becomes visually alive: it "breathes", moves and is aware of its environment.[5]

3. **Box libraries**
   Displays the currently opened Box Libraries and is the main tool for enriching different behaviours. You can use prebuilt boxes for simple programs, but you eventually want to write your own code inside them, which is possible in Python.

4. **Flow diagram panel**
   Placing different boxes here allow for the creation of programs in a simple manner. Linking them makes it visual and informative for the developer.

5. **Run**
   Uploads the current behaviour to the robot in a temporary memory slot and runs it. This is different from uploading the project to Pepper permanently, which allows for interactive behaviours.

Therefore, to make a simple program, like making the robot speak, the user only need to drag the Say box to the flow diagram panel, edit the parameter “Text” to anything you want the robot to say, and run it.

Trigger sentences

We need Pepper to be interactive, and to function on its own when it meets new people. There are two ways of doing that, trigger sentences or trigger conditions.

Pepper’s voice recognition is its most powerful way to interact with people. Trigger sentences are set by the programmer, and allow the robot to know what to do when someone talks to it. They are used to start behaviours or dialogs, and further interaction can exist inside of the behaviours themselves.

To define a trigger sentence, you need to open your project on Choregraphe, open Project Properties, choose the behaviour you want to have a trigger sentence and write it in the respective box.

The behaviour must be uploaded to Pepper (not just running) for it to listen and recognise the trigger sentence.

[5] https://ppd-community.ald.softbankrobotics.com/en/resources/faq/what-autonomous-life

This project uses a trigger sentence to launch the behaviour containing the presentation.

There are also trigger conditions that use environment conditions to launch behaviours, instead of relying only on voice recognition.[6]

As seen in Figure 4, Pepper's eyes become blue when he starts listening to you, and his voice recognition is active. He must have seen you before, and turned his head and body in your direction.

Again, the documentation explains further how to use Choregraphe and it allows you to try different tutorials and learn all the possibilities it provides.

Figure 4. Pepper voice recognition

Box inputs/outputs

All boxes created in Choregraphe's flow diagram panel can receive different inputs, and can also give different outputs.

It's very important to be familiar with them, as well as their nature, to understand how the presentation behaviour on this project works.

All possible inputs/outputs are listed on the official documentation[7], and it is highly recommended that you check it while reading *3.1. Behaviour creation*.

The inputs/outputs used by the project, other than the basic ones, are:

- wordRecognized – OUTPUT – Type: String; Nature: Punctual
- answer – INPUT – Type: String, Nature: OnStart
- counter – OUTPUT – Type: Number, Nature: Punctual
- counter – INPUT – Type: Number, Nature: OnEvent

[6] To learn more about triggers, check out Aldebaran's official tutorials http://doc.aldebaran.com/2-4/software/choregraphe/tutos/create_interactive_activity.html

[7] http://doc.aldebaran.com/2-4/software/choregraphe/objects/box_input_output.html

2.1.5 Python SDK

Another way of programming Pepper is using any of the SDKs provided by SoftBank. The Python SDK is the easiest to learn from scratch, and also very powerful.[8]

It allows calling any method in the current version of NAOqi the robot has installed, and the developer is able to be even more in control of the robot than by just using Choregraphe.

The author of this project learned how to use the SDK, but eventually decided to just rely on Choregraphe for the behaviour programming. This will be further explained in Project Development.

2.1.6 Creating a WiFi hotspot

The robot connects to a Wi-Fi network automatically every time it's booted, called "oslwifi" with the password "osllab2013".

Therefore, to connect to the robot, there must be a Wi-Fi network with that name and password that our computer is connected to, or we can create a hotspot with our computer.

There are two ways of doing this, with the default wireless ad hoc network creation that Windows provides, or using a 3rd party program. You are free to do it as you want.

The author recommends using a 3rd party program called OSToto Hotspot[9] The simple reason being it allows you to stay connected to the internet while creating the hotspot, while doing it with Windows ad hoc network kills your internet connection.

Another option is to use Virtual Router Manager[10]. This program was a bit more difficult to get Pepper to connect properly, so the author recommends trying the other one first.

It's easy to use, since it only requires you to choose the network name and password, as listed above, and click "Start Virtual Router".

It is recommended that you try to connect to the new network with your phone before turning Pepper on. Pepper only tries to connect to the internet once, when she is booting up, and if it can't, it will not retry until restarted.

---

[8] For more information, go to http://doc.aldebaran.com/2-4/dev/python/intro_python.html
[9] http://www.ostoto.com/products/wifi-hotspot.html
[10] https://virtualrouter.codeplex.com/

2.2 Python

As the official documentation describes it: “Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants, on the Mac, and on Windows 2000 and later.”[11]

All these features make Python very easy to learn compared to its counterparts, and a lot more accessible for inexperienced users, since there are huge databases and forums to find helpful information in.[12]

Figure 5. Python logo

Characteristics:

- Simple
  Python is minimalistic and simple. It emphasizes readability, making it readable like English. This is one of its most important characteristics, because it allows you to understand it easily.

- Accessible for beginners
  Python also has an extremely simple syntax, making it very easy to learn, even for non-programmers.

[11] https://docs.python.org/2/faq/general.html#what-is-python
[12] The most prominent one being Stack Overflow http://stackoverflow.com/

- Free and Open Source Software
  Python is a FOSS (Free and Open Source Software)[13]. Also known as FLOSS, where the "L" stands for "Libré", making it clearer that Free stands for "Freedom".
  This means anyone can freely distribute copies of the software, read its source code and make changes to it, use pieces of it in new free programs. FLOSS is based on the concept of a community sharing knowledge. This is another reason that makes Python great, it's constantly improved by developers and community.

- High-level Language
  While using Python, you don't need to understand low-level details of your computer or operating system. Those are automated and made much simpler by the programming language itself. This is a characteristic of all high-level languages.

- Portable
  Due to its open-source nature, Python has been ported (i.e. changed to make it work on) to many platforms. Most Python programs can work on any of these platforms without requiring changes, as long as there are no system-dependent features.

- Interpreted
  A program written in a compiled language like C is converted from the source language into a language that is spoken by your computer (binary code) using a compiler with various flags and options. When you run the program, the linker/loader software copies the program from hard disk to memory and starts running it.
  Python, on the other hand, does not need compilation to binary. You just run the program directly from the source code. Python converts the code to byte-codes, then into the machine language, and finally runs it. All this makes using Python much easier since you don't have to worry about compiling the program, making sure that the proper libraries are linked and loaded, etc. This also makes Python programs much more portable, since copying a Python program to another computer won't cause any problems, it still works.

[13] https://freeopensourcesoftware.org/index.php?title=Main_Page

- Object Oriented Programming
  As defined by Webopedia[14]: "Object-oriented programming (OOP) refers to a type of computer programming (software design) in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure.
  In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects."

- Extensible
  Python can be extended by writing Python modules in C extension modules, allowing you to interface Python with libraries written in other languages (C, or other languages that provide C interfaces).
  It can be used if you want some code to run fast, or need to have an algorithm not to be open. That way you can code that part of your program in C or C++ and then use them from your Python program.

- Embeddable
  This is the opposite of extending Python. You can enrich your C/C++ applications by embedding Python in them. Embedding provides your application with the ability to implement some of the functionality of your application in Python rather than C or C++. This can be used for many purposes; one example would be to allow users to tailor the application to their needs by writing some scripts in Python. You can also use it yourself if some of the functionality can be written in Python more easily.

- Extensive Libraries
  The Python Standard Library is huge. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, ftp, email, XML, HTML, GUI (graphical user interfaces), Tk, etc. This is called the 'Batteries Included' philosophy of Python.

This information has mostly been borrowed from the free-to-use book "A Byte of Python", a book on programming about Python. The original text can be found online.[15]

[14] http://www.webopedia.com/
[15] https://python.swaroopch.com/

Learning

The author of the project had very little programming experience beforehand, and no Computer Science knowledge. It's also worth mentioning that the Python SDK uses the 2.7 version, so that's the one that should be checked.

Therefore, here are some recommendations based on his own learning experience:

- Official Python Tutorials[16]
  There are different tutorials available, for inexperienced programmers or for programmers learning Python.
  The official documentation's tutorial is great as a learning tool.

- Introduction to Computer Science and Programming Using Python[17]
  This is an edX free course offered by MIT. It's meant to be done over two months, but you can do it at your own pace or use the Sandbox to see all the projects. Highly recommended to learn basic Computer Science concepts as well as being given challenging projects that boost your learning experience.

- Codeacademy - Learn Python[18]
  Recommended if you have no programming experience, since it explains every basic thing from the beginning.

If you have no programming experience, you shouldn't need to do more than one tutorial to understand the code this project contains.

If you have programming experience, even if it's in another language, you probably will only need to check the official Python documentation at some point if there is some Python-only feature in the code you don't understand.

[16] https://docs.python.org/2/tutorial/index.html
[17] https://www.edx.org/course/introduction-computer-science-mitx-6-00-1x-10
[18] https://www.codecademy.com/learn/python

2.2.1 System path settings

PATH is an environment variable for POSIX operating systems and Microsoft systems, specifying the paths in which the command interpreter should look for the programs to run.

It is usually referred to as $PATH, in POSIX systems, or %PATH%, in Microsoft systems, to differentiate it from the actual word "path".

This variable must contain all the directories in which the interpreter is wanted to search for programs, the order being considered at the time of the search. Whereas in POSIX systems it is a colon separated list (:) and each directory must be explicit; In Microsoft systems, the separator is semicolon (;) and has no reference to the working directory since it is implicit for the system and is the first directory where the interpreter looks.

To run any Python program on your command interpreter, you will need to set up your PATH settings correctly.[19] This is very important but it shouldn't be much of a problem.

2.2.2 Paramiko

Defined by its official website[20]: "Paramiko is a Python (2.6+, 3.3+) implementation of the SSHv2 protocol, providing both client and server functionality. While it leverages a Python C extension for low level cryptography, Paramiko itself is a pure Python interface around SSH networking concepts."

Since Windows doesn't have a built-in command interpreter SSH function, we need something that allows us to do it. By being a Python implementation, it will allow us to import it into our program and make it as easy as possible for us to build our application.

Another option would have been the PuTTY client, but it's a Windows implementation instead of a Python one, which made it slightly more complicated to use it in the application.

To check Paramiko methods and learn the basic connection and file uploading and replacement that the application will do, check the official documentation.[21]

To check demos and basic implementation, go to Paramiko's github page.[22]

It's also important to mention that the basic connection and uploading functions for the application have been borrowed from an open source github code.[23]

[19] https://docs.python.org/2/using/windows.html#excursus-setting-environment-variables
[20] http://www.paramiko.org/
[21] http://docs.paramiko.org/en/2.1/
[22] https://github.com/paramiko/paramiko/
[23] https://github.com/tadeck/ssh-matic

2.2.3 Pyinstaller

As mentioned before, the end objective is to create an application that can run on any Windows computer and doesn't require any installations for the end user (Python or any of its modules, SoftBank software, etc.).

To achieve that, we want to make our Python program into an executable Windows file, .exe. There are various programs that do this, but PyInstaller is a very simple, and to-the-point program that will do it for you.

As defined by the official PyInstaller website[24]: "PyInstaller is a program that freezes (packages) Python programs into stand-alone executables, under Windows, Linux, Mac OS X, FreeBSD, Solaris and AIX. Its main advantages over similar tools are that PyInstaller works with Python 2.7 and 3.3—3.5, it builds smaller executables thanks to transparent compression, it is fully multi-platform, and use the OS support to load the dynamic libraries, thus ensuring full compatibility.

The main goal of PyInstaller is to be compatible with 3rd-party packages out-of-the-box. This means that, with PyInstaller, all the required tricks to make external packages work are already integrated within PyInstaller itself so that there is no user intervention required. You'll never be required to look for tricks in wikis and apply custom modification to your files or your setup scripts. As an example, libraries like PyQt, Django or matplotlib are fully supported, without having to handle plugins or external data files manually. Check our compatibility list of Supported Packages for details."

To use it after you finish your Python program, you only need to:

1. Add PyInstaller to your %PATH% variable
2. Run it in the command interpreter, in the correct directory and giving it your python file's name, like so:
   - cd C:\Users\me\path\to\directory
   - pyinstaller pythonfile.py

It will build your application and save it in a new directory called "dist".

Figure 6. PyInstaller logo

[24] http://www.pyinstaller.org/

3 Project development

3.1 Behaviour creation

The first part of the project development consists in the creation of the program on Pepper. This is the only thing the people interacting with the robot will see, as they'll never use the application. Only the person preparing a presentation will have to use the application.

The objectives for the interactive presentation on Pepper are:

- Being in Autonomous Life and attracting people closer.
- Starting the presentation when prompted by someone saying, "start presentation".
- Showing the slides of the PowerPoint presentation and saying the text entered by the end user.
- Allowing the audience to navigate the presentation saying "next", "previous", or "stop presentation".

To achieve these objectives, the behaviour will work in a slightly elaborate way.

The main characteristic is the counter. It keeps track of what slide is being shown, and changes in value as the audience navigates through the presentation, letting the robot know what slide to show and what text to read out loud. When there is no image equal to the value of the counter, the presentation ends.

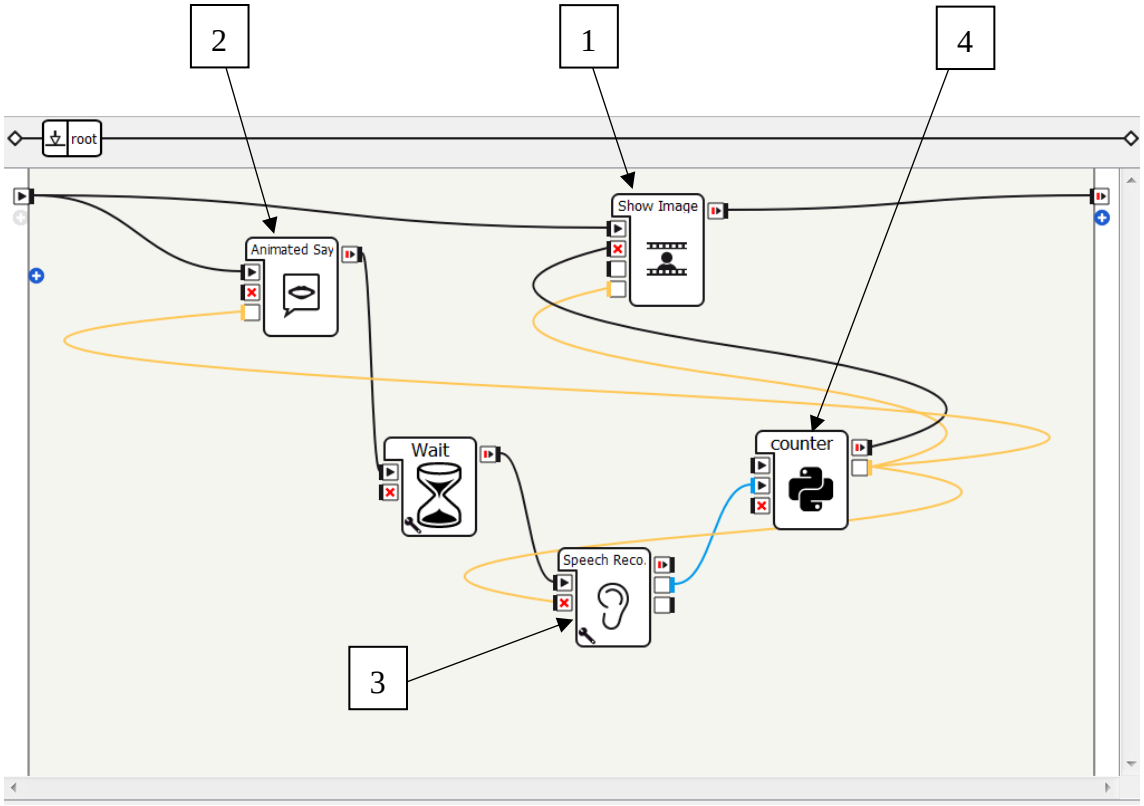


Figure 7. Flow diagram of the presentation

Before getting into the code of each box individually, a general explanation of how the presentation behaviour works is needed.

Upon launched, the counter is set to 1 and “Show Image” and “Animated Say” are triggered, making the robot show slide one of the presentation, and say the respective text at the same time.

After the robot finishes talking, there is a “Wait” box, that delays for a few seconds, and then the “Speech Recognition” box is started. The user can say “next”, “previous”, or “stop presentation”. If the robot understands the user, it will send an “answer” to the counter box, and finish the speech recognition.

The “counter” box checks for the answer input, and depending on what it is, changes the counter appropriately. Then it sends the new counter value to “Show Image” and “Animated Say”, triggering the new image and text that must be shown and said, respectively.

When the counter reaches a value that there is no image of (number of slides + 1), it stops the “Show Image” box, and the behaviour is ended.

3.1.1 Show Image

This box is the first one to be triggered, alongside “Animated Say”. Its function is to display the current slide on Pepper’s tablet, depending on the value of the counter, by accessing Pepper’s memory, where the images are saved.

Let’s have a look at the code before we start breaking it down.

```
class MyClass(GeneratedClass):
    global counter
1   counter = 1
    import time

    def onLoad(self):
        pass

    def onUnload(self):
        pass

    def _getTabletService(self):
        tabletService = None
        try:
            tabletService = self.session().service("ALTabletService")
        except Exception as e:
            self.logger.error(e)
        return tabletService

    def _getAbsoluteUrl(self, partial_url):
        import os
        subPath = os.path.join(self.packageUid(), os.path.normpath(partial_url).lstrip("\\/"))
        # We create TabletService here in order to avoid
        # problems with connections and disconnections of the tablet during the life of the application
        return "http://%s/apps/%s" %(self._getTabletService().robotIp(), subPath.replace(os.path.sep, "/"))

    def onInput_onStart(self, c):
2       # We create TabletService here in order to avoid
        # problems with connections and disconnections of the tablet during the life of the application
        tabletService = self._getTabletService()

        if tabletService:
            try:
                url = "Slide" + str(counter) + ".PNG"
                if url == '':
                    self.logger.error("URL of the image is empty")
                if not url.startswith('http'):
                    url = self._getAbsoluteUrl(url)

                #show the slide
                slide = tabletService.showImage(url)
                #if slide doesn't exist (slide 0 or slide n+1) presentation stops
                if not slide:
                    self.onInput_onStop()

            except Exception as err:
                self.logger.error("Error during ShowImage %s" % err)
                self.onStopped()
        else:
            self.logger.warning("No ALTabletService, can't display the image.")
            self.onStopped()
```

```
def onInput_onHideImage(self):
    # We create TabletService here in order to avoid
    # problems with connections and disconnections of the tablet during the life of the application
    tabletService = self._getTabletService()
    if tabletService:
        try:
            tabletService.hideImage()
        except Exception as err:
            self.logger.error("Error during HideImage : %s " % err)
            self.onStopped()
    else:
        self.logger.warning("No ALTabletService, can't hide the image.")
        self.onStopped()

def onInput_onPreLoadImage(self):
    # We create TabletService here in order to avoid
    # problems with connections and disconnections of the tablet during the life of the application
    tabletService = self._getTabletService()
    if tabletService:
        try:
            partialUrl = self.getParameter("ImageUrl")
            fullUrl = self._getAbsoluteUrl(partialUrl)
            tabletService.preLoadImage(fullUrl)
        except Exception as err:
            self.logger.warning("Error during preLoadImage : %s " % err)
            self.onStopped()
    else:
        self.logger.warning("No ALTabletService, can't preload the image.")
        self.onStopped()

def onInput_counter(self, c):
    counter = c
    self.onInput_onStart(counter)

def onInput_onStop(self):
    tabletService = self._getTabletService()
    tabletService.hideImage()
    self.onUnload()
    self.onStopped()
```

3

4

Figure 8. Show Image box code

This is a combination of the original "Show Image" box's code and my own. I added numbers to the functions I edited to make it easier to spot and explain any changes made. This will also apply to the rest of the edited code.

1. Global variable definition

Defining "counter" as a global variable[25] will allow us to use it in all the functions inside the current class, which is necessary for our presentation.

If counter wasn't global, we would need to find another way of allowing our functions to tell each other which was the current slide. Doing it this way, though, allows it to do it in an uncomplicated way that should be extensible to new functions as well.

[25] A few examples on global variables https://docs.python.org/2.7/faq/programming.html#what-are-the-rules-for-local-and-global-variables-in-python

2. "onStart" input

What's different here to the default "Show Image" box[26] is in the "try" function.

First, it defines the variable "url" as a string, after the current counter value. This is important because it updates the "url" every time the function is called, which is every time the "counter" input is triggered, as we will see in *3. "counter" input*.

Then, it tries showing the image on the tablet, and if it can't, it triggers the "onStop" input. It's worth noting that the *showImage* function returns a Boolean value.

For more information about other functions refer to the official documentation.

3. "counter" input

This function is quite simple. Every time the "counter" box sends a Punctual signal containing a new value for the variable *counter*, it updates its value.

Then, it calls the "onStart" function again, to display the new image on Pepper's tablet, changing the current slide for the new one, completing the cycle.

4. "onStop" input

All that was added here was the method *hideImage*, to make sure Pepper didn't leave the last slide on its tablet when the program ended. Otherwise, Pepper would return to its Autonomous Life status, but the last image would be displayed.

[26] Using Pepper's tablet, from the official Aldebaran documentation http://doc.aldebaran.com/2-4/getting_started/creating_applications/using_peppers_tablet.html

3.1.2 Animated Say

This is the box responsible for accessing the text files and making Pepper say what the end user wants it to in each slide.

1
```
import time
import codecs

class MyClass(GeneratedClass):
```

2
```
    global counter
    counter = 1

    def __init__(self):
        GeneratedClass.__init__(self, False)
        self.tts = ALProxy('ALAnimatedSpeech')
        self.ttsStop = ALProxy('ALAnimatedSpeech', True) #Create another proxy as wait is blocking if audioout is
remote

    def onLoad(self):
        self.bIsRunning = False
        self.ids = []

    def onUnload(self):
        for id in self.ids:
            try:
                self.ttsStop.stop(id)
            except:
                pass
        while( self.bIsRunning ):
            time.sleep( 0.2 )
```

3
```
    def onInput_onStart(self, c):
        self.bIsRunning = True
        time.sleep(2)
        try:
```

4
```
            #function that extracts string from document and uses speech
            def say_from_file(filename, encoding):
                with codecs.open(filename, encoding=encoding) as fp:
                    contents = fp.read()
                    # warning: print contents won't work
                    to_say = contents.encode("utf-8")
                self.tts.post.say(to_say)

            #path if package installed on pepper
            #for OS version 2.5
            #id = say_from_file("/data/home/nao/.local/share/PackageManager/apps/presentation1-8c8a24/html/slide" +
str(counter) + "_tts.txt", "utf-8")
            #for OS version 2.4
            id2 = say_from_file("/var/persistent/home/nao/.local/share/PackageManager/apps/presentation1-8c8a24/html/
slide" + str(counter) + "_tts.txt", "utf-8")

        except:
            pass

        finally:
            try:
                self.ids.remove(id)
            except:
                pass
            if( self.ids == [] ):
                self.onStopped() # activate output of the box
                self.bIsRunning = False
```

5
```
    def onInput_counter(self, c):
        counter = c
        self.onInput_onStart(counter)

    def onInput_onStop(self):
        self.onUnload()
```

Figure 9. Animated Say box code

1. Imported modules

The module *time* allows us to use the "sleep" method, introducing delays.

The module *codecs* allows us to open encoded files, and read the text inside them. This is necessary to read the text files containing what Pepper must say out loud.

2. Global variable definition

Defining "counter" as a global variable will allow us to use it in all the functions inside the current class, which is necessary for our presentation.

If counter wasn't global, we would need to find another way of allowing our functions to tell each other which was the current slide. Doing it this way, though, allows it to do it in an uncomplicated way that should be extensible to new functions as well.

3. "onStart" input

When the "onStart" input is triggered, we define the *say_from_file* function, to avoid potential problems in the behaviour, like the box not loading properly and not being able to access the function. Then, we call *say_from_*file, giving it the path inside the robot's memory to the correct file.

It's like the *Show Image* box, since it uses the counter to access the correct file in each slide of the presentation. It's different to the *Show Image* box because it needs the full path to access the file, while *Show Image* only needed the relative path from the folder called "html".

4. Function *say_from_file*

This function is very important to the behaviour, since it allows Pepper to access a text file, read the text inside it (given the correct encoding), and say it out loud.

Defining this function will allow us to replace text files in Pepper's memory every time we make a new presentation and make Pepper say different things for every presentation it has to perform.

Keep in mind that the path has to refer to the folders inside Pepper's internal memory. This means that any change in folder paths, Choregraphe project name, etc. will make this box not work properly. The best way to find out the correct path is to connect to Pepper remotely (for example, using FileZilla) and looking for the project inside its memory.

This script has the path for versions 2.4 and 2.5 of NAOqi, if the robot you are working with is using version 2.5, simply update the path.

It was taken from the official documentation[27].

5. "counter" input

This function is quite simple. Every time the "counter" box sends a Punctual signal containing a new value for the variable counter, it updates its value.

Then, it calls the "onStart" function again, to display the new image on Pepper's tablet, changing the current slide for the new one, completing the cycle.

[27] http://doc.aldebaran.com/2-4/dev/python/examples/nonascii/index.html

3.1.3 Speech Recognition

It is not needed to have a look inside the code of the Speech Recognition, since it's the default Speech Recognition box, and we don't need to do any changes for it to work as we intend.

We only need to understand Parameters[28], and they are explained in the official documentation.

Basically, all we need to do is write all the answers we want our robot to consider in the "word list" parameter. By default, at the time of completion of this project, Pepper will only consider "next", "previous", and "stop presentation" as valid answers.

You can always enrich and extend the presentation features by adding more speech recognition options, and then editing the code in the other boxes, or adding new ones. Anyone willing to improve the presentation and the application is encouraged to do so without any restrictions by the author.

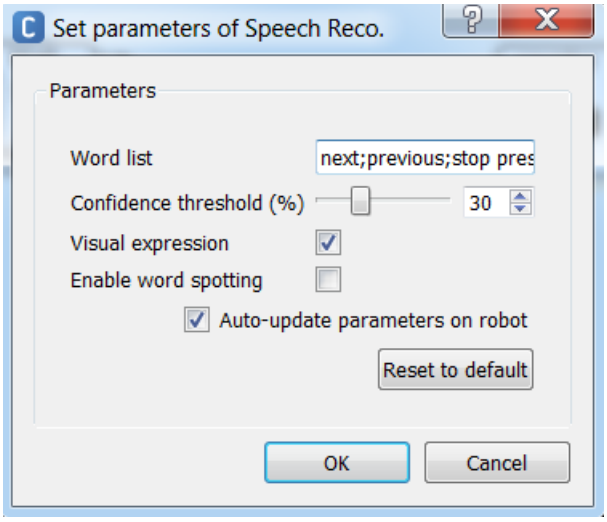


Figure 10. Speech Recognition parameters

After recognising a valid answer from a human, Pepper will send it (as a string variable) to the next box, *counter*.

As we will see next, we must add all possible answers to the counter Python box if we want our behaviour to do something with each command.

[28] http://doc.aldebaran.com/2-4/software/choregraphe/objects/box_optional_components.html

3.1.4 Counter

When the behaviour starts, it defines an integer called “counter” to value 1. This is needed because Pepper has no way of retrieving from its memory what image is currently displayed on its tablet.

The counter refers to which image is currently being shown on the robot’s tablet. Every time the audience says “next” or “previous”, the counter increases or decreases by 1, respectively.

Then, the robot shows the image that equals the counter’s value, and speaks, saying the text that belongs to the file with the counter’s value.[29]

When there is no image with the counter’s value, in other words, the last slide of the presentation has been reached, the behaviour ends and Pepper goes back to Autonomous Life.

```
class MyClass(GeneratedClass):

    global counter
    counter = 1

    def __init__(self):
        GeneratedClass.__init__(self)

    def onLoad(self):
        #put initialization code here
        pass

    def onUnload(self):
        #put clean-up code here
        pass

    def onInput_onStart(self):
        #self.onStopped() #activate the output of the box
        pass

    def onInput_answer(self, p):
        global counter

        if p == "next":
            counter += 1
            self.counter(counter)
        elif p == "previous":
            counter -= 1
            self.counter(counter)
        elif p == "stop presentation":
            self.onStopped()

        pass

    def onInput_onStop(self):
        self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
        self.onStopped() #activate the output of the box
```

1

2

Figure 11. Counter box code

[29] As referenced in 3.1.1. Show Image and 3.1.2. Animated Say

1. Global variable definition

Defining “counter” as a global variable will allow us to use it in all the functions inside the current class, which is necessary for our presentation.

If counter wasn’t global, we would need to find another way of allowing our functions to tell each other which was the current slide. Doing it this way, though, allows it to do it in an uncomplicated way that should be extensible to new functions as well.

2. “answer” input

We added an “answer” input to our Python box, with an “onStart” nature and a “String” type. This means that every time the *Speech Recognition* box sends an “answer” signal, we will execute the code defined inside this function.

The code itself is very simple. Depending on the answer given, we change the counter’s value and send the new value through an output with the “Punctual” nature and the “Number” type to three places: *Show Image*, *Animated Say* and the “onStop” input of *Speech Recognition*, so Pepper stops listening to what people say until it is done talking about the next slide.

Since this box is started the first time *Speech Recognition* sends an answer, and is never stopped until the presentation ends, it keeps track of the counter automatically. However, if we stopped the box after sending an updated counter value, we would have to inform it of the current value of the counter every time it was launched. To avoid this, we just leave the box running during the whole presentation, which doesn’t cause any memory problem because it’s a very light Python box.

3.2 Application creation

We have the presentation behaviour ready and working with files on Pepper's memory.

Now we need a Windows executable that does the following:

1. Connects to Pepper using ssh.
2. Deletes the previous files.
3. Uploads the new images of the presentation.
4. Allows the end user to write what Pepper must say in each slide.

This Python script achieves this, and it is explained in detail later:

```
#!/usr/bin/env python
"""Friendly Python SSH2 interface."""

import os
import tempfile
import paramiko

global ipadress
ipadress = "10.42.0.76"


class Connection(object):
    """Connects and logs into the specified hostname.
    """

    def __init__(self,
                 host,
                 username = [redacted],
                 private_key = None,
                 password = [redacted],
                 port = 22,
                 ):
        self._sftp_live = False
        self._sftp = None
        if not username:
            username = os.environ['LOGNAME']

        # Log to a temporary file.
        templog = tempfile.mkstemp('.txt', 'ssh-')[1]
        paramiko.util.log_to_file(templog)

        # Begin the SSH transport.
        self._transport = paramiko.Transport((host, port))
        self._tranport_live = True
        # Authenticate the transport.
        if password:
            # Using Password.
            self._transport.connect(username = username, password = password)
        else:
            # Use Private Key.
            if not private_key:
                # Try to use default key.
                if os.path.exists(os.path.expanduser('~/.ssh/id_rsa')):
                    private_key = '~/.ssh/id_rsa'
                elif os.path.exists(os.path.expanduser('~/.ssh/id_dsa')):
                    private_key = '~/.ssh/id_dsa'
                else:
                    raise TypeError, "You have not specified a password or key."
```

1
2
3

```
            private_key_file = os.path.expanduser(private_key)
            rsa_key = paramiko.RSAKey.from_private_key_file(private_key_file)
            self._transport.connect(username = username, pkey = rsa_key)

    def _sftp_connect(self):
        """Establish the SFTP connection."""
        if not self._sftp_live:
            self._sftp = paramiko.SFTPClient.from_transport(self._transport)
            self._sftp_live = True

    def get(self, remotepath, localpath = None):
        """Copies a file between the remote host and the local host."""
        if not localpath:
            localpath = os.path.split(remotepath)[1]
        self._sftp_connect()
        self._sftp.get(remotepath, localpath)

    def put(self, localpath, remotepath = None):
        """Copies a file between the local host and the remote host."""
        if not remotepath:
            remotepath = os.path.split(localpath)[1]
        self._sftp_connect()
        self._sftp.put(localpath, remotepath)

    def execute(self, command):
        """Execute the given commands on a remote machine."""
        channel = self._transport.open_session()
        channel.exec_command(command)
        output = channel.makefile('rb', -1).readlines()
        if output:
            return output
        else:
            return channel.makefile_stderr('rb', -1).readlines()


    def close(self):
        """Closes the connection and cleans up."""
        # Close SFTP Connection.
        if self._sftp_live:
            self._sftp.close()
            self._sftp_live = False
        # Close the SSH Transport.
        if self._tranport_live:
            self._transport.close()
            self._tranport_live = False

    def __del__(self):
        """Attempt to clean up if not explicitly closed."""
        self.close()

    def deletefiles(self, server = ipadress, username = "nao", pkey = "salt"):
        wantdelete = raw_input("Do you want to delete the existing presentation? (y/n): ")
        if wantdelete == "y":
            ssh = paramiko.SSHClient()
            ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            server = ipadress
            ssh.connect(server, 22, username, pkey)

            sftp = ssh.open_sftp()
```

4

```
            # Updated code below:
            #for NAOqi 2.5 -> path = "/data/home/nao/.local/share/\
            #PackageManager/apps/presentation1-8c8a24/html/"
            path = "/var/persistent/home/nao/.local/share/\
PackageManager/apps/presentation1-8c8a24/html/"
            filesInRemoteArtifacts = sftp.listdir(path)
            for file in filesInRemoteArtifacts:
                print path+file
                sftp.remove(path+file)

            # Close to end
            sftp.close()
            ssh.close()

        else:
            print "The previous presentation was not deleted."

    def userinput(self):
        #folder is where the pictures are, fullpath is the containing directory
        folder = raw_input("Enter the directory containing your presentation: ")
        fullpath = os.path.split(folder)[0]
        if folder[-1] != "\\" or folder[-1] != "/":
            folder = folder + "\\"
        if fullpath[-1] !="\\" or fullpath[-1] != "/":
            fullpath = fullpath + "\\"

        #initiating counter
        c = 0

        #creating a text directory if it doesn't exist
        newpath = fullpath + "text\\"
        if not os.path.exists(newpath):
            os.makedirs(newpath)

        #connect to ssh
        self._sftp_connect()
        pepperpath = "/var/persistent/home/nao/.local/share/\
PackageManager/apps/presentation1-8c8a24/html/"

        for picture in range(len([name for name in os.listdir(folder)\
                                  if os.path.isfile(os.path.join(folder, name))])):
            #keep track of file number
            c += 1
            #define the file name
            filename = "slide"+str(c)+"_tts.txt"
            #prompt for text
            text = raw_input("Write what you want Pepper to say in slide " + str(c) + ": ")

            #write said text and move file to needed directory
            temptext = open(filename,"w")
            temptext.write(text)
            temptext.close()
            print folder+filename
            print folder + "text"
            os.rename(filename, fullpath + "text\\" + filename)

            #upload files to remote server (pepper)
            self._sftp.put(fullpath + "text\\" + filename, pepperpath + filename)

            #delete the files
            os.remove(fullpath + "text\\" + filename)
```

5

```
        for picture in os.listdir(folder):
            print picture
            if picture[-4:] == ".PNG":
                self._sftp.put(folder + picture, pepperpath + picture)
            else:
                print "Error while uploading pictures: Wrong format. \
                        Make sure the file extension is '.PNG'"
```

```
6   def main():
        """Little test when called directly."""
        # Set these to your own details.
        global ipadress
        ipadress = raw_input("Enter Pepper's IP adress: ")
        myssh = Connection(ipadress)
        myssh.deletefiles()
        myssh.userinput()
        myssh.close()
        print "Success!"

    # start the ball rolling.
    if __name__ == "__main__":
        main()
```

Figure 12. Application code

Firstly, it's important to mention that most of the code is taken from an open source code with the basic methods for Paramiko, of which we only use *__init__*, *_sftp_connect* and *put*. You can check the source code on its Github[30], inside the directory "ssh".

There is no need to understand the source code thoroughly, we only need to know that *_sftp_connect* is a method to connect to a remote server with SSH, and that *put* is a method that allows us to upload files from our machine to the remote server.

For that reason, the author has chosen to skip on explaining every detail of the code taken from the Github source, and focus on the most important points of the application's code instead.

1. Imported modules

Module *os* allows the usage of operating system dependent functionalities.[31]

Module *tempfile* allows the creation of temporary files and directories.[32]

Module *paramiko* is a ssh implementation for Python.

[30] https://github.com/tadeck/ssh-matic
[31] https://docs.python.org/2.7/library/os.html
[32] https://docs.python.org/2.7/library/tempfile.html

2. Class definition

Class oriented programming is a characteristic of Python. We're not going to explain this class thoroughly, since it's source code from Github that allows us to create an ssh connection to a remote server (Pepper, in this case), and we don't need to understand every bit to make use of it.

The class is basically a way of defining an ssh connection as an object, but the author added two functions which are explained later, and help us achieve our application's objective.

3. Function *__init__*

The only thing the author edited from the source code was the username and password. Since we need Pepper's username and password to access its memory, having it in the code will allow anyone to use the application without necessarily knowing the details about accessing Pepper's memory. They are covered for security purposes. If you need to access Pepper, consult one of its current developers to know its username and password.

Vital information: The application won't work if the username and password are incorrect. This means that it won't work in a Pepper robot that isn't the original, and it will stop working if the username or password are changed. If this happens anyone is free to open the application's source code and edit the logging details.

4. Function d*eletefiles*

As seen in the code, firstly the end user is prompted about deleting the old files. The answer should always be "yes" if a new presentation is being uploaded to the robot.

A Paramiko object is defined to connect to the remote server, and the path inside Pepper's memory is always the same for the 2.4 NAOqi OS. The code includes the 2.5 NAOqi OS path, in case anyone needed to work with an updated robot.

Then, all the files in the "html" folder are recursively deleted.

5. Function *userinput*

The end user is prompted for the directory containing the images, which must be named “Slide1.PNG”, … “SlideN.PNG” for the presentation behaviour to work properly. Then the script checks if the format of the directory path is correct, and corrects it otherwise.

We initiate a counter that will allow us to keep track of what slide are we in. Then a temporary folder named “text” is created. Afterwards we connect to Pepper using Paramiko and define the path to the “html” directory in its memory again.

We then have a *for* loop that creates all text files necessary as the end user enters text and uploads them to Pepper as they are created.

Finally, another *for* loop uploads all the slides (images) to Pepper as well.

6. Calling *__main__*

Since it was a small line of code, the author decided not to define another function to enter Pepper’s IP Adress.

After that, we create an object with the new class, to be able to call its methods, with the IP that the end user just was prompted for.

Then *deletefiles* is called, to leave the directory in Pepper empty, followed by *userinput*, and finally *close* to end the ssh connection.

The print function lets the end user know the presentation upload was successful.

3.3 Potential problems

There are a lot of problems that can arise, given the nature of both the application and the presentation behaviour. Therefore, this is intended to be a list that makes it as easy as possible to spot the origin of the problem in case the application stops working.

Troubleshooting list:

- Different robot

  Using a robot that is not the original will run into one main problem: the username and password to access its internal memory aren't the same anymore. Try asking the person working with the robot at the time of the issue.

- Another NAOqi OS version

  A different OS version (i.e. NAOqi 2.4 and 2.5) might store the uploaded behaviours in different directories. Use a client for remote access to find out the new location (FileZilla is a good option).

- Network connection

  Pepper only tries to connect to a Wi-Fi network once, during start-up. To know its IP address, press the button on its chest once, and it will say it out loud. Otherwise, if it couldn't connect, it will say so. Try to connect to your Wi-Fi hotspot using your phone, if you're able to, so should Pepper, given it's close enough to your computer. Try restarting the robot.

- Behaviour name change (Path change)

  The presentation behaviour is called "presentation1". Using Choregraphe, we can check the **Application ID** by going to "Project properties" this ID is the path to the directory where the files are stored. In this case, it's "presentation1-8c8a24". If it changes, the path will not exist and neither the application or the behaviour will work – both would need to be fixed.

- Wrong image format

  The behaviour is written such that the presentation images must have the ".PNG" extension, in **uppercase letters**. The reasoning being that Microsoft PowerPoint saves the images that way, instead of in lowercase letters, and the *Show Image* box needs the name of the full file to show the image, including the extension.

- Wrong image name or directory

  Following the previous point, all images must be named "Slide1.PNG", … "SlideN.PNG" so the application works properly. Also, when prompted for the directory, as indicated in the User Manual, the images must be the only files in said directory.

- User Manual usage

  It is recommended that you check the User Manual if the application isn't working, since it contains relevant information on how the application works and the different steps there are to it. It might give you an idea of what the problem could be.

Appendix A

User Manual

This user manual will go through all the steps and considerations needed to use the Presentation Application.

1. Before application usage

There are a few steps to follow before using the application. These are very important to make sure you will be able to upload your presentation.

- **Saving your PowerPoint presentation as PNG images**

  Simply go to “Save As” in PowerPoint, and choose the image format “PNG”.
  Then, select the option to save all slides.
  They will automatically be named “Slide1”, “Slide2”, etc., so now you are ready to enter the directory path when you are prompted later.

- **Set up a Wi-Fi network**

  You must set up a Wi-Fi network to be able to communicate with Pepper. The network Pepper automatically connects to.

  You can either create an ad hoc network using Windows, or use 3rd party software. The author recommends the second option, using OSToto Hotspot[1], as it is easy to set up. Otherwise use Virtual Router Manager.

[1] http://www.ostoto.com/products/wifi-hotspot.html or https://virtualrouter.codeplex.com/

After installation, just set the network settings to:
Network Name (SSID): oslwifi
Password: osllab2013
Select the shared connection available other than “None”.
Click “Start Virtual Router”.

It is recommended that you double check if the Wi-Fi network works properly by connecting to it with your phone. If you’re able to, so should Pepper.

- ## Turning Pepper on[2]

Make sure Pepper is close to you and your computer, since the Wi-Fi network created by a computer is usually not very strong.
Then, turn Pepper on by pressing the button under its tablet once. Its shoulders should blink in white as it boots, and the boot process should take between one and two minutes.
Wait for Pepper to say “ognak gnouk” out loud, which means it has finished booting.

If you need to move Pepper, please check the official documentation.[3]

- ## Check if Pepper is connected to the network

Once it has finished booting, click the button under Pepper’s tablet only once (don’t hold).
It should say “Hello, I’m Pepper. My internet address is XXX.XXX.XXX.XXX” if it has connected to your newly created hotspot properly.
Please write down or remember Pepper’s IP address.

If Pepper says “I can’t connect to the network” please check the first step again and make sure you can connect to the Wi-Fi with your phone, and make sure Pepper is not more than 1 meter away from the computer.

It’s important to know that Pepper only attempts to connect to the network once, while booting up, so you must turn it off and on again.[4] To turn it off, press and hold the button under the tablet for a few seconds, until you hear it say “gnouk gnouk”. Then release and wait for it to turn off.

[2] http://doc.aldebaran.com/2-4/family/pepper_user_guide/turn_on_pep.html
[3] http://doc.aldebaran.com/2-4/family/pepper_user_guide/move_manually.html#move-manually-pepper
[4] http://doc.aldebaran.com/2-4/family/pepper_user_guide/turn_off_pep.html

2. Application usage

There are a few steps to follow while using the application. Launch the executable file called “Uploading Application”.

- Enter Pepper’s IP. If you don’t know it, press its chest button once.

- Then, a prompt asks you “Do you want to delete the previous presentation”. You should always say YES (write “y” and press the enter key).

- Now you must enter the path to the directory your presentation is in. It is very important that the presentation slides are saved with the name “Slide1”, “Slide2”, etc. They also must be “.PNG” format. The directory should **only** contain the images.
  To save a Microsoft PowerPoint as images, just use “Save As” and choose “.PNG” format. Then, choose to save all slides.

- The application will then ask you to enter what you want Pepper to say in each slide, in numerical order. You can use previously written text and paste it for each slide, or write it manually in the application. The first option is more recommended. You can leave it blank as well.

- When you’ve entered text for each slide, the application will upload it to Pepper and then close.

3. After application usage

After uploading the new presentation to Pepper, you only need to turn it off, and back on. That is needed to reset its cached memory, otherwise the old presentation might show instead of the new one, and it might cause confusion.

There are references to the official documentation saying how to turn Pepper off or on in the previous page.

Please consider that the quieter the environment is, the better Pepper will be able to understand the people using the interactive presentation.

Appendix B

Code extensions

To make the presentation more complete, a few code extensions have been added after the project was turned in. They are explained here so you can edit it as you want if it's necessary.

1. Explanation at the start

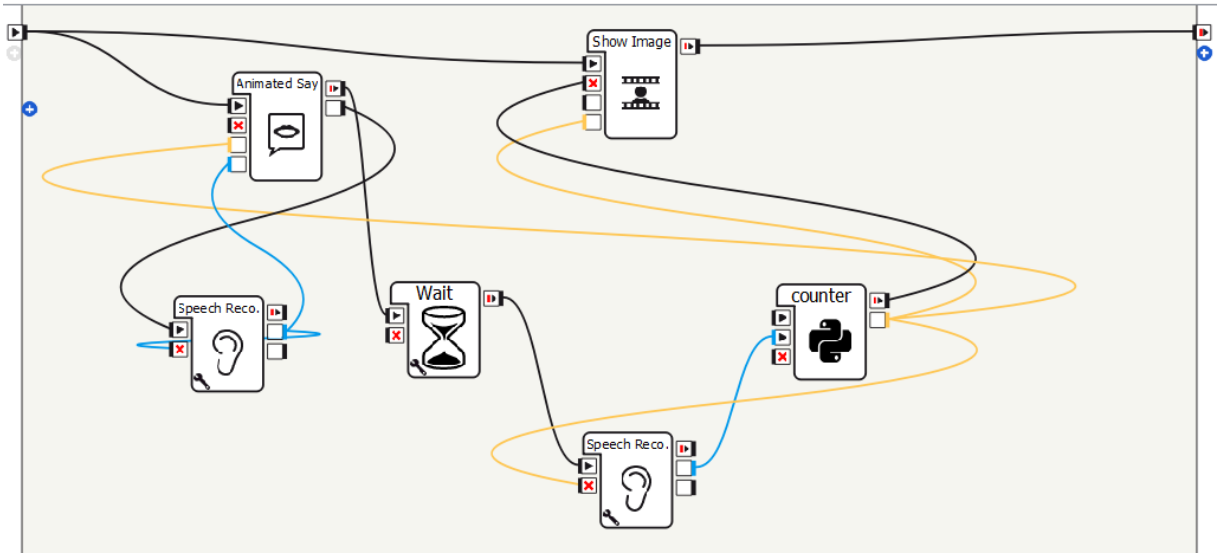
A slide "zero" shows how to interact with Pepper during the presentation, and Pepper says out loud: "Hello! Welcome to my interactive presentation. You can say "next" or "previous" to navigate it. Please say "stop presentation" before leaving! Say next to start."

This is accomplished by adding "Slide0.PNG" and "slide0_tts.txt" files to the *html* folder, and editing the Uploading Program's code to avoid deleting it while deleting all other files.

The counter is initiated at 0 instead of at 1, making slide zero the first one shown every time the presentation is launched.

2. Preventing Pepper from repeating itself

If Pepper goes to a previous slide, instead of saying all the text again, it will ask “Do you want me to repeat? Please say “yes” or “no””.



The only difference from the original behaviour is the extra Speech Recognition box under the Animated Say box.

Animated Say box now keeps track of what slides it has talked about with a list called “slides”. It adds the counter’s number to the list if it isn’t already there, and if it is, it means it’s already been in that slide, so it calls the speech method to ask the user if they want the robot to repeat itself, and activates the speech recognition box.

When an answer is given, the robot either repeats itself or goes to the other Speech Recognition box, where it listens for “next”, “previous” or “stop presentation” again.

Webgraphy

Information about Pepper (SoftBank Robotics) – 03/2017 - - https://www.ald.softbankrobotics.com/en/cool-robots/pepper

Article about Pepper spread and success in the US (Recode.net) – 03/2017 - https://www.recode.net/2017/1/4/14171436/softbank-robot-pepper-sales-brick-and-mortar-retail-ces

Article about Pepper (Inc.com) – 03/2017 - https://www.inc.com/will-yakowicz/pepper-the-robot.html

Technical specifications (SoftBank Robotics Pepper) – 03/2017 - http://doc.aldebaran.com/2-4/family/pepper_technical/index_pep.html

Trigger sentences and conditions tutorials (SoftBank Robotics) – 04/2017 - http://doc.aldebaran.com/2-4/software/choregraphe/tutos/create_interactive_activity.html

Python Definition (Official Documentation) – 04/2017 - https://docs.python.org/2/faq/general.html#what-is-python

Python License of use – 04/2017 - https://docs.python.org/3/license.html

Stack Overflow - http://stackoverflow.com/

Python tutorials – 04/2017 - https://docs.python.org/2/tutorial/index.html

Free online Python edX course – 04/2017 - https://www.edx.org/course/introduction-computer-science-mitx-6-00-1x-10

Codeacademy Learn Python tutorial – 04/2017 - https://www.codecademy.com/learn/python

Python characteristics (from A Byte of Python) – 05/2017 - https://python.swaroopch.com/