Using a social robot to support PowerPoint presentations

Wladimiro Arce

Master project submitted under the supervision of
Bram Vanderborght & Dirk Lefeber

The co-supervision of
Pablo Gomez

Academic year
2016-2017

brubotics®
Brussels Human Robotic Research Center

Preamble

Acknowledgement

This project has been developed under the supervision of Bram Vanderborght, Dirk Lefeber and Pablo Gómez, in the *Brussels Human Robotics Research Center*, formerly know as *BruBotics*.

Objectives have been made achievable thanks to their support and because of all those persons, including my family and friends, who have made my stay in Brussels possible and enjoyable.

Notations and conventions

- **Bold** is used to emphasize key concepts
- *Italic* indicates names, such as commercial brands, products, files, or procedures followed, such a sequence of actions done in a computer software
- `URLs` are shown in that way. Also, in the electronic version, the hyperlink can be followed
- `Code` represents any kind of word of the developed scripts inserted into the text. On the other hand, fragments of code are typeset in the following way:

```
def hello_world():
    #prints Hello world! to the console
    print "Hello world!"
    return
```

Code 1: Python example

About

A complete overview of the project can be found in the website

`https://wladiarce.github.io/PepperPresenter`

The full source code of the here developed application is publicly available under a Github repository, at the following direction:

`https://github.com/wladiarce/PepperPresenter`,

and it is distributed under the *MIT* license:

`https://opensource.org/licenses/MIT`

This document has been developed using LaTeX, a very well known among academia document typesetter, which uses as a basis the TeX typesetting language allowing to compile and render documents in an efficient way.

To do that, an on-line tool has been used. It is *Overleaf*[1], a web-app designed to produce LaTeX documents, free, and collaborative.

References shown were managed using *JabRef*[2], and open source bibliography reference manager, for easiness to a posterior attach to the document.

[1] `www.overleaf.com`
[2] `www.jabref.org`

Contents

1 Motivation **3**

2 State of the art **5**

3 Objectives **8**

Defining the development environment

4 Overview of the project **11**
4.1 The robot: *Pepper* 11
4.1.1 Pepper 12
4.1.2 Full specifications list 14
4.2 The application:
Pepper Presenter App 14
4.2.1 Snapshot of the application 14
4.3 Summary 15

Development of the application

5 *PowerPoint* presentation files (.pptx) **17**
5.1 Managing .pptx with *Python* 19
5.2 Extracting the presentation notes 20
5.3 Summary 21

6 Controlling *Pepper* **22**
6.1 Autonomous Life 22
6.2 Making the robot talk 22
6.3 User defined gestures 23
6.4 Pointing 24
6.5 Summary 26

7 Adding the slides **27**
7.1 Exporting the presentation 27
7.2 Making it easy: PowerPoint add-in 28
7.3 Summary 30

8 User-friendly application **31**

8.1 The tool: *htmlPy* 31
8.2 Front-end and user input 31
8.3 Back-end 33
8.4 Summary 34

9 Validation **35**
9.1 First section: expectations 35
9.2 Second section: presenting 35
9.2.1 Results from slide 1 36
9.2.2 Results from slide 2 37
9.3 Third section: conclusions 38
9.4 Summary 38

Conclusions and future work

10 Conclusions **39**
10.1 Objectives 39
10.2 Using it in a real environment 39
10.3 Further development 40
10.3.1 Installation package 40
10.3.2 Updating the app 40
10.3.3 New state of the art functionalities 40
10.3.4 More operation environments 41

Annexes and Bibliography

A How to install **45**
A.1 Installation instructions 45
A.1.1 Install the Presenter Application 45
A.1.2 Install the *PowerPoint* add-in 46

B How to annotate a presentation **47**
B.1 Advice to create good presentations for *Pepper* 47

C Joint angles calculations for pointing **48**

List of Figures

2.1 Robot NAO doing a presentation 6

4.1 Keepon Robot. 11
4.2 Pepper, a social robot 12
4.3 Scheme of the Pepper Presenter Application 15

5.1 Inside of a .pptx file 17
5.2 Object model of a PowerPoint Presentation 20

6.1 The space where the robot does the presentation 25

7.1 The ExporterApp user form. 29
7.2 The new ribbon control to access the Exporter application . . . 30

8.1 The app front end. 32
8.2 The extended user input. 33

9.1 What do you expect from a robot doing a presentation? 35
9.2 What has the robot mostly expressed? 36
9.3 Where has the robot pointed? 36
9.4 What has the robot mostly expressed? 37
9.5 Where has the robot pointed? 37
9.6 In which measure you think this will be applied in the future? . . 38
9.7 How you felt about the robot presenting? 38

A.1 PowerPoint Add-ins options 46

List of Code

1 Python example iii
5.1 Installing Python modules 19
5.2 Accessing a slide's notes 21
6.1 Using ALAnimatedSpeech 23
6.2 Including user defined gestures 24
6.3 Pointing 26
7.1 VBA snippet to export a presentation and its slides as images in a structured way 28
A.1 Installing the app. Python dependencies 45

Using a social robot to support PowerPoint presentations

1 Motivation

Giving a **presentation** is not an easy task.

A very interesting and maybe complex topic can be shown in an attractive manner that is understood by everybody, but at the same time, a so simple thing that seems understandable by even a kid can be messed up making the spectators lost in the first minutes.

Although there are many tools available to create presentations aid material like visual slides, infographics, videos or interactive quizzes, amongst others, having some or all of those while presenting does not mean that the quality of the presentations is going to be good, independently of the quality of the content.

At the end, those things help, but the key point to master a presentation and fully attract the audience resides on the presenter and **his/her/its** social and communicative skills.

Being able to effectively communicate and show ideas is such an important thing nowadays that from very little we are educated to acquire it. During our whole life we are going to use it.

Many studies about psychology and social interaction have been done about this topic[1][2]. To gain the attention of the spectators, they need to feel they are being addressed. Knowing who you are talking to is the first step to reach to the public. Verbal and non-verbal communication are going to be crucial abilities.

Then, the presenter has to be consistent and focused on what the topic of the presentation is. The right amount of information needs to be given.

But that is not all. With those abilities you can be an ace-presenter, but you need to show your ideas to someone, or to a large group. And here is where a lot of people, and even people who can give really good presentations, get overcome by **stage fright**. It is worthless to be an excellent communicator if you can not express your ideas to a general public.

This is very related to the psychology of each person, and has nothing to see with how a good communicator is or not. Regrettably, it is one of the most difficult things to beat.

The easy solution would be to be off stage and instead give the presentation

remotely or showing a video, but studies have demonstrated (with robots) that being physically present is the key to interact and attract the participants[3]. Stronger behaviors and attitudes where found when people where in front of a physical agent, compared with seeing it through a screen, no matter if it was physically or virtually created.

Summarizing, giving a presentation being physically present is one of the things makes it better, but not everyone is able to stand in front of a crowd, or even it can become a repetitive task that makes it necessary to search for alternatives to do it. Physical embodiment means to have something tangible, and as of that, a nice idea would be to put some other thing than yourself facing the spectators, and transmitting the ideas to them.

And that is the motivation of this project, taking advantage of the actual technology available in the field of **social robots** and use them as a platform to **give presentations**.

2 State of the art

Robots are actually in the best conditions to become, once programmed, more intelligent and autonomous machines than ever. In the last decade, a branch of robotics has had a very active development, focusing on **human-robot interaction**[4].

Within that wing of development, there is a group of interest for what concerns the topic of this project: the **human-robot social interaction**.

Those robots, called **social robots**[5], are capable of interacting with humans in a quite autonomous way. They differentiate from other types of robots in the fact that they are not limited to machine-machine interaction. Computer vision, speech recognition and computer speech together with Natural Language processing algorithms and decision making software have led to physically embodied machines that can follow social behaviors, being able to maintain a conversation or act as if they were humans.

For the interaction to be smooth and natural at its most, those robots even recognize how a human is feeling, based on different factors it perceives and models to calculate that, and can adapt and respond in such a convenient manner, having the possibility to express those emotions by themselves gesturing, showing facial expressions and changing the voice[6].

The actual examples of social robots available is very wide; different types of embodiment can be found (human shaped, animal shaped or other type of representations).

But currently, the state of the art in the development of this machines is so that this sector is not only limited to high end technology companies. With initiatives like *OPSORO*[1], which is an open source platform created for this type of robotics, one can create its own embodiment of any kind and have a custom home made fully operative social robot.

Applications of social robots are endless. You can program them to be a hostess or a waiter in a restaurant[7], receiving you and serving the food; to check in guests in a hotel[8]; to give tours in a museum[9] or simply to be waiting for you at home and interact with you to order food, control the heating or open or close the doors[10].

Some very interesting cases that can be currently found in the use of those

[1] `www.opsoro.be`

robots is in education and health care, like companions for elderly people or to treat and help people with autism[11][12][13].

But another use, more relevant to what motivates this project, is to use such robots to talk in front of an audience, doing presentation of whatever topic they are programmed to. To have a better idea of how a social robot is with such a behavior, the example of the *NAO Presenter App*[14] can be seen. It is an app, commercially distributed, that adds to the social robot *NAO* the ability to make presentations, interacting with the audience.

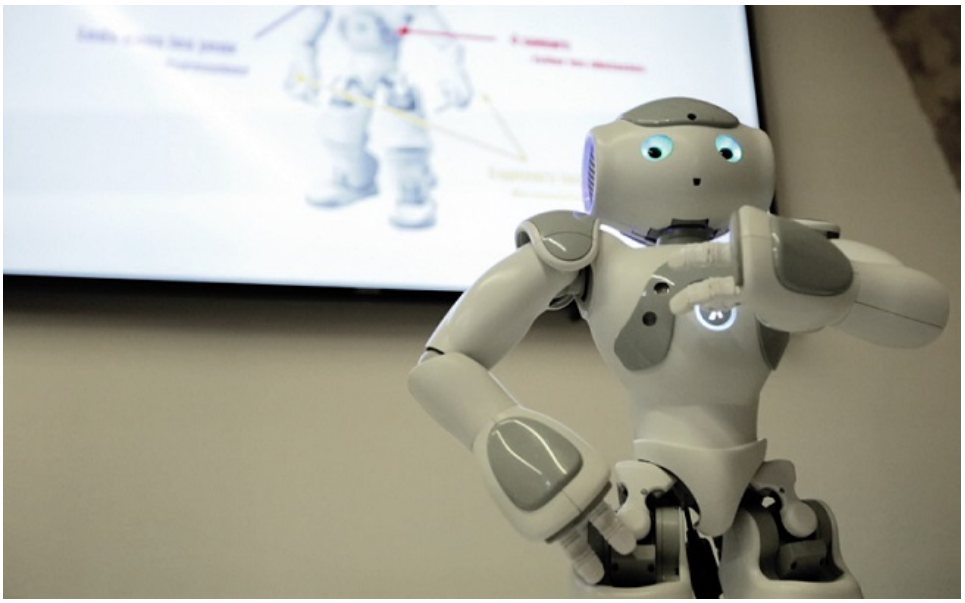

Figure 2.1: Robot NAO doing a presentation

On the other hand, there is a huge research and development around the psychology of human interaction: not only the way one talks, but also the way one behaves. How you express yourself with your body language. The body language and the presence defines the way you have impact in others. The more assertiveness, confident and powerful you think you are, the most you will convince your audience. [15]

Focusing this interaction in the topic of interest for this text, a presentation needs to be like a story, it should not be plain. If the presentation goes back and forth the spectators keep more interested in it. The idea needs to be shown, but what it can be and what it is not need to be introduced too. The presenter needs first to believe his/her own idea, and then he/she will be able to make others believe in it.

And this is done not only communicating to others (in a verbal and non verbal way), but also attracting the attention with supplementary elements like images and videos.[16]

So the key point here is that this is an originally human behavior, but with the current development of the social robots stated previously on this chapter, this is an attitude that can be emulated, and it has already started to be done [14], in a physically embodied device.

3 Objectives

In those 2 last chapters, a brief discussion about the motivation of this project and the current development of social robots and the psychology and behavior under doing public talks has been given.

This project then is going to be focused in using the power of the social robots presented in Chapter 2 as a platform to do presentations in a proper way as has been stated in Chapter 1.

Being designed for socializing, making such a robot present a slide show can be the key to do attractive and appealing presentations, as well as impressing the attendants, if the necessary behaviors to attract the public are implemented on it.

In this case, to target the highest number of users when the project is finished, market share is what really matters. The most used tool, and very powerful if used in the right manner, to make presentations and keynotes is without doubt *Microsoft PowerPoint*. It is not a free tool, but despite of that it is widely spread and licensed by most of the academic institutions and companies, having and estimated usage of around 90% of all the presentations development tools.

Then, an application will be created, able to read *PowerPoint* files. To tell the story of the idea shown in the presentation, the app will read pre-formatted text available in the notes, and will make the robot do the presentation and talk about what it has been told.

Should be when finished easily portable among platforms. Many examples of social robots are available in the market, and each of them, if they have a open development platform, will have its own software development kit.

Even thought, the project will be focused only on one as starting point, but when finished, enabling such a behavior importing the algorithms to other languages or platforms to run in other robots should not be a big problem.

Making a summary on the characteristics it should have, the app will:

- Read **PowerPoint** files.
- Make the robot present those slide shows based on the text it finds in the **presentation notes**.

- Gesture and express **emotions**, autonomously or user-defined.
- Make the robot able to point, emphasizing different zones in the screen.
- Be **portable**, among different operating systems.
- Be **user-friendly** and easy to use, to target a greater audience.
- Be **open source**

Having stated all the requirements, in the following chapters a detailed description of each development stage will be found.

4 Overview of the project

4.1 The robot: *Pepper*

Actually, there are many social and humanoid robots available in the market, and many more out of it for research and investigation purposes.

To get this project to reality, a starting platform needs to be chosen. Amongst all the options available, the presenting application will be developed for the robot *Pepper*, manufactured by *Aldebaran Robotics*.

Figure 4.1: Keepon Robot.
Its embodiment will not permit to use all the functionalities needed in a presentation.

The reasons to choose this robot, which will be presented in the following section, are that it has all the necessary ways of interaction needed for the purpose of this project, it is actually one of the most affordable that can be openly purchased in the market, and more important, that it can be easily programmed by anyone thanks to its open platform with an active development community.

Other social robots, more affordable, can be found, like the *Keepon* robot (Figure 4.1), but their embodiment will not permit to accomplish all the necessary behaviors described in the firsts chapters to do fully attract the presentation attendants, due to not having a human shape, or not having text to speech capabilities that allow it to talk.

4.1.1 Pepper

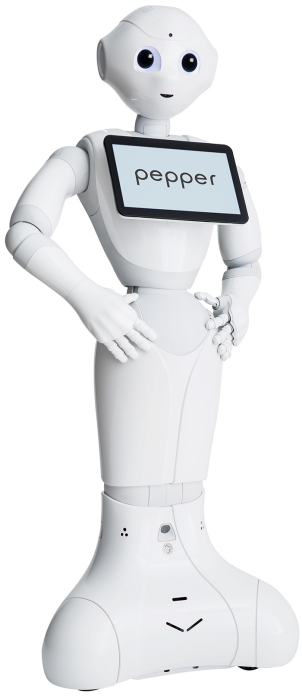

Figure 4.2: Pepper, a social robot

Pepper, as shown in Figure 4.2, is a 120 cm tall social robot with autonomous life capable of perceiving human emotions, from joy to anger, and so on, based not only in the verbal language you use, but also in the non verbal communication it detects: the gestures you do with your face or even the changes of tone in your voice. As well, it is able to interpret emotions and gestures in order to better express himself.

Physically, *Pepper* is human-shaped in its upper part, consisting in two arms attached to the sides of its chest, with five fingers per hand. Its head has 2 eyes which can change in terms of light color and intensity, giving a visual feedback of its state, as well as with the LEDs located in its shoulders.

In its lower part the robot has a simpler design, with a three-wheeler base that allows it to move, and a not so much articulated body.

All the interactive functionalities it has are in great part thanks to:

- Microphones and speakers, which allowing not only to hear, but also to detect where the source comes from.
- A 3D camera and two HD cameras, to recognize his environment and faces.
- Internet connection.
- A tablet, capable of showing information in a more visual way and allowing a different way of interaction.
- 3 multi directional wheels in its base allowing movements up to 3 km/h.
- More than 15 degrees of freedom in his body.

To move around, *Pepper* has in its base a series of sensors mapping its surroundings. It consist on 6 laser line generators, looking at both sides and at the front, 2 infra-red sensors and 2 sonars. 3 bumpers are also available to detect collisions with the near environment.

The heart of the robot is a microcomputer powered by a *1.91GHz QuadCore Intel Atom* microprocessor accompanied by a *4GB DDR3 RAM*. It runs a custom distribution of *Linux* called *NAOqi*, designed and optimized for the purpose of the robot, containing all the necessary libraries and software that endows it with life.

Height	1210 mm	**Depth**	425 mm
Width	480 mm	**Weight**	28 kg
Battery	Li-ion 795 Wh	**Connectivity**	*WiFi / Ethernet*
Touch sensor	3x Head / 2x Hand	**Microphone**	4x Head
Speakers	2x Head	**Camera**	2x RGB / 1x 3D
Tablet	LG 10.1"	**IMU**	Gyro + Accel
Positioning	Laser + Sonar + Infrared	**DOF (Total)**	20
LEDs	Ears, eyes, shoulders	**Buttons**	Chest (ON/OFF)

Table 4.1: Pepper Specifications

The measurements shown correspond to the robot in a completely stand position, with its arms down.

4.1.2 Full specifications list

To get a technical comprehension of how *Pepper* is, more specifications should be given.

In Table 4.1 a more detailed data-sheet of the robot can be found.

4.2 The application: *Pepper Presenter App*

Having stated the robot the application will be developed for, the next step is to clarify the app specifications and how will it be initially implemented on *Pepper*.

The application will be created to run remotely from the robot. By means of that, it will be executed on a computer and the instructions will be sent to *Pepper* through a wire/d/less connection. A computer software that interacts with the robot remotely provides the freedom to create much bigger and scalable applications, with faster execution times, than the ones run locally in the target platform, with less computing power.

Amongst the different programming languages available.*Python*[17] is going to be the one used. With its own **SDK** for the *NAOqi* platform, it has many advantages against others. It is a language that been designed to achieve the requirements of an application with fewer lines of code than with other languages and a very important aspect: has a quite active development community and has turned very popular in the last years mainly for scientific and data processing applications, which makes finding information and solutions to common problems an easy task.

To display the visual aid of the presentations, it is decided to use the same program that will be created to remotely interact with *Pepper*. With this approach, the slides of a *PowerPoint* presentation will be available in the device screen, and for larger conference rooms an external display or projector will be easily pluggable.

4.2.1 Snapshot of the application

A *Python* program running on a computer will:

- Read the presentation and process all the information there.

- Control the robot accordingly, though a wireless connection.
- Show the corresponding slides in full screen to a projector or secondary screen.

The scheme of such a system is shown in Figure 4.3.

Figure 4.3: Scheme of the Pepper Presenter Application

4.3 Summary

The chosen platform to host this project is going to be the social robot *Pepper*, manufactured by *Aldebaran Robotics*, and with an open development platform.

Amongst the many ways it can be programmed, for the purpose of this application a software written in *Python* and run remotely form the robot is going to be created.

This software will be in charge of managing the .pptx files and give instructions to the robot according to what it finds on the file, as well as screening/projecting the necessary visual aids of the mentioned presentation.

5 *PowerPoint* presentation files (.pptx)

The presentation files created by *Microsoft PowerPoint*, from 2007 and above are denoted by the file extension **.pptx**.

This file extension is part of the new stack of extensions created by *Microsoft* in 2006, to be used within the newer version of *MS Office 2007* suite, and denoted as **Office Open XML (OOXML)**[18].

Those files are simply packages containing a series of XML documents that define the archive, whether it is a text document, a spreadsheet or, what is focused this project on, a presentation.

Those packages, a part of the XML documents defining the structure, contain also the additional resources like images that have been included when creating the file.

In order to better understand how an OOXML presentation file looks like, it can be easily opened changing its extension to *.zip* and using any ZIP archives manager. It can be seen in Figure 5.1.

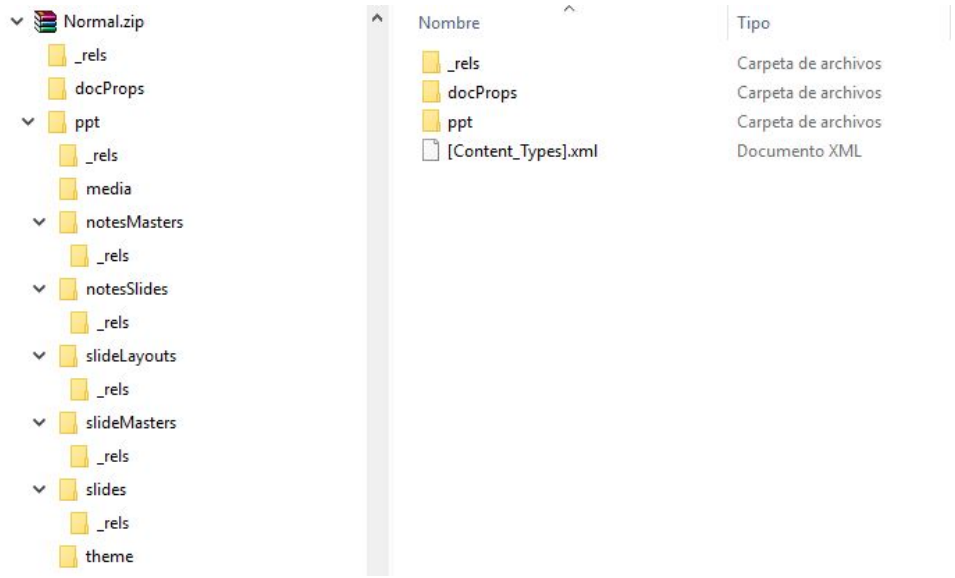

Figure 5.1: Inside of a .pptx file

Here an example presentation called **Normal** has been used. At the left side of the image the full structure is observed. In the right side, the root of the presentation is showed.

In the root of the package a file called *[Content_types].xml* and three folders are found. This structure is the same for all OOXML archives, with the variation of the here called folder *ppt*, that is related with *PowerPoint* presentations. In other documents, like *Excel* spreadsheets, this folder will appear as *xl*.

The *[Content_types].xml* is a file that defines all the **contents** of the package. Every file inside the package and its extension is listed in this XML document.

The *_rels* folders, not only present in the root directory, but also inside every other folders of the elements related to the presentation, as can be seen in the left part of the image, contain a file or files with *.rels* extension. These are in fact more XML documents, but denoted with a different extension, and define the **relationships** of the package, the elements inside of it and, in this case, the elements in the presentation.

Thanks to this structure, the relationships between the contents can be easily changed without changing the content itself, allowing a more efficient creation and posterior modification of the files.

The **contents** of the presentation are found in its respective XML files located in the folders inside the *ppt* directory. All those folders with a *_rels* in it, as *notesMasters, notesSlides, slideLayouts, slideMasters and slides*, visible in the archive tree at Figure 5.1 denote the contents that form the presentation.

This forms what is called Object Model. All of those contents are objects inside the presentations, and the software uses the relationships between them to understand how to render them.

When you include something like text in a slide or in the notes of it when creating a presentation, it is stated in its corresponding XML depending in which level of hierarchy the text was written, and added its relationship to determine where is it.

Also, other folders can be seen inside an OOXML package, but are not important right now to discuss.

5.1 Managing .pptx with *Python*

As seen, a *.pptx* is simply a structured group of *.xml* files. As so, a program to understand those files can be written. The advantage with *Python* is that it incorporates a XML parser within its standard library, with the *ElementTree XML API*[19].

By using this method, a further comprehension on the *PowerPoint* object model used in its presentations packages needs to be performed in order to be able to create a new *Python* library with classes that transcribe this information available into the different XML files to strings and the necessary structures inside the program memory.

Thanks to the active development community of *Python*, a high number of useful modules can be found in its *PyPI - Python Package Index*[20]. Those modules are add-ins for the standard library of *Python*.

They are pre-written code focused on some functionality that is not natively present in the language. By installing those modules, you can access the new classes and functions that the module provides, allowing faster development of applications, without spending time on coding something that has been already done. The extensions are easily installable by running the following command in the OS console: Code 5.1 (only works if *Python* is already installed in the machine).

```
1 pip install package-name
```

Code 5.1: Installing Python modules

After an extensive research, a module created to manage OOXML presentation files is found. It is called *python-pptx* [1] [21], developed by *Steve Canny* and available under the *MIT* license.

With this module *Python* gains the capability of reading .pptx files, and modify their content. Slides, titles, text, images, shapes and the **notes** are now fully accessible and editable.

[1] python-pptx has as dependency the lxml module, which is not included in the package and should be installed separately

5.2 Extracting the presentation notes

It has been just introduced that *python-pptx* lets *Python* access the presentation notes. To have a better insight on how this is performed, an analysis of what internally happens and how the *PowerPoint* Object Model is interpreted to get the presentation notes is going to be performed.

In the following Figure, 5.2, the object model up to the presentation notes of a *.pptx* file can be seen.

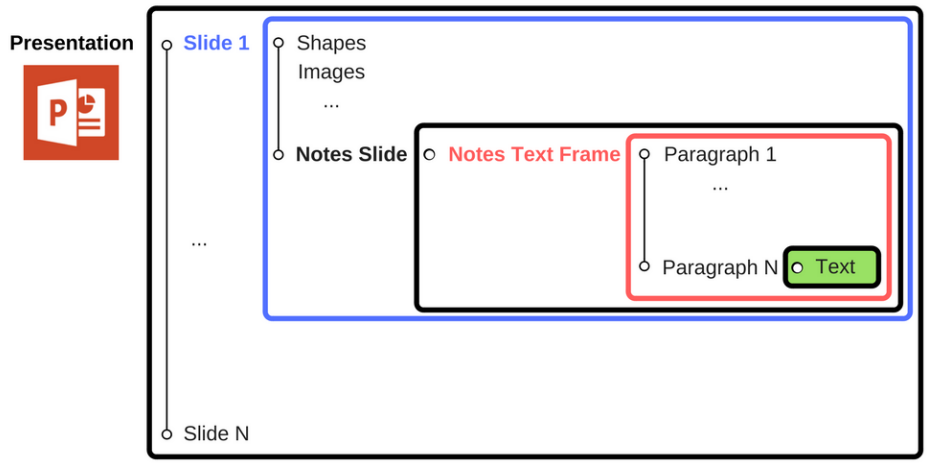

Figure 5.2: Object model of a PowerPoint Presentation
In green is shown where the notes text is located.

Notice the similarity between this object model shown here and the directories tree of a .pptx file, in the left side of Figure 5.1.

To access the notes of a presentation then, it must be done in a structured way, following the hierarchy of the elements in the file. *python-pptx* does so by having each of those objects stated in Figure 5.2 represented by a class[2].

As so, and after reading the documentation provided with the *python-pptx* module, the way to extract the presentation notes is creating instances of each *PowerPoint* object until our desired Notes Text.

[2]A class (in Object Oriented Programming) is a reusable fragment of code that acts as a template for creating objects (called instances of a class), offering also different functions and methods to implement behaviors on these objects.

As can be seen in Figure 5.2, both the **Slide** objects and the **Paragraph** objects can appear several times, depending on what contents has the presentation in it. For that reason, in the *python-pptx* module their classes have been implemented to create those objects as iterable ones, meaning that they can be used in loops to automate their access.

The final code to do so is the following, at Code 5.2. Note that to use the capabilities of this module it needs to be loaded before.

```
1  from pptx import Presentation #to use the python-pptx classes
2
3  pptx_path = "path" #the local path of the .pptx
4  pres = Presentation(pptx_path) #creates the presentation object
5
6  #loop that access all the Slide objects available
7  for slide in pres.slides:
8      notes = slide.notes_slides #create the Notes Slide object
9      text_frame = notes.notes_text_frame #creates the Notes Text
       Frame object
10
11      #loop to read all the Paragraph objects of the notes
12     for paragraph in text_frame.paragraphs:
13         print(paragraph.text) #prints the text to the console
```

Code 5.2: Accessing a slide's notes

5.3 Summary

A *PowerPoint* presentation is no more than a package containing XML documents defining the contents of the presentation and the relationships amongst them.

With *Python*, XML files are natively accessible, but to simplify the task even more, an already created module focused on understanding the specific XML of a .pptx file is available. Called *python-pptx*, it enables the programmer to transform a presentation and its contents into class instances, to perform operations on them.

From all of this objects, the one of interest for this project is the Notes of the presentation. Located at the end of the hierarchy tree of Figure 5.2, those notes can be easily extracted though a loop that reads all the slides in a given presentation, written in Code 5.2.

6 Controlling *Pepper*

The robot used for this project, *Pepper*, has many ways to interact with its environment, as presented in Chapter 4. But for what concerns in using it in the context of presenting *PowerPoint* slideshows, there are two fundamental interactions needed: talking and gesturing.

6.1 Autonomous Life

By default, thanks to the Autonomous Life implemented in its OS, the robot will be able to gesture and express itself while it is in stand-by or performing any action that triggers the robot to interact.[1]

Thanks to that behavior, if *Pepper* is not doing anything it will be randomly interacting if there is people in the room, keeping eye contact with them or blinking its LEDs. As well it will speak with some predefined phrases like *Hello!*, gesturing with its arms.

Autonomous Life is a very good starting point to have a living robot even though it is not being ordered any other action. While doing a presentation there will be moments the robot is stand still, and a much more personal and human way to do it is keeping awareness of its environment, and not limit to stop all its movements.

6.2 Making the robot talk

To make *Pepper* talk there are two possible ways, according to the documentation provided by the manufacturer: the **ALTextToSpeech API** and the **ALAnimatedSpeech API**.

The main difference between these 2 classes are the way they perform the speech. While the ALTextToSpeech method is limited only to make the robot pronounce the strings that are passed to it, the ALAnimatedSpeech goes further, including Autonomous Life behaviors and user defined interactions while speaking.

[1] This is only possible if the Autonomous Life behavior is activated. For some purposes that the robot needs to completely stop this state of awareness can be cancelled.

As speaking during a presentation is not done with plain text, the second method is going the one used.

The way of performing it in *Python* is the following, in Code 6.1.

```
from naoqi import ALProxy #needed to connect to the robot

tts = ALProxy("ALAnimatedSpeech", robot_ip, 9559)
    #creating an instance of  ALAnimatedSpeech
    #robot_ip is the local IP of the robot connected to the
    network
    #9559 is the predefined port to perform the communication

tts.say(textString) #says the string passed to function
```

Code 6.1: Using ALAnimatedSpeech

Going more in detail, with the ALAnimatedSpeech API, the robot will analyze the text string given, and if it finds some keyword (called *Tag*) for which it has any pre-defined gesture, the robot will trigger that animation when speaking. An example of animations can be seen in Table 6.1.

Tag	joyful	sad	floor	there
Animation	Happy_4	Desperate_1	Floor_1	Far_1

Table 6.1: Examples of tags with their corresponding gestures

The name shown in the table corresponds to the internal name of the animation. The full list has more than 100 different Tags[22].

With this way of making *Pepper* speak, actions are not only triggered by those keywords but they can be forced. By including in a string the pseudo-code *^startTag(tag name)* ,substituting *tag name* by the name of the Tag, the robot will automatically trigger the actions linked to it.

6.3 User defined gestures

Taking profit of the forced gestures that the robot can express when it finds the pseudo-code in a string, an extra function will be implemented.

To create even more expressive presentations, and not needing to put all the attention in writing the text including *Tags* that *Pepper* interprets, the

functionality of including Emoticons in the text and make the robot interpret them as triggers to some specific animations is going to be added.

By this method, the user will not need to learn that complex pseudo-code *^start(tag name)*, and will be able to include smilies like **:)**, easy to remember, into the presentation notes, having the same effect.

For the purpose of this study, three forced animations will be added, making use of :) to show happiness, :| to show indifference or doubt and :(to express sadness.

To do this, after extracting the presentation notes with Code 5.2 and before sending its text to Code 6.1, the text string will be conditioned in the following manner (Code 6.2):

```
1
2 swapper = {
3     ":)" : "happy_animation" ,
4     ":(" : "sad_animation" ,
5     ":|" : "doubt_animation"
6     }
7
8 for key in swapper:
9     my_string = my_string.replace(key, "^startTag(" + my_dict[
    key] + ")" ) + " ^waitTag(" + my_dict[key] + ")"
```

Code 6.2: Including user defined gestures

Explanation of Code 6.2

In that code, a dictionary called *swapper* is defined, containing as **Keys** the Emoticons that the user can input, and as **Values** of those keys the animation that they should trigger (note that there the full path to the animation should be typed).

In the loop, the string is checked whether it contains any **Key**, and if so it is substituted by its corresponding pseudo-code, with the **key value** inside it. At the end of the string *^waitTag(tag name)* is added to make the robot wait the animation to finish before continuing with another string.

6.4 Pointing

In order to attract attention in a determined zone of the slide, it is decided to add pointing capabilities to the application. By this way, it will be a more

human behavior and the presentation will be held in a more interactively.

Pepper is not able to freely move the fingers in its hands. They can only be triggered all together opening or closing the hand, but no finger is prepared to move by its own, so the pointing system will not be very precise. For that reason this is going to be implemented by simple trigonometry. The space is going to be described as follows, in Picture 6.1.

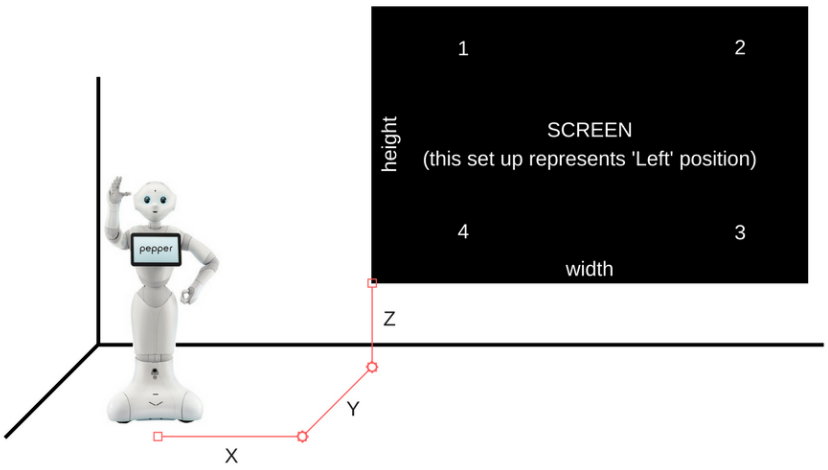

Figure 6.1: The space where the robot does the presentation

To do that, the relative position of the robot with the screen is needed (defined by X, Y, Z and with if the screen is located at its Left or its Right), as well as the screen dimensions. For its the purpose, the screen is divided in 4 quadrants as shown in the previous image. To be able to use the same presentation in different places, the way of inputting the description of the space will be performed in the software, and not in the presentation notes like other behaviors up to now.

The method to enable the presentations maker to make the robot point is going to be the same as in the last section with the user defined gestures, but now instead of adding emoticons to the notes, some pseudo-code will trigger the pointing behavior.

By adding *[Point=x]* in a paragraph (with *x* being the desired zone from 1 to 4), when the application parses the text, with the help of a *Python* dictionary, also as used before, the software will retrieve the joints angles calculated with

the definition of the space, and perform the pointing movement. The basic code to perform that task is the one shown in Code 6.3.

```
def point(position):
            angles_vector = angles[position]
            motion.moveTo(0,0,angles_vector[0])
            motion.setAngles(shoulder, angles_vector[1],0.5)
```

Code 6.3: Pointing

In that function, the program looks for the angles of the joints by searching in the *angles* dictionary the ones with the key passed: *position*, and stores them. This dictionary is previously initialized by computing the values with the trigonometry representing the 3D space described in Figure 6.1. The way of calculating those angles can be found in Annex C.

This vector has 2 values: the first one states the rotation around the Z axis that *Pepper* performs and the second one the angle of the shoulder that should be acquired to point in that determined position.

Then, similarly to what is done in the *TextToSpeech*, 2 functions of the *ALMotionProxy* are called, one to make the robot rotate and the other to move its arm.

6.5 Summary

Thanks to the Autonomous Life behavior, the robot does not need to be programmed in order to perform as if it had life, so it can randomly interact when no instructions are being sent to him.

To make *Pepper* speak, fundamental to do a presentation, the ALAnimatedSpeech API has to be used, in the way that appears in Code 6.1. With that, the robot will say the text strings that are given to him, gesturing and expressing emotions according to the **Tags** there.

As a text sentence may not contain any keyword that triggers actions, the possibility of forcing the robot to perform expressions and gestures is also implemented. Now, if the text contains an emoticon like :) :(or :|, the program will make the robot express such thing.

Finally, the extra functionality of pointing is added. The way of pursuing it is similar than with the forced gestures: the pseudo code *[Point=x]* can be added now to the notes to make the robot point in a determined zone of the screen.

7 Adding the slides

Up to this point the presentation notes have been extracted and the robot is capable of tell that content while expressing and gesturing, but it lacks of the ability to use the visual aid that the slides are.

python-pptx currently has not implemented a procedure to render the XML of a presentation, so it will not be possible to show in the software the slides by means of using only this module. Another complementary method should be used to generate visuals from the presentation. It would have been possible to make *Python* interact locally in the computer with *PowerPoint*, but this would have caused interoperability problems if running the app from different operating systems.

It was decided then to go for a simpler but effective method that will allow it to be used in all the platforms: exporting the presentation as *.jpg* images and manage those files with the application developed. This will carry on drawbacks like not being able to reproduce multimedia content (videos or audio) or animations but will avoid a lot of compatibility problems for a first version of this app.

7.1 Exporting the presentation

Microsoft PowerPoint has as an option exporting a presentation into *.jpg* images.

The method used will be then exporting the presentation into a package containing in a structured way both the .pptx file necessary for *python-pptx* and the corresponding .jpeg files of each slide that the software will use to display in the screen.

A new file extension could have been created for the package containing the presentation and the images that would have added extra security and avoid possible corruption of data, but as this is not the concern of this project, for sake of simplicity, the package is simply going to be a folder containing the necessary items.

To do so a VBA[23] [1] program that interacts with PowerPoint is created, in

[1]VBA (Visual Basic for Applications is the programming language supported by the MS Office suite that lets the developer build custom functionalities that have full access to the host application and many aspects of the Windows API

order to automate the task. The program is created by means of the PowerPoint Visual Basic developer tools.

The code responsible of the core functionality is the one shown at Code 7.1. Note that the variables there SlidesPath and PresentationPath will be generated after user input.

To guide the user, a GUI will also be developed. It will consist on a form showing the guidelines to export the presentation, with two text fields as user input requesting the path and the presentation name. When introduced, a button will run the code creating the folder hierarchy with the .pptx and the .jpg files.

```
1 For Each Sl In ActivePresentation.Slides
2         Sl.Export SlidesPath & "\slide" & Sl.SlideIndex & ".jpg"
    , "JPG"
3     Next Sl
4 ActivePresentation.SaveAs PresentationPath & "\" & Name,
    ppSaveAsOpenXMLPresentation
```

Code 7.1: VBA snippet to export a presentation and its slides as images in a structured way

The final result can be seen at Figure 7.1.

With this user form, the path picking is automatically done through a folder selection dialog. Also a *HELP* button has been added, that shows some tips on how to create a presentation to be *Pepper-friendly*, allowing high interaction by the robot, as well as indicating the possible ways to introduce the user defined gestures explained in Chapter 6.

7.2 Making it easy: PowerPoint add-in

The recently created VBA program can only be accessed though the developer console of the .pptm[2] file where it was created, meaning that for the moment it is a local program, and not globally available in PowerPoint.

To handle this problem, the program will be converted into a PowerPoint complement (.ppam) that is easily distributable and installable amongst any user. To access this add-in, a shortcut will be added to the PowerPoint upper tabs menu (called ribbon menu).

[2] .pptm is a .pptx file with macros or VBA scripts in it

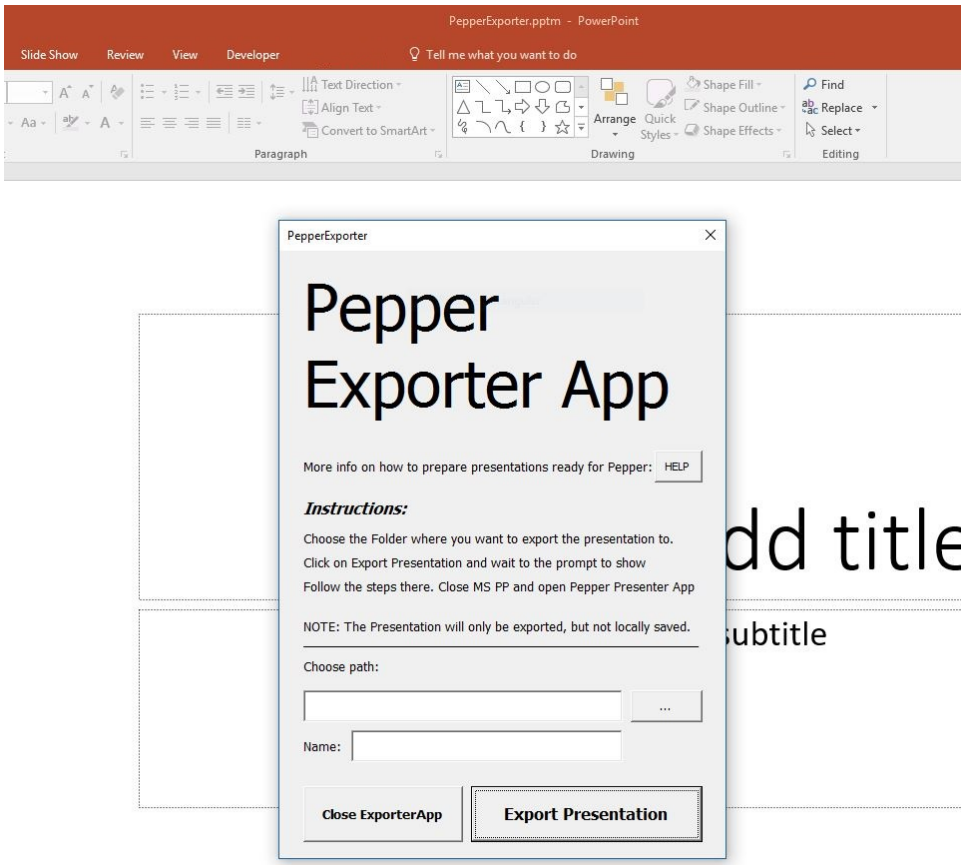

Figure 7.1: The ExporterApp user form.

The documentation needed to understand how PowerPoint complements work is found at the MS Office Developers Network[24].

After adding the necessary VBA scripts to the program, and internally modifying the .pptm file including the new relationships to show the ribbon control in the PowerPoint upper bar, the file is saved as a .ppam (PowerPoint Add-in). The result is shown in Figure 7.2.

The user form stated in Figure 7.1 can be opened from the ribbon control in PowerPoint, making it a much more easy task.

Now, everyone in possession of this app to do presentations will be able to easy install this Add-in through the .ppam file and have the ExporterApp available in their PowerPoint installation.

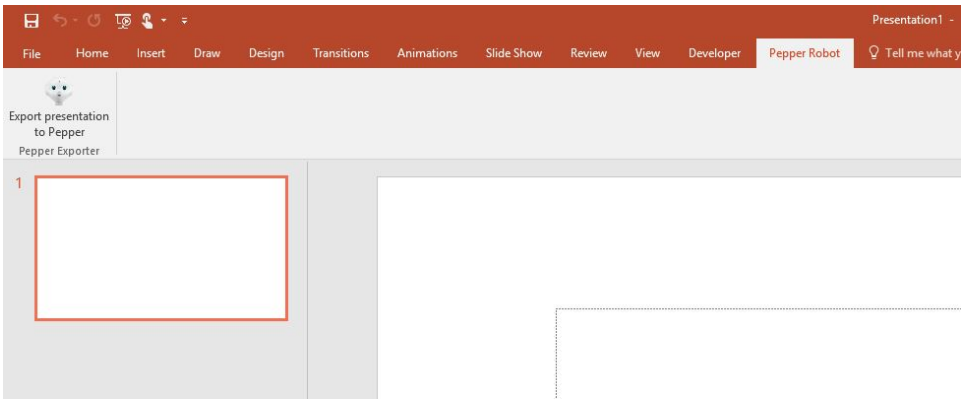

Figure 7.2: The new ribbon control to access the Exporter application

7.3 Summary

The lack of rendering capabilities of *python-pptx* prevents the application to show the presentation using only this tool, so another way has to be found to make *Pepper* present a slideshow showing visual aid.

To do so, it is decided to export the slides in the .pptx file as images in .jpg format, that will be easy to handle by the *Python* program, to show or project on a screen. As a drawback it will not be able to display multimedia content.

The way of doing it is to create a PowerPoint complement, which is easily distributable. The add-in will be in charge of, after getting information by the user on where to export the presentation, create the necessary structure of folder to store there the .pptx file ready for *python-pptx* to open, and the .jpg images used by the core software to present on a display.

To make it more user-friendly, the complement is created with a GUI, and a ribbon control added to the PowerPoint tools bar.

8 User-friendly application

As any other software running in a computer, the easiness of use resides in a clear and intuitive **GUI** (Graphical User Interface).

Up to this point, the described code in the past chapters can run directly from the command line or executing a script, but it has no way to interact with the user letting him/her choose the presentation the robot has to do, or more fundamental things, setting the robot IP address so the API knows where to send the commands.

For that reason, an User Interface is needed, but instead of being lines inputted in the console (faster to develop), it is going to be designed as a graphical one so even the most inexperienced users with programming can use the suite.

8.1 The tool: *htmlPy*

After an exhaustive research on the possible ways to perform this task, the chosen option is *htmlPy* [1] [25], developed by *Amol Mandhane*.

This is another *Python* module that enables to create user interfaces using *HTML*, *CSS* and *javascript*. This programming language has been since long time used to create server side web applications (part of *YouTube* is created in *Python*).

With this module that functionality can be created locally, generating a program that uses HTML and CSS as front-end (the interface) binded with a *Python* written back-end that adds the interactivity and the functions. The update of the front-end is performed through *javascript* called from the *Python* code.

8.2 Front-end and user input

Having in mind the objective of future portability of the software to other platforms, with for sure different screen sizes and resolutions, an adaptable and

[1] htmlPy has as dependency the PySide module, which is not included in the package and should be installed separately

scalable GUI is needed.

For this purpose, and as the GUI is created from HTML and CSS, the responsive framework *Bootstrap3* [2] is going to be used. It has been designed with the aim of creating auto-scalable websites that adapt to any screen size.

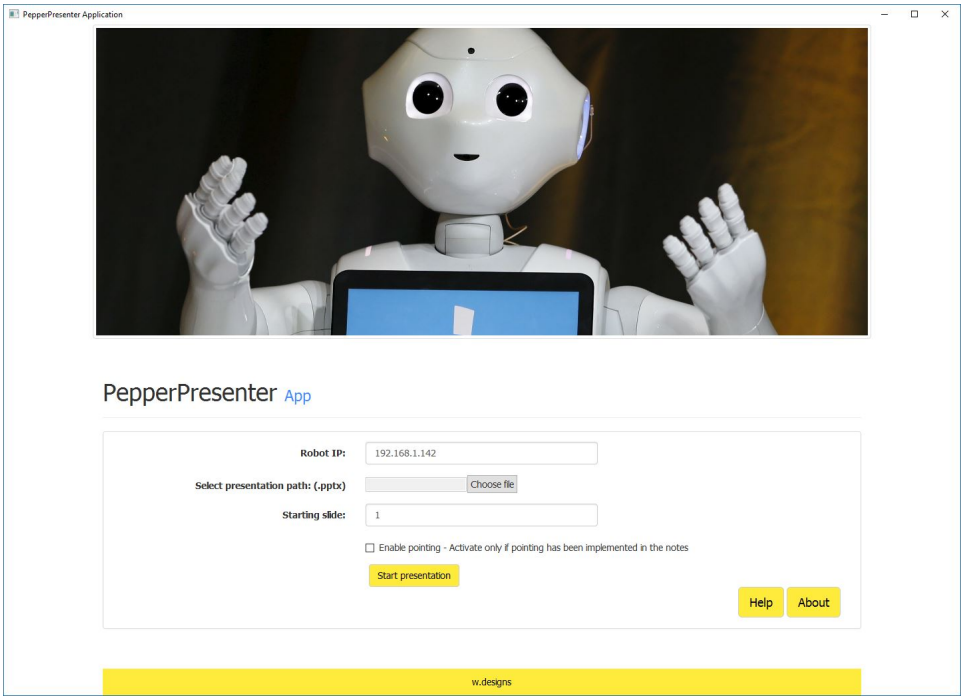

Figure 8.1: The app front end.
With the basic user inputs and buttons to access other sections, like *Help*

As user inputs, those are the ones necessary to have a software that can be used for any situation:

- **Presentation path**. For *python-pptx* to know which file to open and for the software to know where to look the slides to show.
- **Robot IP**. For the robot SDK to know where to connect and send the instructions.
- **Starting slide**. Just introduced, this is not a necessary parameter for the software to run, but it will enable the user to start presentations from a slide different than the first one and will be thus a functionality implemented in the code.

[2] `http://getbootstrap.com/`

- **Pointing**. This check-box will let the program know whether the user is going to perform a presentation allowing the robot to point or not. If it is checked, is shows 6 more inputs needed to calculate the movements (the ones introduced in Figure 6.1. Those new inputs are screen position, screen width, screen height and X,Y and Z coordinates. This extended user input can be shown in Figure 8.2.

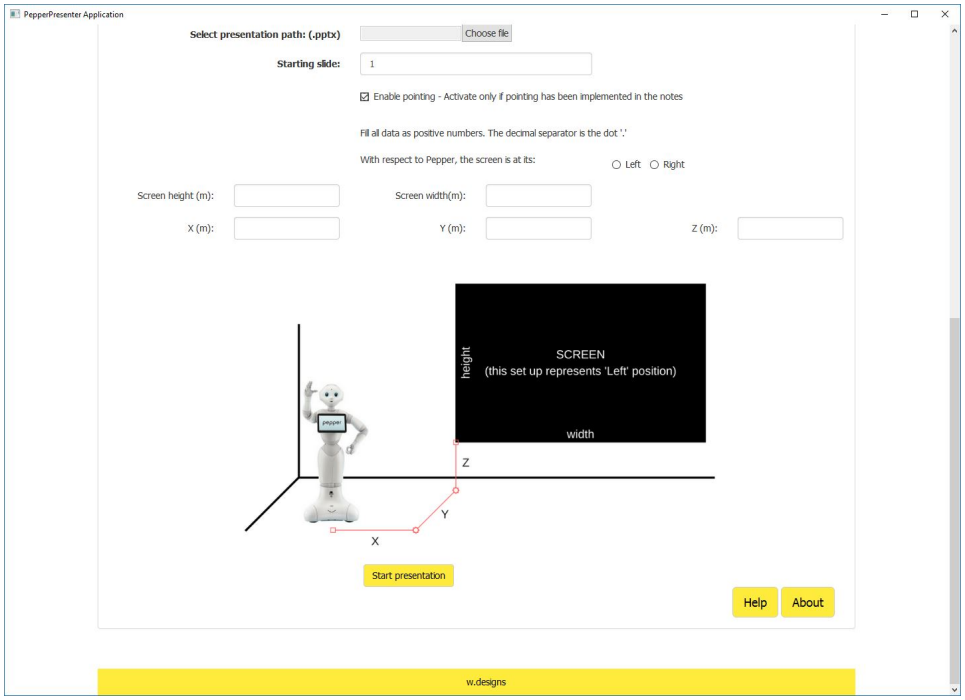

Figure 8.2: The extended user input.
Only accessible if the check box is marked. If not, the user does not need to type that information as the presentation is not intended to need pointing.

Those parameters are inputted by the user in the form, which when submitted sends them to the back-end through a *json* [3] object.

8.3 Back-end

The back-end contains all the functionalities previously described in Chapter 6, and is also in charge of displaying the visual aid that was previously exported

[3] json (javascript object notation) is a data interchange format very used in the web. Is human-readable and easy to parse. It is the format used by htmlPy to share information between the front-end and the back-end.

by means of the PowerPoint add-in proposed in Chapter 7.

The necessary parameters to correctly work are imported from the front-end when the form is submitted, parsing the *json* object into variables. The update of the front-end is performed through *javascript*.

The flow of the application is the following:

1. The form is submitted and the back-end receives the user data.
2. The data is parsed and it is assigned to its corresponding variables. The current slide is assigned the value of the starting slide introduced in the form.
3. Through *javascript* the front-end is updated showing the current slide.
4. The code regarding to make *Pepper* present that slide is executed. The robot does its task, including all the necessary interactions depending in what has been found in the slide notes. Here is where all the functionalities of the presentation are implemented (speaking, gesturing, pointing).
5. When *Pepper* ends presenting that slide, the back-end waits to be triggered by user input (clicking on the screen).
6. If that was the last slide, the front-end is updated closing the slide being shown, returning to the initial screen, and the back-end finishes it execution. If the slide was not the last one, the current slide variable is incremented to the next slide and Step **3** is run again.

8.4 Summary

To make the application more accessible, a **GUI** is created using the *Python* module *htmlPy*. With this interface, the user will be able to use the app in an intuitive way, introducing the necessary data to run the program through an interactive form.

The way how this internally works is quite simple. Once the data is introduced and the user clicks to start the application, this is sent to the *Python* back-end, which runs all the code in charge of the previously stated in other chapters functionalities.

During the presentation, the slides are shown in the screen, so the presenter can use it as an aid, or project them into a bigger display.

9 Validation

To check whether the abilities implemented in the robot are really captioned by the spectators or not, it has been decided to perform a test presentation with 2 slides where the main functionalities are shown. The 15 persons who participated where requested to fill in a test with 3 sections: related to what they would expect, related to what they have seen in the presentations and related to the way they see this technology.

9.1 First section: expectations

The first question of the test is done before seeing the robot presenting. Here, people were requested to choose, amongst different options, the things they would expect from a robot doing a presentation. The results are the following:

Figure 9.1: What do you expect from a robot doing a presentation?

The robot has 3 functionalities of the 5 shown in the question, and those were among the most voted ones, so what people expect of such an application is mostly implemented.

9.2 Second section: presenting

After showing the robot doing the 2 slides presentation where it presents, gestures, express and points, the attendants were requested to answer 2 questions for each slide: one referred to what the robot expressed most and the other to where the robot pointed.

9.2.1 Results from slide 1

In this slide the robot was ordered to point to the Upper Left corner and express joy.

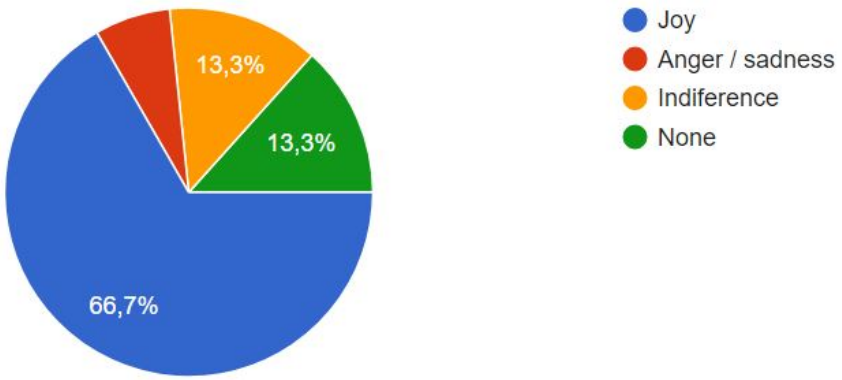

Figure 9.2: What has the robot mostly expressed?

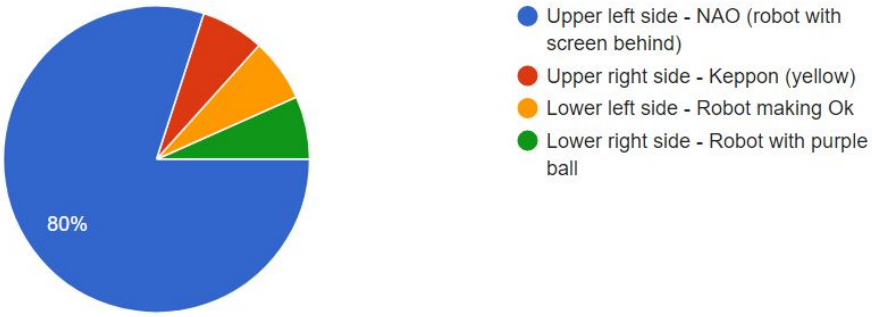

Figure 9.3: Where has the robot pointed?

In this case, people appreciated what the robot was doing.

9.2.2 Results from slide 2

In the second slide the robot was intended to express sadness and point to the Lower Right corner.

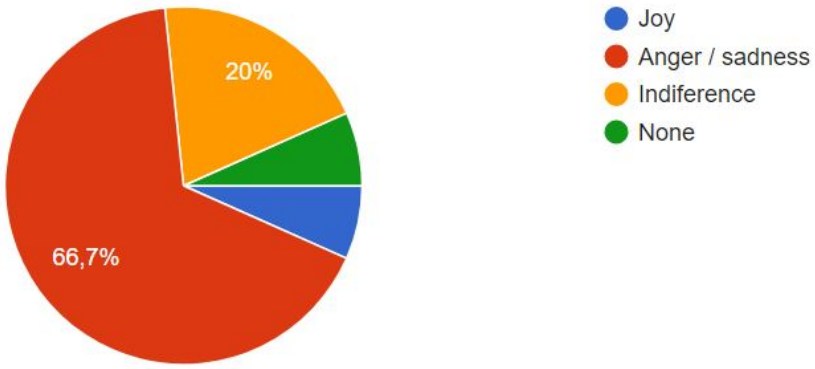

Figure 9.4: What has the robot mostly expressed?

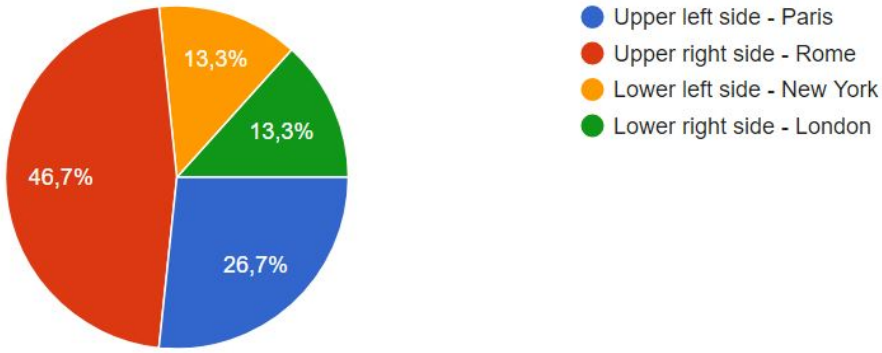

Figure 9.5: Where has the robot pointed?

Here the emotion was mainly appreciated, but the pointing was confusing, people seeing that the robot pointed the right side but in its upper part instead of the lower.

9.3 Third section: conclusions

Two questions where asked: if the technology was going to be applied in the future and if they felt that the robot transmitted all the information.

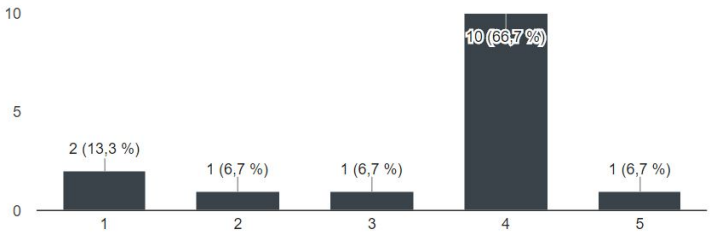

Figure 9.6: In which measure you think this will be applied in the future?
1: not going to be implemented; 5: it will become the standard

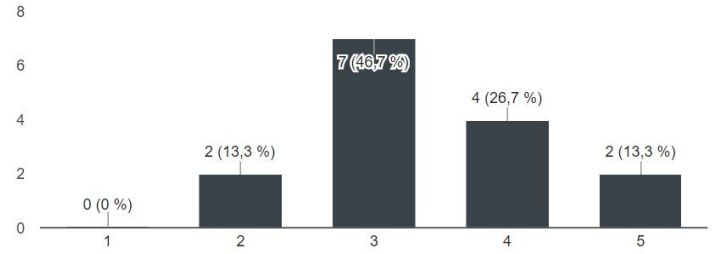

Figure 9.7: How you felt about the robot presenting?
1: does not transmit the information; 5: it has done it like a human would

9.4 Summary

After performing a test where the robot made a presentation and the spectators were requested to fill a questionnaire, the results showed that what has been implemented in the robot is what people expect from it.

The way it performs shows that people appreciates the emotions transmitted by the robot, but becomes a bit more unclear when it concerns to the pointing as leads to some confusion depending where the spectators are watching it from.

As people see it, this technology may have a good adoption in the future, but is still a bit far from performing like a human would.

10 Conclusions

By this point, the application has been completely developed and tested. The final result can be seen, as well as the files can be obtained obtained, in the project website:

`https://wladiarce.github.io/PepperPresenter/`

10.1 Objectives

The objectives initially stated for this project have been successfully accomplished. Along this document, they have been depicted and the application has been developed taking into account and including all of them.

Regarding making the robot point, it has been a challenging functionality but finally, and despite the not so good pointing capabilities of *Pepper* as it cannot use a single finger, the results are more than acceptable. Now, the robot can designate different zones in the screen where it is presenting the slideshow.

Complementing the functionality of user-defined emotions in the sentences, it has also been included that the robot fades its eyes in different colors depending what emotion it is expressing. By this way, the spectators will be able to perceive with more confidence what *Pepper* is transmitting.

10.2 Using it in a real environment

After testing and seeing all the functionalities that it has, it can be said that the application is ready for deployment and to use it in professional environments. Thanks to its easy-to-use user interface there is almost no knowledge required to run the software.

Even though all the functionalities that it has, mainly the kinematic ones, the robot is safe to use in closed environments. All the movements it performs are inside its kinesphere. Including for the pointing gestures, *Pepper* does not move around the room, simply turns and points. This makes this application suitable to be executed in small conference rooms or places where there is no more space than the necessary for the robot to stand.

10.3 Further development

Having the application committed its tasks, many improvements and new functionalities can be added in future versions of it.

10.3.1 Installation package

The app, as can be seen in Annex A, is not difficult to install, but a series of steps are needed that can lead to dependencies or compatibility errors.

The good approach to solve this problem will be to create a executable package containing all the dependencies that installs them automatically, without needing to go one by one.

10.3.2 Updating the app

The first thing that can be easily updated is adding more custom gestures for the user to run directly from the notes.

Now the application only has been developed including 3 gesturing examples, to prove the functionality, but *Pepper* has many other predefined actions that can be also added.

As well, in a next version a new method to display the visual aid can be implemented, allowing to use not only images but multimedia content also.

10.3.3 New state of the art functionalities

Up to now, the way of performing the presentations is not fully autonomous: despite *Pepper* has some ability on interacting by itself, it has to be told everything that has to say.

The next step in the development will be then adding such a capability: that the robot thinks by itself. With the current state of the art in *Artificial Intelligence* it would not be that hard let the robot present for example, slides containing pictures, making autonomously a description of them without needing the user to input that text in the notes.

10.3.4 More operation environments

The way it has been implemented allows the robot to perform presentations showing the visual aid with the help of an external display. This makes the application suitable for scenarios like conference rooms or classrooms, and they can be defined as static ones.

One future implementation can be to make the use of the robot touchscreen, that even it is smaller, will enable it to do presentations in new environments. The app then will not be very effective for big spaces as the ones firstly described, but when displaying in the robot screen the robot will be able to move around while keeping the spectators informed, making it suitable to for example, make tours to small groups in a museum.

Annexes and Bibliography

A How to install

The application to control the robot is cross platform, meaning that it can be installed on *Windows*, *Linux* and *OSX*.

Despite of that, *PowerPoint* and its add-in are only available for *Windows* and *OSX*, the only platforms *Microsoft* distributes the software to. To do a presentation from a *Linux* machine the .pptx file and its corresponding dependencies would have to be previously exported from a compatible platform.

A.1 Installation instructions

A.1.1 Install the Presenter Application

1. Download and install *Python 2.7 - 32 bits*[1] from the *Python* website[17]. Some distributions of *Linux* and *MAC OS* already have this version preinstalled.

2. Install the naoqi SDK for your platform, available in the *Alebaran Community Software* site.

   `https://community.aldebaran.com/en/resources/software`

3. Install the corresponding *Python* modules that are dependencies of the application: *PythonPPTX*[21], *lxml*, *htmlPy*[25], *PySide*, *PyQt4*. It can be done by opening the command line and typing the following command:

```
1 pip install python-pptx htmlPy PySide lxml
```

Code A.1: Installing the app. Python dependencies

4. Download the latest version of the app from the project website:

   `https://wladiarce.github.io/PepperPresenter`

5. Run the app by opening *main.py* and let *Pepper* do the presentations.

[1] Other versions are not compatible with the naoqi SDK

A.1.2 Install the *PowerPoint* add-in

1. Download the latest version of the add-in (*.ppam*) from the project website:

   `https://wladiarce.github.io/PepperPresenter/`

2. In *MS PowerPoint* go to *File*, *Options*, *Add-ins*.

3. In the lower part, click on *Manage* and select *PowerPoint Add-ins*(Figure A.1). Click *Go...* A new window will open.

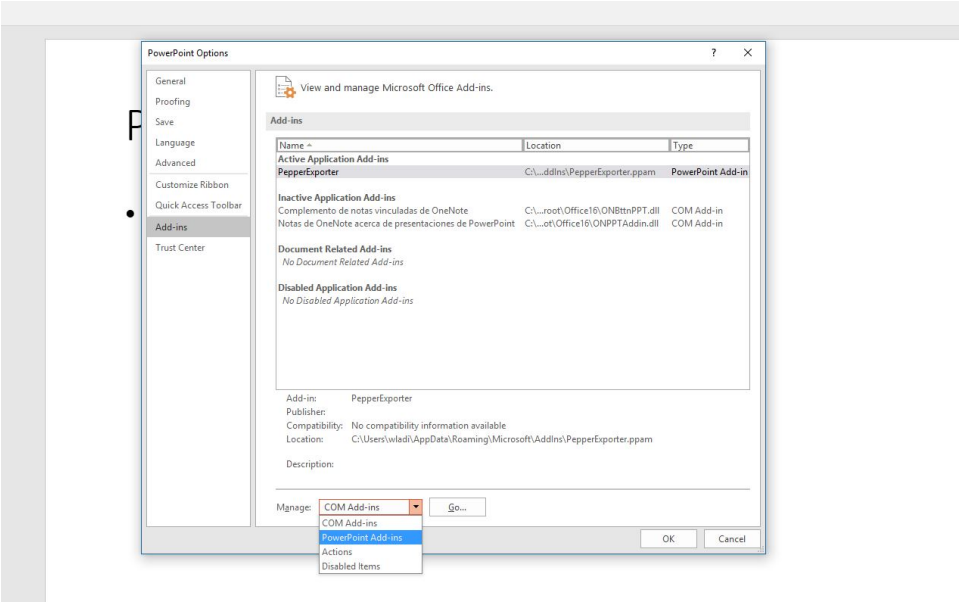

Figure A.1: PowerPoint Add-ins options

4. Click on *Add New...*, select the downloaded file from the dialog, and check that the add-in is loaded. If not, click on *Load*.

5. The new complement should appear now in the *PowerPoint* ribbon controls.

B How to annotate a presentation

With the current development of this application, the robot will read the text that is written in the notes and according to that interact and present the slideshow.

The basic way of making *Pepper* present is with plain text were a story is told but, as described in this document, more functions can be used including annotations in the text and they are the following:

- User defined emotions - triggered by emoticons in the text. Actually three possibilities exist.

:)	:(	:|
happy	sad	doubt

  Those make the robot not only to perform a gesture, but also to change its eyes color.

- Pointing - triggered by the following code found in the text `[point=x]`, where x is the selected zone to point in the screen, from 1 to 4, according to the following scheme.

1	2
4	3

B.1 Advice to create good presentations for *Pepper*

The notes are read and parsed paragraph by paragraph with a small pause between them, so a good procedure is not to create long sentences, neither use the same annotation twice in the same phrase.

As so, the best is to create short phrases where pointing or custom emotions are expressed, and if a bigger use of this functionalities is needed, to keep it in separate lines in the notes, avoiding possible strange behaviors or overlapping of movements.

C Joint angles calculations for pointing

The values that the user inputs, if pointing is enabled, are referred according to the spatial coordinate system presented in Figure 6.1, and are the following:

- X, Y, Z for the relative position of the screen with respect to the robot.
- Width and height of the screen.

To compute this the angles procedure is trigonometric calculations, referred to the 4 available positions in the screen (upper/lower, near/far from the robot[1]).

Defining the following variables:

$$near = X + \frac{width}{4} \qquad far = X + \frac{3*width}{4}$$

$$lower = Z + \frac{height}{4} - 0.91 \quad upper = Z + \frac{3*height}{4} - 0.91$$

The angles are calculated as, for the turning angles:

$$\Theta_{near} = \arctan\left(\frac{Y}{near}\right) + \frac{\pi}{2} \quad \Theta_{far} = \arctan\left(\frac{Y}{far}\right) + \frac{\pi}{2}$$

and for the shoulder joint angles:

$$\alpha_{upperNear} = \arctan\left(\frac{upper}{\sqrt[2]{near^2+Y^2}}\right) \quad \alpha_{upperFar} = \arctan\left(\frac{upper}{\sqrt[2]{far^2+Y^2}}\right)$$

$$\alpha_{lowerNear} = \arctan\left(\frac{lower}{\sqrt[2]{near^2+Y^2}}\right) \quad \alpha_{lowerFar} = \arctan\left(\frac{lower}{\sqrt[2]{far^2+Y^2}}\right)$$

With those angles, and the program knowing with one corresponds to each screen position according to where the robot is, the commands can be sent to the robot when pointing is necessary.

[1] The near/far from the robot is determined in function of whether the screen is at its left or right

Bibliography

[1] Jannette Collins. Education techniques for lifelong learning. *RadioGraphics*, 24(4):1185–1192, 2004. PMID: 15256638.

[2] Lucinda Becker Joan van Emden. *Presentation skills for students*. Palgrave Macmillan, 2016.

[3] Jamy Li. The benefit of being physically present: A survey of experimental works comparing copresent robots, telepresent robots and virtual agents. *International Journal of Human-Computer Studies*, 77:23 – 37, 2015.

[4] Thomas B. Sheridan. Human–robot interaction. *Human Factors*, 58(4):525–532, 2016.

[5] C. Bartneck and J. Forlizzi. A design-centred framework for social human-robot interaction. 2004.

[6] Rachel Kirby, Jodi Forlizzi, and Reid Simmons. Affective social robots. *Robotics and Autonomous Systems*, 58(3):322 – 332, 2010. Towards Autonomous Robotic Systems 2009: Intelligent, Autonomous Robotics in the {UK}.

[7] Juhana Jauhiainen Sakari Pieska, Mika Luimula and Van Spiz. Social service robots in wellness and restaurant applications. *Journal of communication and computer*.

[8] Eduardo Zalama, Jaime Gómez García-Bermejo, Samuel Marcos, Salvador Domínguez, Raúl Feliz, Roberto Pinillos, and Joaquín López. Sacarino, a service robot in a hotel environment. pages 3–14, 2014.

[9] Masahiro Shiomi, Takayuki Kanda, Hiroshi Ishiguro, and Norihiro Hagita. Interactive humanoid robots for a science museum. 2006.

[10] Emotion Robotics. Social robotics and home automation. 2015.

[11] Elaine Mordoch, Angela Osterreicher, Lorna Guse, Kerstin Roger, and Genevieve Thompson. Use of social commitment robots in the care of elderly people with dementia: A literature review. *Maturitas*, 74(1):14 – 20, 2013.

[12] Marcel Heerink, Bram Vanderborght, Joost Broekens, and Jordi Albó-Canals. New friends: Social robots in therapy and education. *International Journal of Social Robotics*, 8(4):443–444, 2016.

[13] Lai Poh Emily Toh, Albert Causo, Pei-Wen Tzuo, I-Ming Chen, and Song Huat Yeo. A review on the use of robots in education and young children. 19, 2016.

[14] Generation Robots. Nao presenter application. https://www.generationrobots.com/en/401787-nao-presenter-application-nao-robot.html.

[15] Amy Cuddy for TED. Your body language shapes who you are. https://www.ted.com/talks/amy_cuddy_your_body_language_shapes_who_you_are.

[16] Nancy Duarte for TED. The secret structure of great talks. https://www.ted.com/talks/nancy_duarte_the_secret_structure_of_great_talks.

[17] Python Software Foundation. Python. https://www.python.org/.

[18] OOXML. Open office xml. officeopenxml.com/.

[19] Python Software Foundation. Python xml libraries. https://docs.python.org/2/library/xml.etree.elementtree.html.

[20] Python Software Foundation. Python package index. https://pypi.python.org/pypi.

[21] Steve Canny. *python-pptx Documentation*. https://media.readthedocs.org/pdf/python-pptx/latest/python-pptx.pdf.

[22] Aldebaran Robotics. List of tags. http://doc.aldebaran.com/2-4/naoqi/motion/alanimationplayer-advanced.html.

[23] Microsoft. *Microsoft VBA Language Reference*. https://msdn.microsoft.com/en-us/library/office/gg264383.aspx.

[24] Microsoft. Complement creation in powerpoint. https://msdn.microsoft.com/en-us/library/office/gg597509(v=office.14).aspx.

[25] Amol Mandhane. *htmlPy Documentation*. http://htmlpy.readthedocs.io/en/master/.