Juho Salli

DEVELOPING AN APPLICATION FOR THE HUMANOID ROBOT PEPPER

Degree Programme in Information Technology
2019

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

Developing an application for the humanoid robot Pepper

Salli, Juho
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Information Technology
October 2019
Number of pages: 24
Appendices: 0

Keywords: robotics, humanoid robot, software development, Pepper

---

This thesis will explain the process of developing an application for the humanoid robot Pepper. The app that was developed is a proof-of-concept type of application for researching the human-robot interaction especially with elder people with memory disfunctions.

The technology stack chosen for the building this app consists of web-development tools, such as VueJS, Babel, and JavaScript. The thesis will describe how these are used in unison in this process.

I will also describe some of the problems encountered while developing the app and provide possible solutions for the issues. Hopefully the solutions will help future developers.

TABLE OF CONTENTS

1 INTRODUCTION

One of the first humanoid robots, dubbed Eric, was exhibited at the annual exhibition of the Model Engineers Society in 1928. The robots' frame was aluminium and it had eleven electromagnets and one motor powered by a twelve-volt power source. It could stand up from a sitting position, bow and move it's head to left and right. (Refell Family History, 2019).

Ever since the release of Eric the development of humanoid robots has been on-going. The progress of the capabilities of the robots in the last almost a century has been great. The robots are able to move in multiple degrees of freedom (Cybernetic Zoo, 2011) , they've learned to understand voice commands (Wikipedia contributors, 2019) and, some of them remember all the conversations it has had (Wikipedia contributors, 2019).

As the population grows older (Statistics Finland, 2015), even more people move into nursing homes for the elderly. At the same time the funds are cut down, the staff of these retirement homes are spread thin. To help the staff perform their daily tasks, some nursing homes are adapting modern technologies in their infrastructure.

One way of utilizing new technologies is to use social robots to keep the seniors company while the staff attends other tasks. For example, the elderly can play games with the robots or read news with them or just have a casual conversation. In cases where the senior person has some form of memory disease the robot is at its best; it can answer the same questions over and over again without showing any signs of frustration.

It must be mentioned that in no way are these robots supposed to replace authentic human contact. They are there merely to assist humans to perform their work more efficiently by reducing the work load. It's also safe to say that robots cost less than humans, since they usually require one-off purchase expense.

I developed a proof-of-concept application for a local nursing home. The robot that this app is developed for is a humanoid-robot Pepper. The application uses face recognition to greet the person by name and it also features voice control to provide auditory communication with the robot. The user can play games or read news with the robot.

This thesis describes the process of developing the application as well as provides some solutions to the problems that I encountered while building it.

2 PEPPER THE HUMANOID ROBOT

Pepper is a humanoid robot developed by Aldebaran. It was released in 2015. The robot is 120 centimeters tall, it has two high definition cameras and one 3D camera embedded. The cameras are used to identify individual faces, different objects and emotions on the faces around it (Reese, 2016). Pepper also has four microphones placed in its head (SoftBank Robotics Europe, 2018). The robot uses microphones not only to listen to people, but also to detect the direction of which the sound comes from.

The primary uses for Pepper are thought to be in retail used as a customer representative and in public spaces to guide people in the right direction.

2.1 The machine vision library in Pepper

Pepper has a built-in vision solution provided by Omron (SoftBank Robotics Europe, 2018). The library tries to detect faces in the view of Peppers' cameras. In my tests, given decent lighting it performs fairly well.

In addition to face detection the vision solution tries to recognize the face it sees. To succeed in the recognition the face must first be taught to the robot. Up to ten images can be stored per face. Any new image that get added will replace the oldest one (SoftBank Robotics Europe, 2018).

Given the proof-of-concept type nature of the application the face recognition works well enough. Sometimes it mislabels a face and the developer must know how to handle these kind of mislabeling. I will address this and other problems in the machine vision library more in chapter 3.

2.2 Speech recognition in Pepper

Pepper has a built-in speech recognition library open for developers to use (SoftBank Robotics Europe, 2018). The library can be set to detect specific words or certain phrases.

The big issue with this solution is that it can only react to words that have been specified beforehand. This leads to some problems just to get a reaction from a user. For example, there are multiple words in Finnish language that can be used in place of "yes" to reply to a question.

2.3 Tablet for interaction

In Pepper's chest there's a tablet. It has a screen resolution of 1280 by 800 pixels with a capacitive 5-point multi-touch panel (SoftBank Robotics Europe, 2018). All graphical user interface (GUI) work is done using common web technologies such as HTML, JavaScript and CSS.

There's no documentation available about the specifications of the software it's running or the operating system, but it is some custom software built on top of the Android operating system.

This leads to the problem of being dependent on the updates provided by the manufacturer. The web view of the software can't handle modern JavaScript features which leads to the conclusion that the tablet hasn't had any updates in quite some time. I'll address this problem later on.

3 THE APPLICATION

The application developed can be thought to have two states: the "no user"-state in which there is no known person using the app and the "has user"-state where the robot has identified a person and that person is using the application.

In the "no user"-state the robot just waits until it recognizes a person it thinks it knows. It then prompts the user to confirm the recognition. The person can either answer by speaking or by clicking a button in the tablet view. In the case of faulty identification, the robot just returns to the "no user"-state and tries again.

After a correct recognition the robot greets the person by name and asks what she/he wants to do. The questioning is performed by displaying the text in the tablet and by the robot speaking Finnish. This multisensory approach serves both those with hearing impairment and bad eye sight. At this point, the robot is in what can be called "has user"-state.

For this proof-of-concept three actions were chosen from which the user can select what to do: play Tic-Tac-Toe, play memory game or read news. Again, the person can pick one by either clicking the corresponding button on the tablet or speaking out loud the name of the action.

In the game Tic-Tac-Toe, the robot doesn’t try to do anything smart in the selection of its next move. It just picks a random free slot and places its tick there. It takes some effort to lose the game to the robot. This kind of behavior was chosen because the test group consists mostly of elderly people with memory diseases. If this proves to be too boring or otherwise "not good" solution, the algorithm for selecting the next move can easily be improved.

After the chosen task is complete the robot asks if the users wishes to repeat the task or select a new one.

The memory game is a simple 4 by 4 grid of pairs of numbers. At the beginning of the game, the numbers are shown for a few seconds before they are hidden. In the end, when user has successfully found all pairs, the robot gives the user positive feedback and prompts him/her for a new game.

If the user selects to read news, he/she is taken to a view in the tablet that lets the person select the topic of which he/she would like to read about. Currently the user can pick the topic from a few preselected options. After the topic is chosen the robot fetches the top 20 articles and displays the headlines as a list on the tablet screen. The user can then click one of the headlines and the robot will display the text and read the article aloud for the user.

3.1 Software architecture

The architecture of the software is fairly simple; the app consists of the user interface (UI) part and the backend service.

The UI consists of the view in the tablet and of the voice communication between the user and the robot. In addition to registering the app itself in the robot, the backend service handles the registering of the words that needs to be recognized.

The choice for the architecture came from purely practical reasons. Some of the network operations can’t be done in the browser (in the tablet) and the registering of the words that need to be recognized has to be done in the backend service.

3.2 Backend service

The manufacturer provides a few software development kits (SDKs) for developing services to be run on the robot itself, such as one for the C++ -language and one for the Python programming language. There’s also few SDKs that don’t allow you to run the code on the robot, but allows you to control it remotely (SoftBank Robotics Europe, 2018).

The SDK and the language chosen for this project was Python. The choice was based on the fact that I’m more confident in writing Python than for example C++. Python is also fairly high-level language and is possibly somewhat easier language to adopt for the future developers of this project.

The service is responsible of a few things: registering the words that are used for communicating with Pepper and providing network and text parsing operations for the news reading part of the software.

3.2.1 Registering the words

Pepper doesn’t have speech recognition per se. It can only react to the words or phrases that are provided for it beforehand. Therefore, it is more about spotting a word or a phrase and less about actually recognizing speech.

The word vocabulary is set when the service is initialized, using the setVocabulary-method of the ALSpeechRecognition-module (SoftBank Robotics Europe, 2018). The words to be recognized are given as a list of strings to the method.

As there are many words in Finnish language for just indicating a positive response (the same holds true in English too; "yes", "sure", "okay", and "absolutely" can all be used in the same context), the decision was made to keep the list of words short and to only accept few of the most common ones. The same decision was made for the negative responses as well. The reason for these decisions comes from the fact that the developer has to figure out the responses for each word individually.

3.2.2 Networking and news parsing

The news reader fetches the titles for a given topic from an application programming interface (API) of a Finnish newspaper and this can be done in the UI (in the tablet). However, the API doesn’t provide an endpoint for querying the articles themselves.
A solution for getting the article texts is to request the article as a regular HTTP-request using GET-method and parse the content from the response. The parsing can be done

quite reliably as the content is formatted with certain structure. While this solution can be implemented in browser code, a more efficient (and easier) way to do it is to do it in Python.

The UI calls a specific function in the backend service with the URL of the article as a parameter. The function then makes a HTTP-request to the given URL, parses the content using the HTML class-attributes and returns the article text to the UI.

The drawback of this solution is the fact that it is depending on the format (or the layout) of the HTML. If the site structure or class names change, then this solution breaks. As this is just a proof-of-concept this kind of dependency is acceptable.
A more robust solution would be to work with some news-company and have access to their API and request the text from the database.

3.3 The User Interface

It’s only possible to develop web pages for the tablet in Pepper. This means one can only use HTML, JavaScript and CSS. Given this limitation, I chose to build the UI with VueJS-framework (You, 2019). Vue is used to build modern single page applications that can scale from simple interface (as is the case in this application) to production quality apps serving millions of users. Any interested reader is advised to find out more about Vue as I won’t go into too much detail about it.

Softbank Robotics offers two different JavaScript libraries for the developers to use. The first one is the official SDK (SoftBank Robotics Europe, 2018) and the second one is a library built on top of the official one to provide more high-level functions, called Studio Tool Kit (Softbank Robotics Europe SAS, 2015). The Studio Tool Kit was chosen to be used in this project for the simplicity of calling native Pepper functionalities.

The user interface is responsible for most of the logic in the application. For example, it handles registering the face detection event and all the logic behind the actions after a known face is recognized. It also takes care of the needed actions after a known word is heard.

The visual aspect of the UI was kept fairly simple to not to overwhelm to the elderly. The buttons were made big for the ease of reading. Also the number of buttons in a single view was kept as small as possible.

Users can trigger an action by either pressing a button or speaking the word that’s written in it. For example, one could click the "News"-button or speak out loud "I want to read news" and the robot will then trigger the corresponding part of the application.

The word spotting works reasonably well even for the Finnish language. However, it thinks it detects words even when no one isn’t speaking. In the case of this app this didn’t cause any issues as the application subscribes to a small set of words. For some reason, silence triggers the “detection” for the word “koira” (“dog” in Finnish) quite often.

4 PROBLEMS

In this chapter I’ll address some of the issues I came across while developing the application. I’ll also provide my suggestions as possible solutions to fix these short comings.

4.1 Documentation

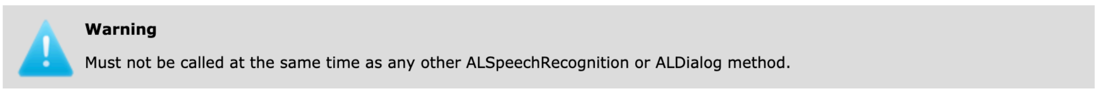
The official documentation has a lot of pages and a lot of content and there has definitely been some effort to provide help for the developers. However, I must consider the documentation as one of the problems.

As the documentation is usually the first place where developers look when starting to get to know new technologies, it should be easy to navigate and the language should be valid. This is not the case with the official Pepper documentation.

The documentation has a whole bunch of typos and the English sometimes looks like it has been translated by some tool and the sentences don’t make a lot of sense. This makes understanding some of the content somewhat hard as one needs to “debug” the text for it to make some sense.

For example, in the ALSpeechRecognition-page (SoftBank Robotics Europe, 2018) there’s a line “Once started, ALSpeechRecognition places in the key SpeechDetected, a boolean that specifies if a speaker is currently heard or not.”. It takes a few readovers to understand that if the robot has heard someone speaking, it will insert a boolean value to the SpeechDetected-key.

In the ALSpeechRecognition API documentation page, there’s a following warning for the setLanguage-method: “Must not be called at the same time as any other ALSpeechRecognition or ALDialog method.” (SoftBank Robotics Europe, 2018). There is no reasoning given for this nor any example for the meaning of “called at the same time” (Figure 1). I had no problems with calling this method in the midst of other ALSpeechRecognition methods in my code.



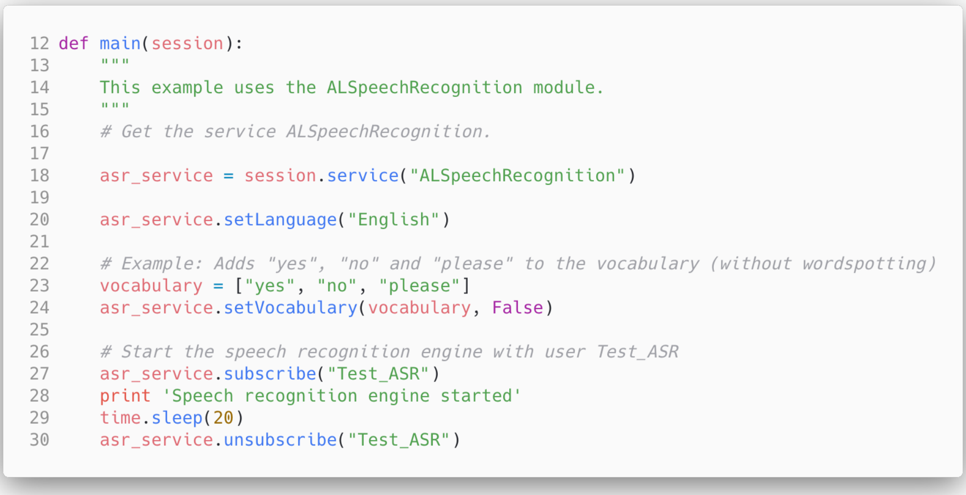
Warning

Must not be called at the same time as any other ALSpeechRecognition or ALDialog method.

Figure 1. The warning message in the documentation.

There is also at least one case of example code that causes an error when executing it.

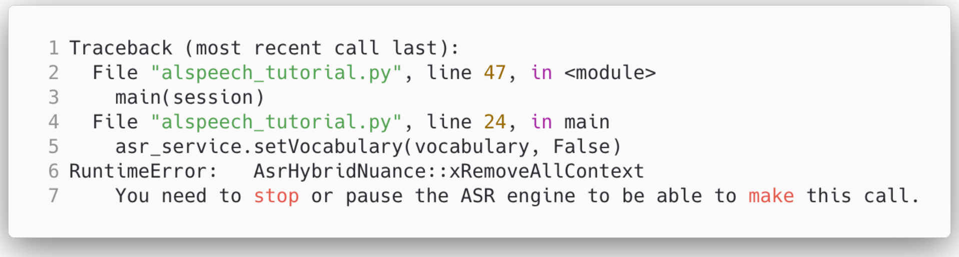
Executing the ALSpeechRecognition tutorial (SoftBank Robotics Europe, 2018) code without any modifications throws an error. The RunTime error happens on line 24 of the code when the setVocabulary-method gets called (Figure 2).



```
12 def main(session):
13     """
14     This example uses the ALSpeechRecognition module.
15     """
16     # Get the service ALSpeechRecognition.
17
18     asr_service = session.service("ALSpeechRecognition")
19
20     asr_service.setLanguage("English")
21
22     # Example: Adds "yes", "no" and "please" to the vocabulary (without wordspotting)
23     vocabulary = ["yes", "no", "please"]
24     asr_service.setVocabulary(vocabulary, False)
25
26     # Start the speech recognition engine with user Test_ASR
27     asr_service.subscribe("Test_ASR")
28     print 'Speech recognition engine started'
29     time.sleep(20)
30     asr_service.unsubscribe("Test_ASR")
```

Figure 2. The tutorial code that throws an error. Boilerplate code left out.

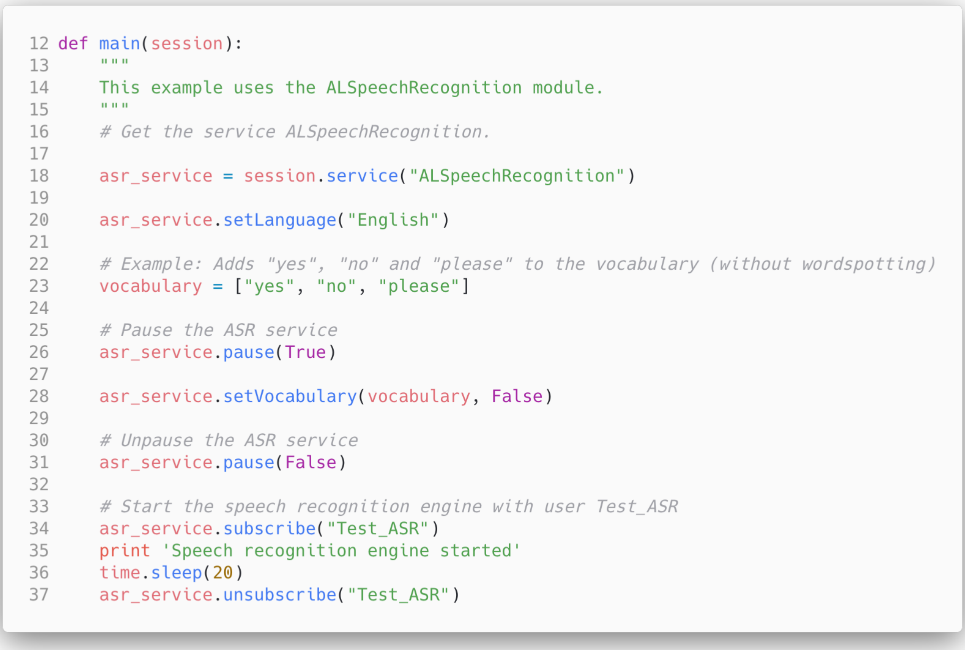
The error message (Figure 3) clearly states that the ASR (ALSpeechRecognition) engine needs to be stopped or paused before calling the setVocabulary-method.



```
Traceback (most recent call last):
  File "alspeech_tutorial.py", line 47, in <module>
    main(session)
  File "alspeech_tutorial.py", line 24, in main
    asr_service.setVocabulary(vocabulary, False)
RuntimeError:   AsrHybridNuance::xRemoveAllContext
    You need to stop or pause the ASR engine to be able to make this call.
```

Figure 3 The error message.

The bug is simple to fix by wrapping the erroneous code between asr_service.pause(True) and asr_service.pause(False) method calls (Figure 4).



```
12 def main(session):
13     """
14     This example uses the ALSpeechRecognition module.
15     """
16     # Get the service ALSpeechRecognition.
17 
18     asr_service = session.service("ALSpeechRecognition")
19 
20     asr_service.setLanguage("English")
21 
22     # Example: Adds "yes", "no" and "please" to the vocabulary (without wordspotting)
23     vocabulary = ["yes", "no", "please"]
24 
25     # Pause the ASR service
26     asr_service.pause(True)
27 
28     asr_service.setVocabulary(vocabulary, False)
29 
30     # Unpause the ASR service
31     asr_service.pause(False)
32 
33     # Start the speech recognition engine with user Test_ASR
34     asr_service.subscribe("Test_ASR")
35     print 'Speech recognition engine started'
36     time.sleep(20)
37     asr_service.unsubscribe("Test_ASR")
```

Figure 4. By adding the lines 26 and 31 no error gets thrown.

4.2 Debugging JavaScript

While developing the user interface, I use my laptops' internet browser for displaying the web page. Using an up-to-date browser let's you use the latest features of JavaScript, HTML and CSS. Debugging while developing is simple when you are able to use developer tools.

Debugging problems on the robot is a bit harder as there is no way to attach to the browser session remotely. As mentioned before, there is no documentation available about the software of the robot. This includes the browser as well.

The source code itself is written using the ES6-standard features of the JavaScript programming language (Ecma International, 2015). For production build (the one deployed to the robot) the code is transpiled into the older ES5-standard version (Ecma International, 2011) of the language by using BabelJS tool (Babel team, 2019).

Transpiling is the process of outputting source code to either a different version of the same language or to a different language (Wikipedia contributors, 2019). The transpiling is done because all browsers don’t yet support the features introduced in the new standard. Most of the code written worked well on the tablet, but there were some cases where some of the scripts weren’t loaded and the page was mostly blank. This led to the need of debugging the code run in the tablet.

The JavaScript code utilizes Array.prototype.includes() -function and the browser in Pepper doesn’t support it. Therefore it needed to be added in by polyfilling the function. A polyfill is a piece of code that implements a feature on web browsers that do not support the feature (Wikipedia contributors, 2019). While the includes()-function is a new feature in new standard, tools like BabelJS won’t add it since transpiling doesn’t add new properties to primitives and they need to be polyfilled by the developer (McGinnis, 2017).

For the not-being-able-to-debug -problem I found a solution to override the console.log()- function. Instead of the normal behavior of writing the given arguments in to the console, the function will now write all arguments to the page itself. The function just simply appends the arguments to the innerHTML-property of a specified div-element.

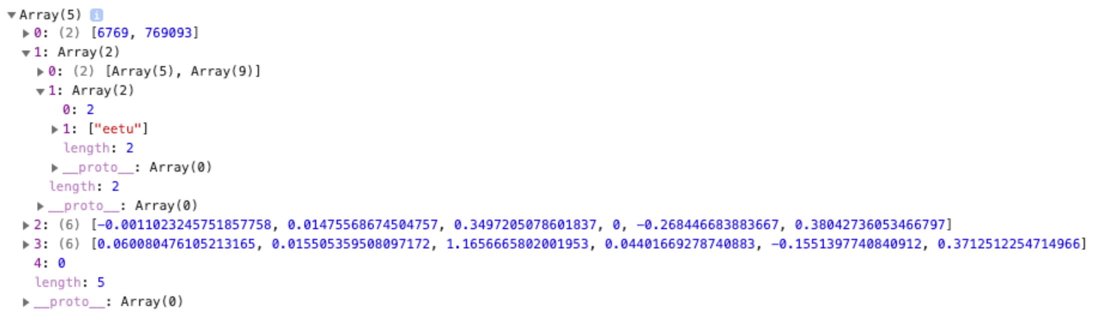
With this solution I could insert console.log()-calls to different places in the code and verify what part of the code got executed and what didn’t. While this isn’t the most elegant way to debug an application, it helped me to move forward in the development.

4.3 Labeling recognized faces

When teaching a face to the robot, one must attach a label for the face (for example the name of the person). The teaching process is simple and easy to use; you just call a single function with one parameter (the name of the person).

The documentation about the structure of the value returned by the FaceDetected event is misleading or is missing information. There are four different possible values listed for one part (named Time_Filtered_Reco_Info) of the multi-dimensional array the return value can be (SoftBank Robotics Europe, 2018).

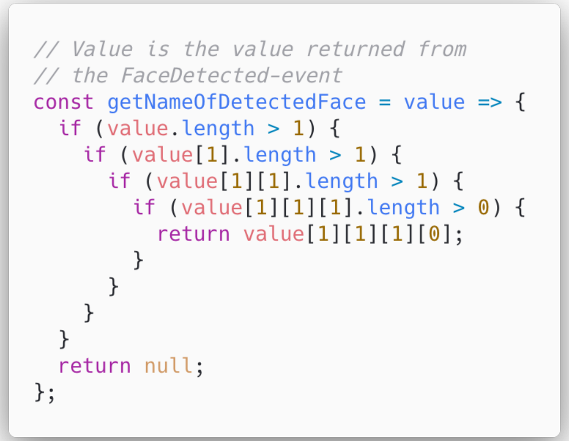
First possible value is that the Time_Filtered_Reco_Info is an empty array if there is nothing new. The second option is an array that has an integer with value 2 as the first element and the second element being an array containing the label of the detected face if there is only one face recognized. The third possible value is an array with an integer value 3 being the first element and an array of multiple labels if multiple faces are recognized. The fourth option listed in the documentation is an array containing a single element; an integer with the value four if a face has been detected for more than 8 seconds but not recognized.



```
▼Array(5)
  ▶0: (2) [6769, 769093]
  ▼1: Array(2)
    ▶0: (2) [Array(5), Array(9)]
    ▼1: Array(2)
        0: 2
      ▶1: ["eetu"]
        length: 2
      ▶__proto__: Array(0)
      length: 2
    ▶__proto__: Array(0)
  ▶2: (6) [-0.0011023245751857758, 0.01475568674504757, 0.3497205078601837, 0, -0.268446683883667, 0.38042736053466797]
  ▶3: (6) [0.060080476105213165, 0.015505359508097172, 1.1656665802001953, 0.04401669278740883, -0.1551397740840912, 0.3712512254714966]
    4: 0
    length: 5
  ▶__proto__: Array(0)
```

Figure 5 An example of FaceDetected events' return value when one face is recognized (“eetu” in this case).

As I’m only interested in finding the label attached to a recognized face, based on the documentation I can write the following code to cover all the possibilities given in the documentation.

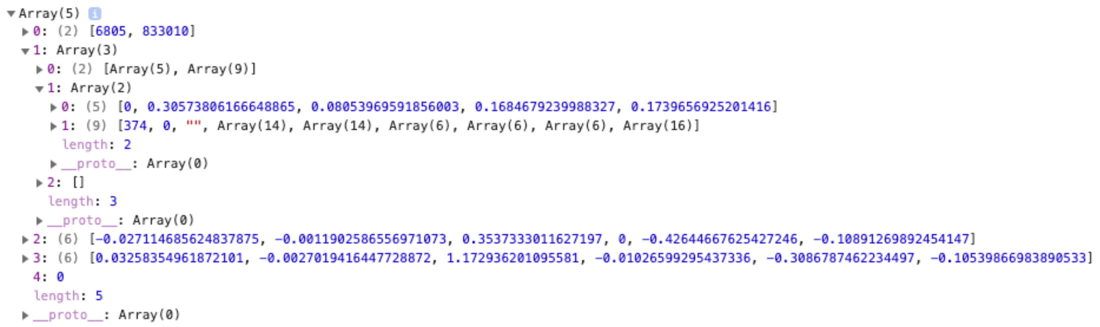


```
// Value is the value returned from
// the FaceDetected-event
const getNameOfDetectedFace = value => {
  if (value.length > 1) {
    if (value[1].length > 1) {
      if (value[1][1].length > 1) {
        if (value[1][1][1].length > 0) {
          return value[1][1][1][0];
        }
      }
    }
  }
  return null;
};
```

Figure 6 The code for getting the label of the recognized face.

The first check if done because the event sometimes returns an empty array. The second check is to make sure the Time_Filtered_Reco_Info -part is available. The third condition is to ensure that there's some content in Time_Filtered_Reco_Info. The last if-statement verifies that there is atleast one label available. If all the conditions evaluate as true, the function returns the first label found in the array. Otherwise it returns a null-value.

However, I found out that there's a fifth option not listed in the documentation. The Time_Filtered_Reco_Info can also be an array containing two arrays. This happens when you present a face to the robot that it cannot put any label to. Meaning not every unknown face will cause this, because it can mislabel the person as someone else.

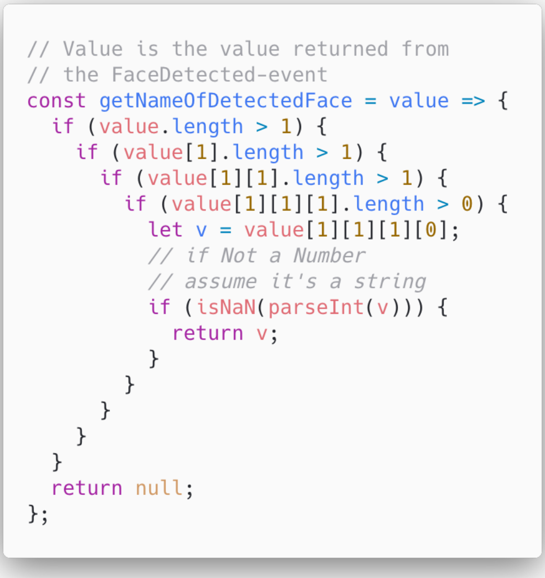


```
▼Array(5)
 ▶0: (2) [6805, 833010]
 ▼1: Array(3)
   ▶0: (2) [Array(5), Array(9)]
   ▼1: Array(2)
     ▶0: (5) [0, 0.30573806166648865, 0.08053969591856003, 0.1684679239988327, 0.1739656925201416]
     ▶1: (9) [374, 0, "", Array(14), Array(14), Array(6), Array(6), Array(6), Array(16)]
      length: 2
     ▶__proto__: Array(0)
   ▶2: []
    length: 3
   ▶__proto__: Array(0)
 ▶2: (6) [-0.027114685624837875, -0.0011902586556971073, 0.3537333011627197, 0, -0.42644667625427246, -0.10891269892454147]
 ▶3: (6) [0.03258354961872101, -0.0027019416447728872, 1.172936201095581, -0.01026599295437336, -0.3086787462234497, -0.10539866983890533]
  4: 0
  length: 5
 ▶__proto__: Array(0)
```

Figure 7 The fifth, undocumented return value for FaceDetected-event. The function in Figure 6 will return 374 with this data.

In this new case, the function will return an integer value with no meaning.

With this new knowledge I can modify the function to check for the type of the found value. The function now returns a null also in the case where the found value is an integer.



```
// Value is the value returned from
// the FaceDetected-event
const getNameOfDetectedFace = value => {
  if (value.length > 1) {
    if (value[1].length > 1) {
      if (value[1][1].length > 1) {
        if (value[1][1][1].length > 0) {
          let v = value[1][1][1][0];
          // if Not a Number
          // assume it's a string
          if (isNaN(parseInt(v))) {
            return v;
          }
        }
      }
    }
  }
  return null;
};
```

Figure 8 The modified function to also handle the fifth case.

5 DISCUSSION

In this chapter I'll provide some closing thoughts about Pepper, developing an app for it and explore some use cases for the robot.

5.1 The developer experience

When I first started to work with Pepper, I was quite enthusiastic to build software for a humanoid robot. I was also excited to use all the different interaction methods the robot has. I especially wanted to build features using speech recognition.

It didn't take long for the first problems to surface. Due to some of the issues, I had to cut down my list of features that I would have liked to use. The limited capabilities of the speech recognition system forced me to rethink a lot of the dialogue I initially had in mind.

After realizing the limits of the robot I got a little worried. Could I make the application interesting enough so that people would like to use it.

To my relief, the preliminary test results showed that I had managed to make the application interesting enough. Most of the elderly on the test group enjoyed using the app.

I believe the developer experience could be improved with rather simple solutions. First step would be to fix the documentation: remove any code that produces errors and document all different forms of the data. With these two corrections the developers wouldn't need to spend time figuring out the weird actions in their code.

One possible way to improve the documentation could be to publish it as a separate open source project. By open sourcing just the documentation, the company would still hold the rights to the code, but the documentation could be improved by anyone. Any developer encountering a mistake could propose a fix for it or even write about a feature not covered in the documentation.

This project has certainly reminded me about the importance of documentation in sofware projects. Especially in products like Pepper, where the global developer community isn't that large, the official documentation is possibly the only place to seek guidance.

One other improvement could be to provide some convenience-functions for the developers. For example, in my app, I wasn't interested in any other data than the name of the recognized face. But to get the name, I had to parse through an array with irrelevant data. Had there been something like a getNameOfRecognizedFace() - function, the development would have been much easier and less prone to erros. As a side effect, the developers could have possibly spotted the fifth data-form that was not documented.

5.2 What others have already done based on this work

The developed application has been tested in two different care homes for elders and the reception has been good, from both the elders and the employees. More studies based on this app and the findings are being written and published later.

The upcoming studies focus on the human-robot -interaction and the preliminary results are promising as many of the elders in different test groups enjoy interacting with the robot. There's also been an interest at a national level about the possibilities and use cases for this application.

5.3 The pros of Pepper

Despite the developer experience being a bit disappointing, there are positive things to say about Pepper. People from other fields than technology have been interested in the robot. For example, many health care professionals have come forward and asked for a presentation about it. Many non-technology-oriented people have even come up with new ideas on how to utilize Pepper in their field of work (for example in schools or in day care).

Pepper has lowered the barrier of approaching robots for people with no knowledge about them. When asked about the reasons for this, the usual answer is “it isn’t some scary gray box”. When asked to give a more descriptive answer, the answers usually include the human like features; the face (with the eyes and the mouth) and the hands with the fingers.

I believe that one fitting use for Pepper is what the introduced app was developed for; that is, it can offer a basic form of companionship for the elder. Simple interaction like responding to a few words is easy enough to program. Also the built-in feature of the robot just nodding at random intervals while being talked to was interactive enough for our test group. It gave the elders the feeling of someone reacting to what they were saying, which is an important part of any interaction.

One other use for it could be in customer service; for example in stores or in shopping malls. It could point people to the right direction based on keywords, such as the name of a specific shop or a certain item. The guidance could be done by displaying an interactive map in the tablet with the route highlighted and the robot speaking some general directions.

5.4 What this work enables

Most of the people in health care business who have seen the developed app have been quite enthusiastic about the possibilities that robotics and technology can bring to their work. The most common added value is the easing of the workload of the employees. While the robot keeps company to the elders, the nurses can focus on dealing with other daily tasks.

From technology’s point of view, I hope this work will make the development process easier for people starting to write software for Pepper. One other hope I have is that this work will help in the comparing different humanoid robots.

While Pepper has its drawbacks and it isn’t the most advanced robot available, I believe it is a valuable step in the development of humanoid robots. My colleagues in the field of health care technology have become quite interested in the possibilities of humanoid robots after seeing this app. Pepper will give us a solid baseline to compare to in the future. When acquiring the next humanoid robot, we know what kind of features we want and we will most definitely pay extra attention to the developer experience.

REFERENCES

Babel team. (2019, September 6). *Babel*. Retrieved September 22, 2019, from https://babeljs.io/

Cybernetic Zoo. (2011, September 5). *1988 – "Manny" Robot Mannequin – (American)*. Retrieved September 29, 2019, from http://cyberneticzoo.com/robots/1988-manny-robot-mannequin-american/

Ecma International. (2011, June). *ECMAScript® Language Specification*. Retrieved September 22, 2019, from https://www.ecma-international.org/ecma-262/5.1/

Ecma International. (2015, June). *ECMAScript® 2015 Language Specification*. Retrieved September 22, 2019, from https://www.ecma-international.org/ecma-262/6.0/

McGinnis, T. (2017, October 3). *Compiling vs Polyfills with Babel (JavaScript)*. Retrieved October 5, 2019, from https://tylermcginnis.com/compiling-polyfills/

Reese, H. (2016, August 11). *Pepper the robot: The smart person's guide*. Retrieved March 28, 2016, from Tech Republic: https://www.techrepublic.com/article/pepper-the-robot-the-smart-persons-guide/

Refell Family History. (2019). *AH Reffell & Eric Robot (1928) - The UK's First Robot*. Retrieved September 29, 2019, from http://www.reffell.org.uk/people/ericrobot.php

SoftBank Robotics Europe. (2018, October 26). *ALSpeechRecognition API documentation*. Retrieved October 2, 2019, from http://doc.aldebaran.com/2-5/naoqi/audio/alspeechrecognition-api.html

SoftBank Robotics Europe. (2018, October 26). *ALSpeechRecognition Tutorial - Full example*. Retrieved October 2, 2019, from http://doc.aldebaran.com/2-5/naoqi/audio/alspeechrecognition-tuto.html#full-example

SoftBank Robotics Europe. (2018, October 26). *Microphones*. Retrieved February 4, 2019, from http://doc.aldebaran.com/2-5/family/pepper_technical/microphone_pep.html

SoftBank Robotics Europe. (2018, October 26). *Pepper face detection*. Retrieved March 5, 2019, from http://doc.aldebaran.com/2-5/naoqi/peopleperception/alfacedetection.html

SoftBank Robotics Europe. (2018, October 26). *Pepper face detection, relearning a face*. Retrieved March 5, 2019, from http://doc.aldebaran.com/2-5/naoqi/peopleperception/alfacedetection-api.html#ALFaceDetectionProxy::reLearnFace__ssCR

SoftBank Robotics Europe. (2018, October 26). *Pepper JavaScript SDK*. Retrieved April 1, 2019, from http://doc.aldebaran.com/2-5/dev/js/index.html

SoftBank Robotics Europe. (2018, October 26). *Pepper SDKs*. Retrieved April 1, 2019, from http://doc.aldebaran.com/2-5/dev/js/index.html

SoftBank Robotics Europe. (2018, October 26). *Pepper setVocabulary-function*. Retrieved March 31 2019, from http://doc.aldebaran.com/2-5/naoqi/audio/alspeechrecognition-api.html#ALSpeechRecognitionProxy::setVocabulary__std::vector:ss:CR.bCR

SoftBank Robotics Europe. (2018, October 26). *Pepper speech recognition*. Retrieved March 5, 2019, from http://doc.aldebaran.com/2-5/naoqi/audio/alspeechrecognition.html

SoftBank Robotics Europe. (2018, October 26). *Pepper tablet*. Retrieved March 5, 2019, from http://doc.aldebaran.com/2-5/family/pepper_technical/tablet_pep.html

Softbank Robotics Europe SAS. (2015, November 2). Retrieved March 31, 2019, from Github: https://github.com/pepperhacking/studiotoolkit

Statistics Finland. (2015, October 30). *Population projection 2015-2065*. Retrieved October 13, 2019, from https://www.stat.fi/til/vaenn/2015/vaenn_2015_2015-10-30_tie_001_en.html

Wikipedia contributors. (2019, September 27). *Nadine Social Robot*. (T. F. Wikipedia, Producer) Retrieved September 29, 2019, from https://en.wikipedia.org/w/index.php?title=Nadine_Social_Robot&oldid=918225655

Wikipedia contributors. (2019, September 24). *Nao (robot)*. (T. F. Wikipedia, Producer) Retrieved September 29, 2019, from https://en.wikipedia.org/w/index.php?title=Nao_(robot)&oldid=917538665

Wikipedia contributors. (2019, July 8). *Polyfill (programming)*. Retrieved October 5, 2019, from https://en.wikipedia.org/w/index.php?title=Polyfill_(programming)&oldid=905401257

Wikipedia contributors. (2019, June 16). *Source-to-source compiler*. Retrieved September 22, 2019, from Wikipedia, The Free Encyclopedia.: https://en.wikipedia.org/w/index.php?title=Source-to-source_compiler&oldid=906556977

You, E. (2019). *VueJS*. Retrieved September 22, 2019, from VueJS: https://vuejs.org/