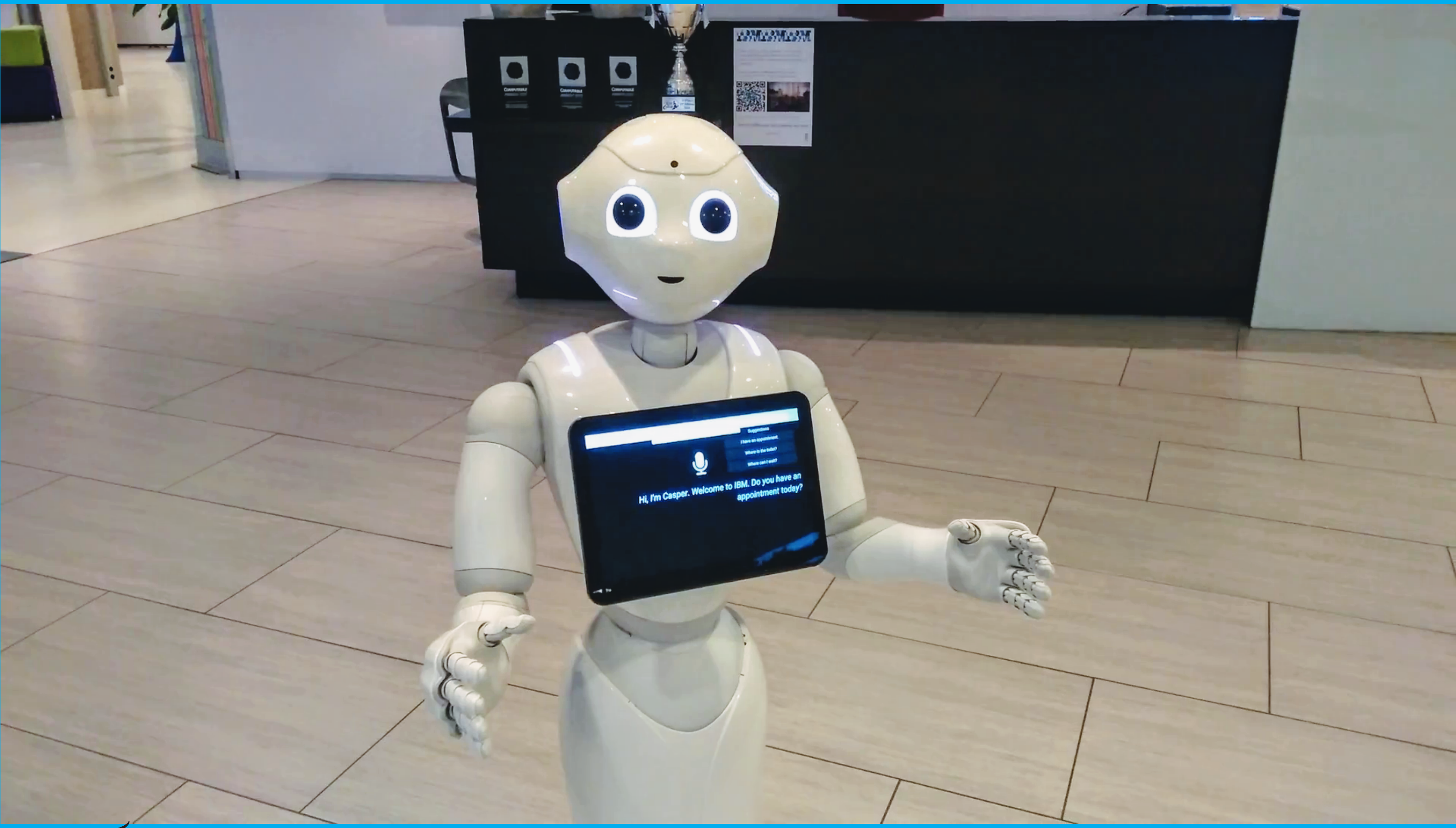
Exploring trade-offs in on-board versus cloud-based social robots

M.C. de Graaf

Msc. Thesis


Hi, I'm Casper. Welcome to IBM. Do you have an appointment today?

TU Delft

Exploring trade-offs in on-board versus cloud-based social robots

by

M.C. de Graaf

to obtain the degree of Master of Science in Computer Science
at the Delft University of Technology,
to be defended publicly on Friday August 23, 2019 at 14:00.

Student number:	4172949	
Project duration:	September 1, 2017 – August 23, 2019	
Thesis committee:	Dr. K.V. Hindriks,	TU Delft, supervisor
	Dr. Hayley Hung,	TU Delft
	Zoltán Szlávik & Benjamin Timmermans	IBM Benelux, Center for Advanced Studies

Abstract

Social Robotics is an emerging field in Computer Science. Most social robots currently commercially available to buy do not have fast hardware components. As a result, the built-in software has low accuracy and performance with (amongst others) speech and facial recognition and dialogs during social interaction with users. Cloud computation offers state-of-the-art techniques, performance, and accuracy with its massive available computational power, but at extra costs and increased latency.

In this work, we extend and improve a social robot's standard capabilities and performance by making use of cloud computation. This thesis covers an exploration for the trade-offs present when replacing or augmenting built-in robot software with IBM cloud services, based on the humanoid Pepper robot from Softbank.

The approach for this exploration was guided by a hospitality use case demonstrated in the offices of two companies: a Dutch Health Insurer and IBM Netherlands. Two products were developed for this single use case based on different development toolboxes. The first toolbox contains all development software from the robot's manufacturer (the NAOqi toolbox), while the second toolbox makes use of cloud services (the Watson Toolbox). Using the product built with the NAOqi toolbox, we evaluate interactions with real users and obtain baseline data and experiences. After evaluating the second product built with the Watson toolbox, we can compare differences in human-robot interaction quality, robot component quality, development methods, and software engineering complexity and Total Costs of Ownership for both products.

The main findings include an overview of relevant test metrics and test methods for a social robot's component, including acquired data for some components of the Pepper robot. We show possible architectures for a (semi) cloud-based system, and their trade-offs. Evaluations show that the cloud-based system indeed performs better and has higher human-interaction quality compared to the product built with the NAOqi toolbox, yet downsides such as latency and operating costs are present. This is also reflected in the analysis of single components, where specifically Speech-to-Text from the cloud shows a significant increase in performance and capabilities. We show that a mix of toolboxes results in the best working and cheapest social robot when considering Total Cost of Ownership. IBM Cloud pricing structures and operating costs are analyzed for this. Finally, we contribute to the currently available knowledge on this subject with a decision matrix combining all previously mentioned information in a compact form accessible to people not knowledgeable in the hospitality robot or cloud domains. With the matrix, early-development advice decisions for creating a social robot can be formulated using the data gathered in this thesis.

With a broad approach, this research focuses on finding and discussing trade-offs, rather than an in-depth analysis on all component. Providing methods to determine the data as mentioned above, findings, and trade-offs are more important than the actual numbers found in this thesis, as advances in this domain are quick and expected to change often. The end product built using the Watson toolbox is an improvement on multiple levels, yet is still not always able to autonomously and correctly finish all intended interactions. However, its capabilities, performance, and robustness are closer to the level of being used commercially. The techniques we use to extend Pepper's capabilities could be applied to any social robot.

Preface

Work on this thesis started two weeks after returning from a work assignment with robot Pepper in Tokyo. At that point, it was still unclear what this thesis would exactly be about, but it was clear it involved building two new robot applications at two different companies. After spending two weeks in a hotel room literally sleeping next to Pepper, almost having a kind of connection with the robot, I thought this would not be a big challenge. If only I had known.

I want to thank my supervisors from both companies for their support and all the valuable things I have learned from them in this process. I'd like to thank Koen Hindriks for helping me formulate the assignment and coach me from the beginning until the last days. In our meetings we did not only talk about my thesis but also about the broadest hypothetical situations robots could end up in, resulting in interesting and fun discussions with our sometimes completely different views.

I also want to thank Joachim de Greeff for being my guide and valued colleague at the start of my thesis with Interactive Robotics, and maybe more importantly, introducing me into the world of robotics in the first place four years ago.

I'd like to thank the main people of the CAS team at IBM, supervisors Zoltán Szlávik and Benjamin Timmermans for their guidance, patience, and wisdom, but maybe most of all for all the fun times we had together. This of course also includes Manfred Overmeen, who could not only help me with serious and technical questions I posed to him, but also provided me with a lively working space with all kinds of fun, interesting and innovative creations around me, including the much needed distraction-device called a 3D printer.

My family has been a great source in giving me the motivation to continue for which I am very thankful. I enjoyed moments to really relax and forget about all the work still ahead of me, because of them, for which I am grateful.

A special mention must be given to Jamey Sparreboom, who did his thesis at the same time as me and was always ready to listen and give feedback. We also worked together on obtaining results for our speech recognition theories, for which test data is used in this thesis. We have spent a considerable amount of time together apart from the thesis work, which helped me to keep my sanity.

Throughout this project, I have spoken to, laughed with, and learned from so many people that it would take the length of this thesis again to name them all. A concluding big thanks to everyone at IBM, Interactive Robotics, the Insight Lab of the TU Delft or general friends for all the fun, serious conversations, shared desperation or reassuring words!

Now, on one of the last days of an almost two-year journey, I look back to a difficult process, lots of positive, but also negative emotions and experiences. For most, I'm happy to have had them and become wiser because of it. Now, it is vacation time!

Ending with a dreary note, I want to dedicate this work to my mother Marlene Bredius, who sadly passed away just before the end of 2018. In her last months, she intensely looked forward to me finishing my thesis and attending the presentation. Unfortunately, she went too soon and is greatly missed.

M.C. de Graaf
Delft, August 2019

Contents

1 Introduction

This chapter provides a broad introductory context for this thesis on the related areas of speech controllable devices, intelligent and social robotics, and cognitive machine processing services of IBM Cloud. Next, the problems in current robot hospitality systems, leading to the research questions of this thesis, are discussed in Section 1.2. Finally, the approach to answering these questions is described in Section 1.3.

1.1. Context

Imagine walking into an office building. Your attention is drawn to a robot waiting at the door, friendly greeting you and asking about your visiting purpose. You reply with the details of the appointment you have made before. While the robot informs an employee of your arrival, you are offered information on the company and can a play little game while waiting. When leaving, the robot waves goodbye and tends its attention to newly arriving customers. A robust personal social robot hospitality experience without needing a single human employee, how can this be achieved?

This thesis covers the development of this scenario with a Pepper robot. The possible ways of implementation are discussed, as well as metrics, tests, trade-offs, and methods to determine which way works best. The broader focus is targeted at using cloud services to improve general interaction performance since current solutions using onboard hardware often seem to underperform in the aforementioned scenario.

Computer systems have evolved tremendously over time, as are the ways us humans interact with them. Where initially the mouse was a revolutionary input device compared to a keyboard, we now have an input device for almost every modality one can think of, although the difference in usability for general use differs.

While speech is one of the most used forms of human to human communication, this is not necessarily true for human-computer and human-robot interaction. Since speech recognition rates are at an acceptable level for general use nowadays [1], more and more systems like Google Home/Assistant, Alexa or Siri are implemented which can (mainly) be controlled by voice commands and give feedback using spoken text. Communication is often still restricted to simple commands and utterances though, and the use of true natural language use is limited. The same holds for current human-robot interaction, where it is (for now) uncommon to use natural language [2].

Aside from intelligent systems we can speak to and communicate with, the (social) robot revolution is alive and quickly developing. Whereas a decade ago robots were almost exclusively in use at production lines and other large industrial locations for specific and straightforward automation tasks, nowadays robots are getting into the home [3], workplace[4], retail stores [5] and other parts of society. A robot has to meet some minimal standards to be interesting for home users (and companies): it should offer a good service for an affordable price, next to having a minimal failure rate combined with enough independence. One example of a robot which meets these criteria is the vacuum cleaner Roomba service robot [6].

With the introduction of this and likewise robots, the amount of interaction between humans and robots is growing. Based on a early proposal by Duffy [7], we define a Social robot as

"a physically embodied entity capable of performing social behaviors attached to its role".

Examples are NAO[1], Jibo[2] and Pleo[3] which are more and more used as human helpers, assistants or secondary living companions. Also part of this list is Pepper,[4] a humanoid robot of about 1.2m high developed by Aldebaran (now SoftBank Robotics).

To discover how to create a scenario like the one presented at the start of this chapter, a robot is wanted that

- is large enough to talk to comfortably when standing and easily spotted in a room
- looks friendly, so people are willing to approach it and are not scared away
- can sense the environment in various ways using microphones, vision sensors, obstacle detection, location awareness, and a wifi connection
- can influence the environment using speakers, arms, wheels
- can provide visual feedback using a tablet or changing posture
- allows for various programming alterations
- can be used commercially (approved as safe to use by regulators, long battery power, widely available to order)

As the Pepper robot fulfills these requirements and is fairly easily obtainable, it is used during this thesis.

About Pepper Pepper is a humanoid robot designed to interact with humans naturally and intuitively. It is equipped with various kinds of sensors such as microphones, (3d) cameras, gyroscope, touch sensors, sonars, and lasers as well as a tablet, speakers and several LED's to communicate[5]. Processing power and sensory information are limited (compared to regular computers, not compared to similar robots), and capabilities such as speaking in natural language, facial recognition or (a form of) intelligence are standard available only in a basic form. This means that these features are available, but are limited in options, adaptability, intelligence and performance, and needs configuration every time. These functionalities run on the robot resulting in quick response times.

Pepper weighs around 28 kg and can function for around 12 hours on one full battery charge. This, together with the other characteristics, makes it suitable for use in an office since the robot can work for a long time, is not intimidating and not dangerous to operate. With its wheelbase (legs are reserved for Pepper's larger brother ROMEO) it can drive up to 3 km/h. Aside from its built-in autonomous abilities, the robot can be controlled manually using a Wifi or Ethernet connection.

From the author's experience, some practical issues were noticed when performing various activities with Pepper. When people see Pepper for the first time, the following kinds of behavior can be observed: because of anthropomorphism, people expect the robot to be able to talk about and respond to anything. They start asking random unrelated questions, speak only with keywords, talk to it like a child or talk excessively verbose resulting in a dialog system giving unexpected answers. Also, environment noise can provide useless or twisted results from speech to text functionality, or detect speech when no one is interacting with the robot resulting in an unneeded response. As the technology is still new, the robot does need to be prepared for these kinds of interactions and make sure to create correct expectations, especially in places where almost every interacting user uses Pepper for the first time as is the case with the hospitality use case. The robot should be transparent in displaying capabilities and current options in the interaction.

Toolbox The Pepper Toolbox is defined in this document as a set of tools used to program and control the Pepper robot (and other robots running NAOqi OS). The toolbox includes Choregraph (flow-based visual drag&drop programming), NAOqi (higher-level operating system) and a set of SDKs for various programming languages such as Python, Java, C++, and Javascript, as well as an interface to ROS (Robot Operating System). Due to high levels of abstraction, even novice programmers can quickly develop applications for the robot.

While that toolbox is easy to use and can be extended as wanted, it does have several drawbacks. Standard built-in modules such as face recognition, speech recognition, and dialog have limited options for configuration and developers often need to introduce extra constraints (on the environment or interaction) to make

[1]NAO robot: `https://www.ald.softbankrobotics.com/en/robots/nao`
[2]Jibo: `https://www.jibo.com/`
[3]Pleo: `http://www.pleoworld.com/pleo_rb/eng/lifeform.php`
[4]Pepper: `https://www.ald.softbankrobotics.com/en/robots/pepper`
[5]For full specifications, see `https://www.ald.softbankrobotics.com/en/robots/pepper/find-out-more-about-pepper`

sure performance is acceptable for use. As mentioned, the Pepper robot does not come with a processor fast enough to perform lots of different analysis algorithms quickly aside from the standard basic functions[6]. While an option is to improve the robot's hardware, this is more expensive and more importantly, consuming substantially more battery power limiting full-day charger-less operation and more heat-dissipation is needed. Off-loading these calculations from the robot, for example, to the cloud, could help.

Cloud robotics Cloud computation is used more and more nowadays as options grow bigger, and prices decrease. Various companies are offering cloud services. Pepper could benefit from the capabilities and resources of the cloud to compensate for its technological constraint in terms of storage, processing power, and communication.[8]. The cloud offers several significant advantages such as continuous software updates, so one always has the latest innovative technology available, pay-as-you-go service (only pay for what is used), scalability, secure and easy storage, quick setup, and prototyping. It also offers mobility with the use of online editors, for example, to let a company employee change an interaction dialog, instead of a programmer.[9]. By making use of the increased computational power when using a cloud service, one would think performance and interaction quality of a robot could be improved, even when sensor data from the robot is not always perfect. Cloud services use specific dedicated hardware substantially outperforming the built-in hardware of Pepper. For example, processing an image to check whether a face is present could be performed faster and more precise in the cloud due to the increased computational power, next to other simultaneous analysis such as person's age, gender, and even emotional state. In the past years the trend of using a cloud infrastructure in products, also for robots, has become clear by the emergence of companies providing specific cloud based solutions for robots, as for example Interactive Robotics[7], Ortelio[8], C2RO[9] and Rapyuta Robotics[10] have done or are developing now as this could prove a broad market opportunity now and in the future[10].

Several companies, such as Google (Cloud), Amazon (AWS), and Microsoft (Azure) offer (cognitive enabled) cloud services. They all offer comparable machine learning functionality, but with different forms of payment and ease of setting up. It was chosen to use IBM Cloud for this thesis. The main reason for this is due to performing this thesis at IBM Benelux. Apart from that, IBM Cloud offers various advantages compared to other cloud suppliers. It allows to run, deploy and manage applications over the cloud in a few minutes whereas more (configuration) effort is needed for the other providers' services, it supports many different programming languages, and service accuracy is high. Also, IBM's Assistant (Conversational agent) and Speech Recognition service is considered best among the largest cloud providers [11] which are important elements for the robot to improve upon.

Toolbox using the cloud IBM Cloud consists of a broad range of intelligent machine processing services running on the IBM Cloud platform and are accessible through various programming and interfaces. IBM keeps updating and improving these high-performance services, which can directly be implemented on running applications. They can be used as a cloud infrastructure for processing Pepper's sensor data and other interactive workings. Although many services exist[12], only a subset is needed to implement a social robot. If implemented properly these AI services are expected to exceed the results that can be obtained using the standard Pepper Toolbox. This is because of the available processing power, extensive training models, continuous improvements, and a multitude of various concurrently running algorithms to get the desired results. Combined with a form of control logic (as multiple implementations are possible) these services are coined the 'Watson Toolbox' for the rest of this document.

In this thesis, a global comparison is made of IBM Cloud functionality, which is relevant for social robotics. For a robot to perform social interaction skills (as described in the introductory scenario), at least the robot must be able to process audio, vision and communicative utterances. This corresponds to the use of the following subset of AI services from the IBM Cloud: Natural Language Understanding and Processing, Assistant (dialog), Visual recognition, Speech to Text, and Text-To-Speech. Also relevant and suitable for extended functionality on a social robot could be knowledge services such as Discovery (News) for insights in data

[6]Pepper processor specifications: http://doc.aldebaran.com/2-5/family/pepper_technical/motherboard_pep.html
[7]Interactive Robotics: https://www.interactive-robotics.com/
[8]Ortelio: http://robotics.ortelio.co.uk/
[9]C2RO: http://c2ro.com/
[10]Rapyuta Robotics: https://www.rapyuta-robotics.com/technology_cloud/
[11]Best Speech Recognition: http://www.businessinsider.com/ibm-speech-recognition-almost-super-human-2017-3?international=true&r=US&IR=T and Conversational agent https://www.ibm.com/blogs/watson/2018/04/forrester-ibm-leader-conversational-computing-platforms-wave/
[12]IBM Watson services overview: https://www.ibm.com/watson/products-services/

and recent news, Weather or Personality Insights for tone and characteristics through text analysis. The IBM cloud does not offer a robot navigation service and is not relevant in the scenario and will thus not be covered. Google announced that a cloud robotics platform to be launched in 2019 would support cloud robot navigation[13], including a whole ecosystem for connecting robots with the cloud.

Using these kinds of cloud services also has several inherent disadvantages, some mentioned before as advantages. First and foremost: processing time costs money, often paid per minute or API query. Although prices are relatively low per request, performing requests all day could still result in high costs. Also, a stable internet connection is always needed if the functionality is dependent on the cloud service, and no local fallback alternative is available. Lastly, the cloud often implements improvements, but these changes can also result in a service becoming incompatible, merged with another service, or even deleted after which the developed product stops working without the client having any influence on that process. When implementing cloud services for a robot solution, this needs to be taken into account.

1.2. Problem definition

Social robotics is a relatively new field of Computer Science and Engineering. Specific issues for robots such as localization [11–13], planning [14] and interaction [15] have been addressed significantly in labs, and often successfully, but limited success has been reached with developing robots that can operate in real situations for an extended time.

Although the Pepper Toolbox offers much functionality by itself, the drawbacks mentioned in the introduction could be improved upon. Utilizing the benefits of cloud services seems a way to achieve this improvement. However, no clear data is available on the differences between cloud-enabled social robots and ones using built-in hardware and software.

To illustrate some open questions motivating this research, think about the following. What changes when one moves sensor information processing from on the robot to being performed in the cloud? It could improve performance and accuracy with the substantial increase of processing power available on the cloud, but what must the robot do when the connection to the network is lost? Does processing cloud data requests and results happen fast enough to use for a (social) user experience? If not, do we need to define minimal response times for the robot and otherwise fall back to local modules? What functionality and processing should run on the robot, and what should run in the cloud, taking the above questions in mind? Does using the cloud save time and effort for developers by reducing complexity and configuration time as if often suggested[8], or is it increasing development time because of increased architectural complexity?

These are complex questions, and developers and companies want to make an informed choice when considering to implement cloud services in robot software development. However, no method currently is available to do this. Trade-offs involving costs, performance, accuracy, and development effort are present and differ per system component. The cloud could prove a way to increase robot performance, save development costs, and always have an up-to-date product, next to other advantages. Disadvantages like latency and continuous usage costs are also present when using a cloud infrastructure which needs to be considered. A method that facilitates making choices between using Pepper Toolbox components versus Watson Toolbox components and choosing between these trade-offs is the most significant contribution of this thesis.

To evaluate and get more insight into whether using the cloud in robot products is advantageous in the end we look at a specific hospitality use case using the Pepper robot, for which also two actual products will be fully developed. The hospitality use case is chosen because it allows the robot to make use of all its modalities (think of for example speech to text, dialog, and face classification) but not necessarily needing all functions such as face or object recognition. The scope is kept small as no movement is needed in this use case removing the need for localization. Next to that, the use case can be made concrete with a clear task objective. Also, the environment for this type of use case is challenging but still managed (human support is always present). When looking around at current prototypes, the hospitality domain seems promising for the first real-life robot applications.

Pepper is chosen as a physical robot form since it is broadly commercially available and a well-known example of social robotics while having the correct form factor for the intended use case. Next to that, the author of this thesis is already experienced with the various programming options and limitations for this robot.

[13]Google Cloud Robotics: `https://cloud.google.com/cloud-robotics/`

Given the information in the previous sections on the two toolboxes, cloud and hospitality, we have enough background to pose the research objective of this thesis:

`Which trade-offs have to be taken into account when making a choice between using the Pepper Toolbox versus the IBM Watson Toolbox for creating a social hospitality robot?`

To address this question, we take a broad look at the different aspects necessary for making that choice. These aspects, relevant for answering the main research question, are then translated into a subquestion.

1.2.1. Research question 1

The first aspect is the artificial intelligence component quality. One will not (want to) move to AI components in the cloud when these are not better performing than the local working versions. Thus, a qualitative comparison needs to be performed, determining whether cloud components perform better than a local running solution. This is a requisite for considering a cloud solution, as otherwise there is no clear advantage for looking into this. If so, one can make use of the best of both worlds and use a selection of cloud services, suggesting an architecture supporting dynamic loading of built-in and cloud modules.

1. `How does the component quality of the built-in Pepper Toolbox compare to the IBM Cloud services?`

Approach With a focus on the quality of the AI components themselves, an analysis is performed which services are comparable, followed by an analysis between these cloud services and NAOqi modules using metrics relevant for that service-module pair (e.g., the keyword error rate in case of a Speech-to-Text service).

1.2.2. Research question 2

The next aspect is the social interaction quality. Although related to the previous question on AI component quality, this focuses more on the social part where users and context are important factors. The interaction itself must not get worse and preferably get better by using a cloud solution. Waiting ten seconds for a response from the robot is not workable for most use cases. It needs to be tested how using the AI components changes the interaction itself.

When the hospitality robot is implemented in an office, it is to be expected that it needs to perform for a complete workday. The robot is likely the first 'face' a potential customer or client sees, and consequently, the company wants to leave a positive impression. The robot should thus perform correct and react according to the user's expectations. Also, it should assist existing personnel instead of adding extra work to their duties by constantly needing help or attention. Resetting the robot every hour because it fails is unwanted and will create resentment among existing personnel (and users).

Implementing an effective hospitality robot hence requires certain robustness, be mostly autonomous and should handle correct and accurate to the social role it is given.

2. `How does the use of Watson's versus Pepper's toolbox compare with respect to the quality of the human-robot interaction?`

Approach To test how interaction quality differs, two products built using the two toolboxes are developed and evaluated for their performance and user liking in an interaction. Metrics as response latency and completion of a 'happy flow', indicating completion of interaction as intended by the developer, are analyzed.

The word 'quality' in this question is meant to describe how effective and robust the interaction is. An example of analysis is the conversation module, where differences in user input can substantially change consecutive interaction, such as extracting entities and intents instead of listening for specific words. Also, can the system be made robust enough to leave it unattended for a whole workday?

1.2.3. Research question 3

The third aspect is software engineering complexity. When the solution takes much more time to develop or the architecture is complex, or the solution is not easily extensible, developers would not quickly recommend using that solution. The implementation needs to make sense for developers to embrace a cloud-based approach.

How much effort is spent by developers on making a solution running locally versus (partially) in the cloud? Also, what are differences in approach and used architecture? What does one lose or gain when choosing for another architecture in terms of learning curve, capabilities, maintainability, and ease of implementation?

3. `How does software engineering complexity differ for an architecture using only built-in components versus one that also uses cloud components?`

Approach With a focus on differences in development effort and ease of implementation, we look at developer options, testing time and setup, architecture complexity, technical difficulties, component configuration, and general setup.

1.2.4. Research question 4

The fourth and final aspect important to consider, especially for managers and clients, are (operating) costs. For non-cloud solutions, development costs are a one-time expense (when not considering maintenance), and further product usage is free. For cloud solutions costs often also include cloud usage fees for the rest of the products life cycle. Whether this is a periodically billed predetermined amount or a pay-per-use variant like pay per API-call or CPU cycle, this has to be taken into account when developing the product to determine the Total Cost of Ownership (TCO) for the product's lifespan. By having quantifiable insights, business value can be optimized.

An example of this is facial recognition, which performs fast and accurate in the cloud but can become costly quick. When naively implemented, say, sending a stream of images with the rate of an average video sensor (25 FPS) at €0.003 per image, one is looking at a hourly running cost of €270,- (if a face is present in the image, classification of a face is cheaper with half the price). Of course, this gets cheaper with many requests and a face is not always present in the image, but this quick calculation gives a sense of what could happen if the cloud is treated equally to local processing where processing is 'free'.

4. `What are the up and downsides of using a cloud solution for robotics, when looking at business values?`

Approach With a focus on operating costs, an analysis is performed on the varying costs per API call. Also, a cost indication and analysis for having the social robot run an entire month is given next to possible solutions for reducing these costs. Also, a TCO (Total Cost of Ownership) calculation is made for a product build with toolboxes and a mixed version.

1.2.5. Concluding

Having the answers to the above four questions, one can make a better-informed choice as is intended by the main research objective. Instead of diving deep into one specific component, we take a broad approach and take every component and angle into account by implementing the complete product twice. By mapping the relevant factors, aspects, and metrics per question, an insight can be obtained in what role they play and how important they are for the trade-offs in the decision-making process. Based on this exploration, a method can be founded for companies or developers to ease the decision on how and whether to include the cloud in social robotics.

We hypothesize a significant overall increase in performance, accuracy, and efficiency compared to most NAOqi high-level modules when combining Pepper and IBM Watson modules running from the IBM Cloud platform. Furthermore, we expect that a lower amount of effort is needed to put into developing and configuring a robot application for a specific location when using the more general cognitive abilities of Watson.

The data results found during this thesis are expected to change in a relatively short time due to the fast-changing nature of AI services or software updates of the robot by Aldebaran. However, the way of gathering the data and overall results will stay valuable for a more extended period as this could be reused and remeasured using the same method.

A method on how to get these answers and be well informed is the most considerable contribution of this paper. The structure of this will be a categorized matrix with considerations to make and weighted factors, grouped per relevant component or service, as well as a TCO calculation. A case is made in the discussion whether a (partial) cloud solution offers a better business value and whether this is recommended.

1.3. Approach

To demonstrate the use of social robotics using a cloud infrastructure, we restrict the application domain to a concrete use case where Pepper serves as a hospitality robot receiving visitors and clients in an office building. The use case, fully described in Section 3.1, is kept simple as to demonstrate the possibilities of cloud integration outside a lab environment and in the real world, without introducing too many external variables or making the scope too broad for the primary research purpose. Outside of the scope are robotics research areas as navigation, mapping or localization since the robot is not expected to move much in our use case.

Within the role of hospitality robot, Pepper notifies employees when a client arrives for an appointment and keeps the client occupied and entertained while waiting to be picked up. Pepper can answer general questions or play a simple game with the user, and finally ask for feedback on the experience to improve future interactions and get data for our evaluation.

The use case is implemented twice as a functional product in a phased approach; once using the Pepper Toolbox to set a baseline for comparisons, followed by a solution using the Watson Toolbox. The two phases are performed at different companies. A general comparison between the two phases is made afterward.

The first phase is short, lasting two months, and intended to set a baseline for the second phase. The product at the end of this phase is lightweight, has all the basics (see requirements in the next chapter), but nothing more and all processes such as sensor processing and decision making run on the robot itself. The only toolbox that is used is the Pepper Toolbox. All modules consist of built-in blocks from Choregraph or self-made scripts (e.g., for retrieving a company-specific calendar) using built-in functionality. The final product can respond event-based on the input given by the users but is restricted by pre-programmed recognition of sentences as input, and pre-programmed responses as output. Alternatively, the tablet can be used as an input/output method in the same manner as a fallback option. This product is developed with a small team in which the author developed a larger part of submodules.

The second phase is more prolonged, lasting 9+ months, and aims to end with an IBM Cloud-based social robot. The end product makes use of different IBM Cloud services for as much of the functionalities specified in the requirements (Section 3.3) unless evaluations show that component performs better with the NAOqi toolbox. This could result in improvements with communication, due to better speech recognition results and natural language processing giving more insight on the user's intent and allowing more ways of giving user input than just keyword or sentence matching. Also, being quicker with person detection or even identification, using the robot's visual feed could change the way the robot interacts. As with the product from the first phase, the tablet will be used here as a fallback option, though mainly meant for displaying information or options for the user in the current interaction step. The aim is to do all input by the user through voice. This product, containing a middleware solution to connect with the cloud, a cloud infrastructure for control, logic and scenario development for our use case, will be designed, developed and implemented by the author only. This means a slower development time due to less available manpower, yet a comparison can still be made since working hours are tracked.

Comparing (development) between the two phases is only possible in general because of the difference in developer team size. The location also differs, with other types of clients speaking other languages, having other backgrounds (technically skilled versus unskilled) in a different context. Valuable lessons can still be learned however, as the same robot and the same use is used, and the differences in the environment do not influence the working of the technology itself.

When generally looking at (social) service robots, one can identify three levels which define its structure: Hardware (and mechanical structure, including sensors and actuators), control architecture (sequencing low-level behaviors performed at the hardware level) and the application level (planning, high-level behaviors, using control level)[16]. Each level is dependent on the level below, and thus low robustness in one of the lower levels results in low robustness at the top level. The main focus of this thesis will be on the application level, and partly the control level. The hardware-level cannot be changed, as the Pepper robot is not adaptable, as well as a large part of the control level.

The approach for the four subquestions defined in the previous section is presented below.

Question 1: To analyze the component quality of the toolboxes, the metrics of a pair of services are measured and analyzed. An approach is presented on performing these measurements and tests, so this can be reproduced at later times and with different models. A comparable pair of services means a NAOqi module and IBM Cloud service having the same or similar functionality. Measurements must be performed using the same context of the hospitality use case using the robot's sensors, e.g. faces captured by the robot's camera in an office, and not in a garden, for measuring the performance of the face classification modules, due to differences in lighting.

Some tests are performed: Word Error Rate and latency of Speech to text and the number of frames per second for Visual Recognition. To give more insight into the services, other tests are described, but no numbers have been obtained as this is not the main goal of this thesis. The approach and methods themselves are.

Question 2: For answering the second research question on the quality of human-robot interaction, two evaluations on the products developed with the Watson and Pepper toolbox will be performed. The behavior of the robot and user is observed and measured to compare the workings, pros and cons, and differences in the experience of both products. How do the products recover from errors, such as misunderstanding the user, losing internet connection, or even interaction-breaking problems such as the crash of one component?

We also look at which user interaction effects are dependent on the (partial) cloud choice. Robot response timing is important for interactions as humans are sensitive to that and can directly be related to latency effects introduced by the cloud. It could be an issue for giving a natural (the resemblance between human and robot behaviors) and pleasant interaction flow. Is it needed to define a minimal response time or do current services return results fast enough? Finally, we look if there are correlations between the user's details (age, gender, length of interaction) and how users approach the robot.

Added advantages and effectiveness of the cloud dialog module is described, along with the ease of configuration and the use of intents and entities. With this, one can retrieve information quicker, with more possible input sentences for the same functionality. Similar advantages in other services are also looked upon.

Question 3: On the research question of software engineering complexity, a description is given on both (and a mix of) architectures and their complexity on various levels. The amount of development time, obstacles during development, and pros and cons for developers for both solutions are discussed. Also, maintenance, technical challenges, and robustness of the solution are illustrated. For a quantifiable comparison, the amount of Lines of Code (LOC) and how to interpret these is discussed. The testing and debugging process of both products also differ. Since all these items are of influence to the developer and development process, we detail per item which toolbox has the advantage.

Additionally, we also expand on other systems available for this purpose and why these are not used for this project.

Question 4: The last question is on the up and downsides of the cloud when looking at business value. To start, the cost structure for cloud services, usually based on the number of API calls, is explained per component. The costs for running this component for one day in a standard configuration is estimated. Other configurations could mean significant cost reductions such as not continuously sending data but only when the interaction requires it. Options to do this are shown per component.

With this data available, we can calculate a theoretical Total cost of Ownership (TCO) estimate for the use of the robot for one, three, and five years of usage. This is done for both toolboxes and a mix, using a simple COCOMO method for effort estimation. Actual costs from the evaluation are also displayed for comparison Which solution (built-in, cloud, or a mix) is cheaper, and what is sensible to use? The trade-off is low development costs with high usage costs, or higher development costs but free thereafter (not taking maintenance into account).

2 Related work

This chapter presents related work to our research. This thesis builds on previous research on robot performance metrics, similar robot setups, (cloud) architectures, positioning, interaction, and offloading computation to the cloud. The following search terms have been used: *hospitality robot, social robot {pepper, nao}, cloud robotics, robots connected to cloud, human robot interaction, cloud robotics business value, 'software complexity' 'cloud robotics'* using filter 'from 2010+'.

Per research question we discuss relevant work. We found that for RQ1, and even more RQ4, little literature is available.

2.1. RQ1: Component quality

In this section literature is discussed related to gathering metrics from robot components, and how to compare those. This is related to the question: *How does the component quality of the built-in Pepper Toolbox compare to the IBM Cloud services?*.

In **Towards metrics of Evaluation of Pepper robot as a Social Companion for Elderly People**, an autonomous social robot is being developed for use with the elderly, using the NAO and Pepper robots[17]. This research focuses on benchmarking dialogue performance, mainly using emotion analysis with audio (various domains) and visual (facial expression) analysis added with speech recognition modules. However, built-in (from the Pepper' Toolbox) emotion recognition was used for visual recognition, next to an audio emotion recognition corpus not completely suited for the target test group, resulting in correct emotion recognition rates of 50% at best with often lower rates.

This paper points out that a social robot has to adapt its vocabulary, speech velocity, and behavior depending on age and emotions shown by the interacting user. Also, the paper suggests evaluation and engagements metrics for interaction such as user reaction time, silence time, and speaking time of the humans after a question by the robot. Analysis of emotion and changing robot responses accordingly can be integrated into our product as both the cloud and robot supports measuring this and changing the voice based on parameters. Due to time constrictions, this has not been included in our research, yet would add to the bigger picture. During our evaluations, the age of users is estimated to get insight into how this influences the interaction, which could later be extended into the product by changing behavior accordingly.

Towards a Robust Interactive and Learning Social Robot extends the capabilities of the NAOqi framework to facilitate more flexible and robust human-robot interaction, and empirically evaluates the created systems and characterize their strengths and weaknesses, similar to our research goal[18]. NAOqi's speech recognition is combined with Google Cloud Speech platform to increase accuracy and allow for general speech input. A test setup is described using a studio microphone to record robot-domain-related test sentences and play them back through a high-fidelity speaker located at the height of a human mouth facing pepper. Both systems listen simultaneously and publish their results.

This test setup for comparing the built-in and cloud system will be reused with our test on IBM Cloud's Speech recognition service. Researchers note that when users say something that is not in the vocabulary, the inbuilt speech recognition software sometimes still erroneously matches it to a phrase in the vocabulary with a too high confidence level to differentiate. Results show that the combined approach (combining built-in and cloud STT) yields a significant gain in accuracy relative to both the cloud-based and inbuilt speech recognition software individually.

The authors approach to use the four microphones differs from our research. To improve accuracy and help recognize speech from any angle, they separately stream audio from each differently positioned microphones to Google and use the recognition result with the highest confidence. In our case, a beamformer is used to improve the users audio signal. Going further than using Pepper hardware alone, the researchers add additional input modalities by using external stationary microphones and a phone application. Other modalities the researchers focus on is improving the vision of the robot by applying external (yet non-cloud) image analysis libraries as OpenPose, using ROS as architecture.

2.2. RQ2: Human-robot interaction quality

This section discusses how the use of cloud components can influence the quality of human-robot interaction. Interaction metrics, social robot implementations, and changes in interaction due to using the cloud services are examined. This is related to the question: *How does the use of Watson's versus Pepper's toolbox compare with respect to the quality of the human-robot interaction?* **Setting Up Pepper For Autonomous Navigation And Personalized Interaction With Users** implements a cloud-enabled ROS-based Pepper robot for the hospitality domain by connecting to the cloud services directly, making use of IBM Cloud Speech to Text[13]. The goal is to accomplish a more autonomous and personalized human-robot interaction, partly focusing on navigation. Face recognition is also performed remotely, but not in the cloud. No evaluation of the system was performed.

Four items of implementation were reused in our product. First, the authors illustrate that streaming audio, instead of saving it to a file and then sending it to the cloud service, reduces latency. (2) Another way to cope with latency is by making this process transparent to the user by showing a 'processing' icon in the interface. Although this does not directly remove the latency, it makes the users aware of the fact that the robot has heard their input and is currently processing it. (3) Whenever the robot hears or says something, the same text appears on the tablet and stays there until another utterance follows or 10 seconds have passed. This allows the user to follow the robot, even in noisy environments. (4) The problem of out-of-vocabulary words using Speech-to-Text, often happening with foreign names, can be corrected by the user on the tablet using a virtual keyboard.

Providing many examples for robot- and interaction metrics driven by a similar use-case as with our research is work by Pinillos and Marcos [19]: **Long-term assessment of a service robot in a hotel environment**. Here, a robot is constructed in several phases, where hardware and robot interaction is changed according to feedback from a previous phase. The use case for the robot is serving as a bellboy in a (real) hotel environment, providing information on the city and hotel and performing other hotel-related services. The robot is tested on over 1300 people and 57 hours of interaction over 74 days showing extensive research and many observations. To compare performance, a broad array of metrics is used, which are also useful for this thesis. Examples of the social metrics for interactions which are used are: Average interaction time, questions asked most, idle time (the robot is doing nothing except waiting for a user) and interactions against every hour of the day. Also, actions like question/answer pairs, input modality, and the topic are recorded, next to component specific metrics relevant for answering RQ1. Finally, useful tips for development of social robots are provided, such as testing the product in the same environment it is going to be used in.

We will reuse the designed performance metrics and data saving for our evaluations. The research also noted people did not always know what they can use the robot for, after which a suggestion list was added to the interface, next to an indicator for when speech recognition was active (the robot is listening) and showing the last spoken words of the robot. This will all be included in the tablet interface for our product. Observations in the paper made on users not being comfortable talking loudly to the robot in the presence of other people or not knowing how to talk to it are confirmed in our evaluations.

Empirical Results from Using a Comfort Level Device in Human-Robot Interaction Studies An important factor for a comfortable conversation with humans is the location (and displacement) of the robot[20]. Although experiments are performed in a living room situation, compared to the less personal restrictive office situation as in this thesis, and with a slightly larger robot, the experiments by Koay and Dautenhahn show that the majority of subjects feel discomfort when the robot blocked their path or when the robot was on a collision course towards them. This effect increased when the robot was in closer proximity than three meters -the social zone mainly reserved for human-human face to face conversation [21]- especially when performing a task. Relating to the hospitality robots in our case, this will be important with, for instance, human receptionists located near the robot. In our case, the robot will not move. A take away from the experiment to be reused in our evaluation is that the position of the robot should not be in the direct path from the door to reception, but aside from it.

R3D3: the Rolling Receptionist Robot with Double Dutch Dialogue by Linssen and Theune also focuses on natural human-robot conversation and interaction [22]. They started construction on a social robot (named R3D3) meant to function as host or receptionist for various establishments. Interaction is performed through the use of natural language and nonverbal behavior. As is the case with our project, R3D3 should understand what people are saying (the content) and why they are saying it (intent), which is still a challenge in human-robot interaction. This means more complex interaction is performed, breaking the ’simple commands only’ barrier. Added to this is synchronized non-verbal behavior.

This research differs from ours because two communication entities are used: a robot and a virtual human (displayed as virtual head on a tablet below the physical robot head). Our product is in the form of a humanoid Pepper robot and thus performing all actions as one embodied conversational entity. The authors conclude that the robot should take the initiative in conversations to fulfill its intentions.

They also noticed users are willing to cope with limited understanding when a sentence was misheard by the robot, by repeating important keywords of a previous utterance. We also saw this behavior in our evaluations. The cloud product we create for our second evaluation should also break the simple commands only barrier, including synchronized non-verbal behavior. Likewise, we combine this with automated speech recognition to analyze users’ intentions. If the robot does not understand something, it asks the user to repeat. The differences in approach with ours are using an existing type of robot, the use of the cloud and not using a virtual human as a communication method.

2.3. RQ3: Software engineering complexity

For the third research question implementations of similar robot (cloud) architectures are analyzed. Robots-as-a-Service and and other cloud robotics systems are included. This is related to the question: *How does software engineering complexity differ for an architecture using only built-in components versus one that also uses cloud components?*

Build a robotic calculations and inference agent presents a code pattern on how one integrates a NAO robot, Watson Assistant API, and Jupyter Notebook (Watson Studio) by using a Node-RED instance (a programming tool hosted on a server, capable of connecting to cloud services)[23]. This pattern demonstrates a scenario where the robot can answer queries on financial data by integrating with the IBM Watson Assistant service and IBM Watson Studio. This scenario is not relevant to ours, yet the implementation and workflow using Node-RED could offer a better alternative than connecting the robot directly with cloud services. After broader analysis, we decided the cloud-enabled robot will make use of this solution further described in Chapter 4 on architectures.

Towards a new approach of Robot as a Service (RaaS) in Cloud Computing paradigm proposes a Robot-as-a-service (RaaS) architecture [24]. The idea of Cloud Robotics, very much alike this thesis, revolves around robots outsourcing computing capabilities to the cloud, like communication, resources, storage, and computing power. Their solution is based on a service-oriented architecture paradigm (where each specific service communicates with each other) and uses ROS as robotic middleware, allowing robots to connect to the cloud and perform various tasks while being agnostic on robot features or hardware.

New skills are downloaded as needed; an executable plan as a service is retrieved from internal or external repositories in the cloud (if compatible with the particular robot). A virtual robotic layer is proposed, which contains the virtual robot systems (VRS). The VRS is a robotic operating system which is virtually executed in the cloud environment, controlling all applications of the robot. Our research uses a similar approach with Node-RED, yet lacking the ability to download skills from an internet source, but in principle, being robot agnostic.

Related to this are RoboEarth[25] and Rapytua[26]. **RoboEarth: A World Wide Web for Robots** was a research project, part of the Cognitive Systems and Robotics Initiative founded by the European Union, making a worldwide, open-source platform that allows any robot with a network connection to generate, share, and reuse data. It aimed to greatly speed up robot learning and adaptation in complex tasks. Robots using RoboEarth can execute tasks that were not explicitly planned for at design time. It provides a distributed database using OWL to store information like object descriptions, maps, and task specifications.

As part of its proof of concept, they also implement a generic, hardware-independent middle layer that provides various functionalities and communicates with robot-specific skills. A demonstrator was build showing the abilities of a World Wide Web for robots. In the demonstrator, robots were able to execute hardware-independent action recipes successfully and could autonomously improve their task performance throughout multiple iterations of execution and knowledge exchange using a simple learning algorithm. Note that processing, planning, and reasoning on this data still happens locally on the robot.

Rapyuta: The RoboEarth Cloud Engine implements this work, being an open-source Platform-as-a-Service (PaaS) framework designed specifically for robotics applications[26]. In PaaS, the cloud computing platform typically includes an operating system, an execution environment, a database, and a communication server. It allows offloading the processing, planning, and reasoning to a secured customizable computing environment in the cloud, providing direct high-bandwidth access to the RoboEarth database which is why its also called 'the RoboEarth Cloud Engine'. These computing environments are tightly interconnected, allowing robots to share services and information with other robots, making Rapyuta a useful platform for multi-robot deployments. The computing environment is set up to run any process that is a ROS node, and all processes within a single environment communicate with each other using the ROS interprocess communication and JSON otherwise.

The main robot target group is service robots (serving drinks, helping household tasks), having a similar highly-unstructured non-deterministic environment as in our research. Since we do not aim at grasping things using Pepper, or need to communicate with other robots, or need extensive world knowledge, and this cloud infrastructure does not aim at our specific needs, we will not use this infrastructure in our research. Relevant information to use is that with a Rapytua enabled robot, external communication (and not cloud-cloud) is the biggest constraint of Rapyuta's throughput.

Cloud robotics: architecture, challenges and applications discuss the technical challenges in computation, communications and security, and illustrate the potential benefits of cloud robotics in different applications [27]. As a part of this, they analyze complexities of various computing models involving robot peer to peer (machine to machine) and cloud infrastructure (machine to cloud) architectures. Three models are considered; a peer-based, a proxy-based and a clone-based model, each with their own complexity and robustness of collaborative-robot architecture. With peer-based, every robot has it's own cloud component, resulting in medium complexity and robustness. Proxy-based has a robot group leader connected to the cloud and other robots connect with the leader, having low robustness and a high complexity. With clone-based each robot has a corresponding system-level clone in the cloud: a task can be executed in the robot or in its clone, having high robustness and low complexity.

The use case introduced in the previous chapter is written for a single robot, but could also be used in a multi-robot setting. The research shows that robots running fully from the cloud instead of sometimes using built-in functionality have a lower architecture complexity. Our development experience partially agrees with this, as performing logic from the cloud only has a low complexity in the constructed system. Sensor processing however could benefit from being run locally without introducing more complexity.

2.4. RQ4: Pros and cons of cloud robotics for business value

The last research question is: *What are the up and downsides of using a cloud solution for robotics, when looking at business values?* Whilst there is literature available on the business value when transferring (general systems) to the cloud,[28, 29], relating this to social robotics is a novel approach that has not yet been investigated in literature. The general literature states using cloud infrastructure dramatically lowers the cost of entry for smaller firms trying to use innovative state of the art technologies available only to the largest of corporations.

Although not explicitly mentioned, we find this also holds for robotics, allowing the use of high performance algorithms and AI for sensor processing and logic on robots. The same holds for the following cloud advantages, which are to the best of our knowledge also valid for robotics: a faster time-to-market, reducing upfront costs, availability of remote management and configuration, security and regulatory issues are less prominent due to certificates of cloud service providers.

We make a first contribution in connecting robotics with the costs of cloud computation in relation to local processing. Other business advantages and disadvantages for cloud robotics are also discussed.

3
Use case & requirements

This chapter describes the use case for which the robot applications were developed. We describe the challenges with this use case for the robot. Based on the use case requirements are constructed.

3.1. Use case

A (Pepper) hospitality/receptionist robot will be developed during this project using the two toolboxes. For this, a use case has been created:

Pepper is strategically positioned outside the secured area of an office to help visitors when entering an office. She says hello to everyone who enters the building, and when a visitor approaches Pepper, she will ask whether the visitor is here for an appointment or needs any other help. Pepper asks the visitor to identify themselves or has already detected who is there using visual and facial recognition. Otherwise, speech recognition or the tablet can be used for identification. Pepper now searches its known calendars to check for an appointment with this visitor. If found, Pepper sends a message to the person who has an appointment with the visitor to let him/her know a guest has arrived. If no appointment is found, the visitor is directed to a nearby receptionist who can provide additional assistance.

3.2. Challenges

The requirements belonging to this use case can be found in Section 3.3. This use case offers a simple and widespread real-life scenario, but presents several challenging aspects for the robot:

Speech recognition Offices (in particular the reception area) usually contain several audio sources; people calling or talking with each other, the coffee machine, a television for visitors and even sounds of sliding doors. This is not optimal for speech recognition and often fails in crowded (and thus noisy) or echoing surroundings. For this reason, a tablet is used as a fallback option and should thus always reflect the same state represented by voice communication.

People (and face) detection and recognition Detecting whether a person is in an image is relatively easy to do, even with bad video quality. A person's skeleton can be distinguished even with bad lighting conditions since it is an easy comparable shape which does not often appear in other objects. A person's face, however, needs distinguishing elements to detect and cannot be seen if the face is completely black due to back-light on the camera. Only with enough light and contrast in the image, facial features can be extracted and used to recognize humans. Experience shows that this is a problem for the robot's camera. From a distance (>2m) this is even harder to perform, which makes timely drawing attention of an approaching user a challenge.

Placement and positioning The robot does not move from its position. This is to keep the scope of the research small (no mapping, navigation, or localization needed), and it is not needed for the use case. It must, however, be placed visibly yet not interfering with regular business in the office: what is the best position?

Expectations Creating correct expectations for the user on available functions such as using the tablet for immediate feedback, asking a question in a way resulting in the right needed answers, etc. Since the product to the same but differ in a way how: how to let the user know how it works knowing that most of them do this for the first time. This also involves reacting correctly according to all age ranges. Younger people often understand faster what the robot's capabilities are (and thus what you can do with it as a user) compared to the older generations. Instruction and (speed of) response can be personalized to support this.

3.3. General requirements of components

The requirements below are constructed in such a manner that they are valid for both developed solutions and selected to provide a balance between company needs and technical feasibility. They are represented according to the MOSCOW model, which is a method to prioritize requirements in the following categories: Must have, Should have, Could have, and Won't have. This way, both solutions have at least the core features in common and are likely to share the same functionality, but are not obliged to be the same. They have initially been set up by the author for the use case of the first phase, to test the workings of such a hospitality robot in the office of a Health Insurer. For these requirements, the following is assumed.

The location where Pepper is used should be flat (or the incline is less than 2%) and with office lighting conditions, minimizing intense light backlight. This is due to technical restrictions from Aldebaran built into Pepper. Not complying with this results in an error when booting the robot (and sometimes during operation), often resulting in the need for a reboot.

NAOqi cannot be entirely avoided for the solution build using Watson, as this is the base system for running and controlling the robot. However, complete Naoqi modules not handling low-level core functionality can be replaced by Watson modules (for example, face recognition or automatic speech recognition).

Also, a steady network/internet connection is available to Pepper. Watson modules are expected to be fast but rely heavily on a stable connection to work consistently.

Finally, Pepper only deals with single persons during an interaction. Interaction for groups would create a too broad scope for this research as interaction is different for groups. When small groups are encountered (<3 persons), one person is selected (the first person in range) for interaction.

Overall A general aim is that people interacting with Pepper should be able to perform all following modules using speech as the (main) input method and the tablet as fall back. Aside from this, alive movements by the robot should be displayed, which can be contextualized (to current interaction state, behavior, or previous user interactions).

People detection In order to engage users when walking in, people must be (autonomously) detected and drawn to the robot. When engagement is lost, it does not make sense to continue, and the robot should thus stop. If a person's face is recognized, personal information as name and appointment time do not need to be fetched from the user anymore.

Pepper must be able to:

- Detect people walking into a building when moving within three meters of the robot. Sensor data quality is too low to guarantee correct workings outside this zone, and this distance is also an acceptable distance for normal human-human communication [21].
- Attract the attention of a person
- Engage that person into an interaction.

Pepper should:

- Detect whether the interacting person is still present and engaged during the rest of the interaction. If the person has walked away, Pepper has to perform a reset of the functionality and start looking for new people again.

Pepper could:

- Recognise people's faces and perform a personalized interaction (taking into account previous conversations or other acquired personal data)

Client identification (and calendar) The main goal of the use case is finding and accepting appointments coupled to the right person. The robot should first tell users it is capable of doing that. If the robot cannot understand the user, it should forward it to the actual receptionists.

Pepper must be able to:

- Provide options to help, for example, to let an employee know a visitor has arrived or answer simple questions
- Direct visitors to the reception when functionality falls short.

Pepper should:

- Identify the visitor, retrieving name or other personal data (using either the tablet, speech or facial recognition).

Pepper could:

- Access a calendar (local or remote) to display available appointments and confirm these when asked to. If confirmed, Pepper lets the appointments employee know his/her visitor has arrived via some means of communication (SMS, email, call).

Assumption: List of appointments entered manually at the start of the day or available at some server.

Game As to entertain the user and keep him/her engaged when waiting, a little game is offered after all essential questions have been asked.

Pepper must be able to:

- Perform a game with a visitor (to improve awareness of the company or provide other information). Examples are:
  – A quiz
  – Helping AI products (clicking on the car in an image)
- Communicate and save the results of the game

Feedback Since we need data for comparison, the user needs to be asked for feedback on the interaction, which is now ending.

Pepper must be able to:

- Ask for feedback on the visitor's experience (for example displayed as five faces in varying happiness, forming a scale of 1 to 5.)

Data More data is needed for comparison and to know better how the robot is used.

Pepper must be able to save usage data like:

- Start and stop time of interaction
- Which module/component has been reached in the interaction flow
- Which modality is used to input data (tablet, speech)
- which score the client reached in the game
- Feedback score given by the client
- (other) Gathered customer information.

4 Architectures

This chapter describes the architecture of both the Watson and the NAOqi toolbox and other architectures which could be used as an alternative. This includes the software setup, how to connect with this as a programmer, languages, and way of programming. How does sensor processing work, and what is sent over the network and what not.

We start with describing the NAOqi toolbox of the first phase from our approach: the architecture, experiences during development, and alternative ways for development. Next, the second stage using the Watson toolbox is described in more detail. An overview of the newly created architecture called 'CASperSocket', services and servers used, how they connect, component-specific solutions to make the software work as intended and drawbacks on this setup. A description on Project Intu is given after this, which was the first option tried but did not meet all requirements for this project, hence the development of our CASperSocket product. The chapter concludes with the Lines of Code for both products for later comparison.

4.1. First phase: the NAOqi toolbox

In this phase, traditional development methods are used, and the cloud is not taken into account.

4.1.1. About NAOqi

The Pepper (and older but smaller robot model NAO) runs an operating system called NAOqi OS (based on Gentoo Linux)[1]. This OS powers the robots various functions, actuators, media systems, and everything else related to the robot. It provides and runs numbers of programs and libraries, among these, all the required one by NAOqi, the piece of software giving life to the robot.

One can communicate with the robot using a graphical programming tool called Choregraph, using the NAOqi API via various programming languages or an SSH terminal session. The latter allows for more direct OS access. However, users cannot use the root account on Pepper, while this is allowed on NAO. A wide range of applications for Pepper can be created using these tools. The supported languages for the SDK are Python, C++, Java, and Javascript, while a ROS connection is also available[2]. One can create and install 'behaviors' on the robot, a set of instructions for the robot to perform, which can be started and stopped as needed. Examples of behaviors are: wave to someone, drive one meter forward, do a handshake, or perform a little dialog.

NAOqi provides a set of modules to develop through, each with default methods for specific robot functionality. Table 4.1 shows the various modules and their function we use in the product. A complete overview can be found in the documentation[3].

4.1.2. Use case setup

The use case has been transformed into a general dialog structure. A similar structure (but used in the second evaluation) is displayed in Appendix B. For development, the Choregraph method is used as it was expected to be the easiest option, with the author having previous experience with this method. Larger projects in

[1]NAOqi OS documentation: `http://doc.aldebaran.com/2-5/dev/tools/opennao.html`
[2]Supported programming languages NAOqi SDK's: `http://doc.aldebaran.com/2-5/dev/programming_index.html`
[3]NAOqi API overview: `http://doc.aldebaran.com/2-5/NAOqi/index.html`

NAOqi API name	Relevant module(s)	General function
NAOqi Core	ALBehaviorManager, ALMemory, ALTabletService	Start and stop behaviors, save values to memory, access tablet functions
NAOqi Motion	ALMotion	Move the robot, specific joint control and reflexes.
NAOqi Interaction engines	ALAutonomousLife, ALDialog	Manage robot's awareness state, focus, and dialog functions
NAOqi Audio	ALSpeechRecognition, ALTextToSpeech, ALAnimatedSpeech	Perform speech recognition and generation, as well as (contextual) movement during speech
NAOqi Vision	ALPhotoCapture, ALMovementDetection	Take pictures and detect movement
NAOqi People Perception	ALFaceDetection, ALPeoplePerception, ALEngagementZones	Detect if a person is near, if one or several faces are visible, and how close people are to the robot.
NAOqi Sensors & LEDs	ALLeds, ALTactileGesture, ALTouch	Control the LEDs, detect if head, hand or other sensors are touched

Table 4.1: NAOqi API names, with the modules relevant for our product, including their general function.

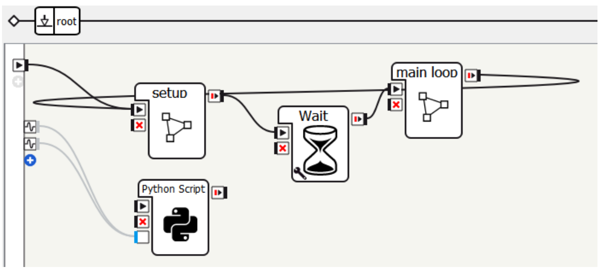


Figure 4.1: Choregraph view of main behavior. It shows four boxes performing initialization of variables needed later, loading the tablet with a company image and starting and restarting the main loop.

Choregraph can get unclear fast. To create more overview and ease shared development, the use case is split up in different behaviors: people detection -> client identification -> quiz -> feedback. One main behavior manages general logic: starting and stopping the sub behaviors as well as resetting and starting the whole interaction with one user. State (or box, or behavior) transitions depends on the received stimuli such as the presence of a user detected (motion/facial recognition, lasers), user speech recognized by ASR or tablet touch input. After a box is completed, it passes processing on to the next one.

Splitting the behavior in smaller chunks has more advantages such as easier testing, as it becomes more clear where errors occur, and more accessible version management as code is not saved linearly in Choregraph (more on this later).

How this looks in Choregraph is displayed in Figure 4.1. One can see so-called 'boxes'[4] with lines connecting them. By drag and dropping boxes with specific functionality and connecting them, a behavior can be created. Different layers can be created within these boxes. If we enter the 'main loop' layer, we see the content of Figure 4.2. The lack of structure with lines going in many directions is clearly visible, and for our scenario, six more similar layers exist.

4.1.3. Experience with development

Development with Choregraph is easy with a low learning curve. Including simple external code in the form of Python boxes did not create problems, which is used to read into external calendars, save data of the interaction, and to send emails to employees.

With more advanced scenario's and linking behaviors, some issues arise with consistency and developer tools which are inherent with the way of developing. We sort the problems per category: developer tools,

[4]Behaviors and boxes in Choregrapgh: http://doc.aldebaran.com/2-5/software/choregraphe/choregraphe_first_steps.html

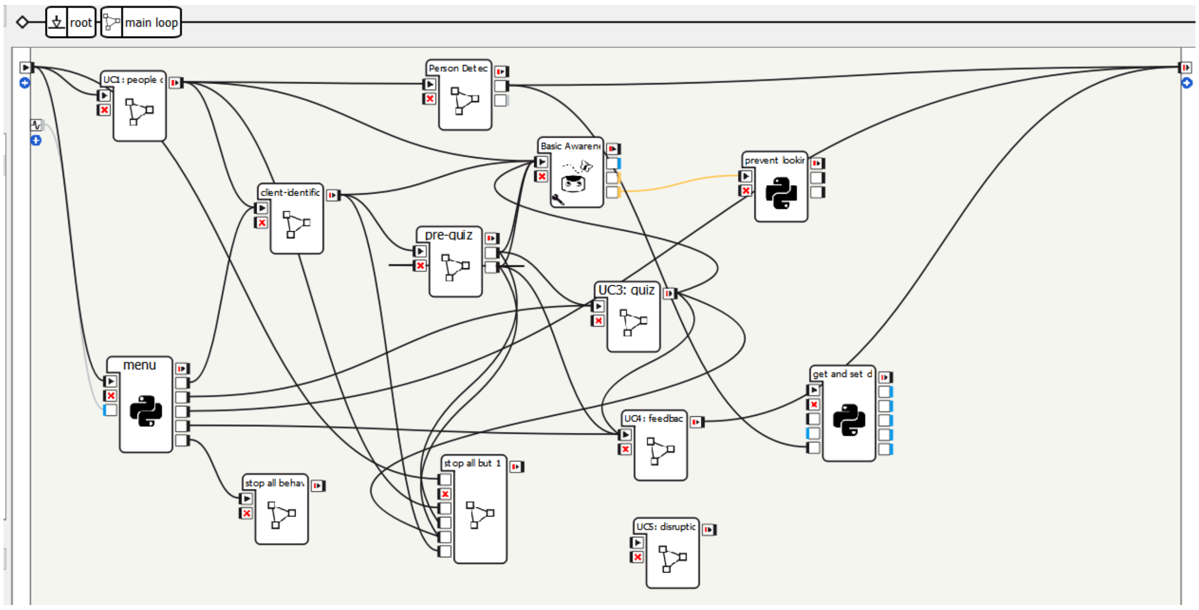


Figure 4.2: The content of the main loop box of Figure 4.1. This displays how the different behaviors are started and passed on, with data saving in between. The overview must be read from left (box initialization) to right (box completed), following the lines.

execution errors, hardware problems.

Developer tools The editor for code editing in Choregraph provides little gimmicks professional IDE's do have. There is no code completion, static code checking, error underlining, or debugging tools. Choregraph provides a simple interface for less experienced developers with drag & drop boxes from a library containing standard pre-defined functionalities. When one wants to use more features, Choregraph cannot provide in this, unless build in a Python box. Easy switching between in Choreograph developed behavior, and python code (written in the developer's favorite IDE) is not possible.

No easy debugger is provided for developers. Except for manual print statements, the log and a memory watcher you are unable to step through the execution or watch object values. Analyzing the robot's state is hard to perform.

Testing of behaviors containing specific boxes such as speech recognition and local file access cannot be performed on the virtual robot incorporated in Choregraph. This means most times the actual robot is needed to run a scenario, making testing a more tedious and challenging process.

Execution errors Speech boxes do not execute with full certainty as sometimes they are skipped or cut off. This seems to be caused by concurrent calls to the audio module, stopping current speech execution. The documentation, however, suggests that the box is fully executed and continues after that, which is not always the case.

After speech recognition was implemented, strange behavior started happening. Signals generated by user input, originating from speech or tablet, have to be sent to multiple boxes. However, when these signals are sent cannot be controlled. This leads to boxes receiving input when this is not expected or are not correctly initialized, resulting in unexpected behavior. This creates the need to always check for types and content of incoming signals and manually prevent code from executing using state variables. Using the Python API instead of Choregraph, this problem would not exist or be less influential.

Using built-in boxes, errors produced during execution of those boxes stop the complete behavior (and often stops the complete scenario). Errors are not caught, and execution can continue. While there are ways around this by self-implementing error catching the standard behavior does not guarantee a robust working application which continues or restarts after a small error.

Hardware issues With the feedback behavior, it was often noticed that the tablet does not respond well to touch. Sometimes a touch is not registered, and thus, one needs to touch multiple times. This module is tablet-only, without speech, since feedback can be given by humans in many ways which cannot be transcribed using the built-in speech recognizer.

With the identification of clients behavior, camera issues were found. Recognition, whether someone is standing in front of Pepper, is easy to implement using the people detection box. The timing and accuracy with which this happens in real scenarios are much harder to perform. This makes testing a long and tedious process since surroundings change all the time, such as the position of Pepper, lighting conditions, and the speed of users approaching Pepper. Testing was performed in-office with correct lighting conditions. However, at the client's location, Pepper would stand facing at the main entrance consisting of only glass. When the sun was shining, facial recognition rates dropped to zero since faces appeared black in the images returned by Pepper's camera sensor. Even with the window shades down and cloudy weather, facial recognition was far worse compared to testing. The advice mentioned in the relevant work from Pinillos [19] should have been followed here stating that tests should be performed in an environment similar to the environment in which the technology will be used in operations.

4.1.4. Code duplication

Code duplication happens in multiple ways when the product is created using Choregraph. When using the drag and drop functionality of dragging some function or boxes onto the flow, a new piece of python code is created in the background. One can edit the underlying code for the boxes. Every box requires initialization code for instantiating the inputs and outputs of the box itself, but also for modules used in the box. Often the same modules are used in different boxes, and thus this initialization can happen multiple times within one flow.

One flow is saved into one file. This file is a mix between XML and python, where the XML describes metadata on the boxes such as inputs and outputs, location, tooltips, various parameters, timeouts, etc. and has a content field for the actual python code. Often boxes have just one function, which means many lines of (initialization) code and accompanying XML for one actual code statement.

While this is just extra data to save this greatly influences the use of version control. Moving a box in the flow editor, for example, regenerates the code for this box in the file. While there is no change in function or visual flow, the code in the background changes significantly, and 10+ lines of code get deleted and inserted, as is reflected in a difference view between commits. As a developer looking at the newest changes in a branch, it is hard to spot what changed code and function wise.

By separating the behaviors in sub behaviors, we also introduced another problem: data saved in one behavior cannot be accessed by the next behavior. This means all data is saved in one behavior in a JSON file and opened again in another. Loading and saving correctly thus requires code to be duplicated in every behavior.

4.1.5. Alternate development methods

As mentioned before, the product could also have been developed using Python or one of the other languages available for the API. This way, standard IDE's can be used with all the benefits (debugging, code underlining and correction, testing suites), and programs can be created using a more classic approach. This also means version control works better as all XML code, duplication, and 30+ lines of code changes for small edits are not present anymore.

Using ROS would also have been an option, giving low-level access to sensor data. Since this is not one of the recommended development options by Softbank and more of a specialized toolbox for specific use cases, and no experience with ROS present in the development team, this option was not considered.

4.2. Second phase: the Watson toolbox

In this phase a product is built based on the same use case and requirements as the first phase. The components are run from or are connected to a cloud service which has the same basic functionalities as the NAOqi counterpart. There are several possibilities to implement the system using IBM cloud services. When looking into code samples from IBM cloud documentation and earlier related work from IBM (see Section 2.1), three options are likely to work.

The first option is to manually connect all IBM cloud services and build self-created logic and interfaces for information sharing between services, connecting, changing and configuring services and building a scenario The code could be built using, for example, Python to be able to run on the robot, or it could be hosted on a cloud server allowing python. Advantages of this option are a similar coding method than using NAOqi with an API and a single codebase. Disadvantages are long development times as no framework for this exists yet causing long development times, no visual coding options as with Choregraph and forced use of outdated Python 2.7.

The second option is to use Node-RED[5]. This can either be run on Pepper itself or in the IBM Cloud[6]. Node-red is an event-driven flow-based programming tool for wiring together hardware devices, APIs, and online services. It provides a browser-based editor that makes it easy to wire together flows using a wide range of nodes, including all IBM Cloud Watson services, that can be (re)deployed to its runtime in a single click. Among the long list of available nodes, one can implement a websocket, mqtt, databases, logic, functions, email, and social media interaction with a single drag and drop action. It allows for a similar coding experience as Choregraph for high-level functions but also allows low-level options.

Finally, Project Intu[7], a former IBM Watson service, could be used. Intu is middleware to be used on various form factors such as an avatar, robot, car, space (cognitive rooms, 'Watson in the walls') or other form factor having sensors and actuators, to route sensor's information to all kinds of Watson services. It has been open-sourced since July 2017. It is agnostic as to the specifics of the platform it is running on. An experimental version has also been built to work with Pepper and NAO. This would be the easiest to use option as the system is ready for the use case to be implemented and start comparing the workings and performance of both toolboxes.

As option number three seems most promising in terms of effort and ability to answer the research questions, this option has been explored first. However, Intu has been in slow development since it has been open-sourced, with little work in the last 18 months at the time of checking. The gateway on IBM Cloud has been taken offline at that point. Changes have been suggested to connect the cloud services without this gateway. The software works with these changes but not with full functionality and stability. Because the product seemed end-of-life, and a steep learning curve was needed to adapt and use the software, the decision was made to make create an application on our own using option 2 with Node-RED.

This section contains a description of the workings and schematics of this system, called CASperSocket after the name of our department (CAS), The name of our Pepper robot CASper and the extensive use of websockets. Further analysis and an explanation on the workings of Intu can be found in Subsection 4.2.8. Option one was not taken since it was estimated to be more effort, less generically usable afterward, and development less similar to the method used in the first phase.

4.2.1. Overview

CASperSocket is, partly like Intu, designed as middleware. A part of it runs on the robot, in this chapter referred to as *CASperSocket-robot*, and another part connecting to it in the cloud referred to as *CASperSocket-cloud*. The robot part performs little control logic by itself and is mainly meant to send the appropriate sensor data and setup proxies to lower level NAOqi functionality for physical robot control. The python-based system is designed to run on the robot itself, but if wanted a large part can also be run externally. There are a few components which must be run on the robot itself mainly because a system subprocess must be launched not related to NAOqi, which cannot be performed effectively via a remote connection. This is, for example, the case with microphone recording, done via a subprocess using the Linux application 'arecord'. This is the absolute minimum which has to run locally on the robot for speech recognition by Watson to work.

The software itself is built around an event-driven architecture, as after initialization the robot awaits commands from the cloud or some sensor input to start doing something. The initialization starts by setting up proxies to NAOqi API's and starting several managers. The first manager to start is the WebSocket Manager, as all other managers need this. This manager starts and keeps count of the websockets and closes and reinitialize them if needed. The main manager (start/stop behaviors/managers, access memory values) is started and as all other managers connected to CASperSocket-cloud. All managers also start a thread listening for incoming JSON messages. Next follow managers for audio, tablet, video, and helper, with the latter taking care of functions which do not belong elsewhere. Communication from the now running robot part only happens with JSON messages to the cloud part. Within the independent managers, one can switch from using the built-in method from the Pepper Toolbox or the cloud version from the Watson Toolbox. Other functionality is quickly added by making a new manager for it and adding it to the initialization. Coupling between the managers is kept to a minimum. This makes the architecture extensible, maintainable, and

[5]About Node-RED: https://nodered.org/
[6]Node-RED on IBM Cloud: https://console.bluemix.net/catalog/starters/Node-RED-starter
[7]Project Intu, about: https://www.ibm.com/blogs/watson/2016/11/experiment-embodied-cognition-project-intu/, and version control: https://github.com/Watson-intu/

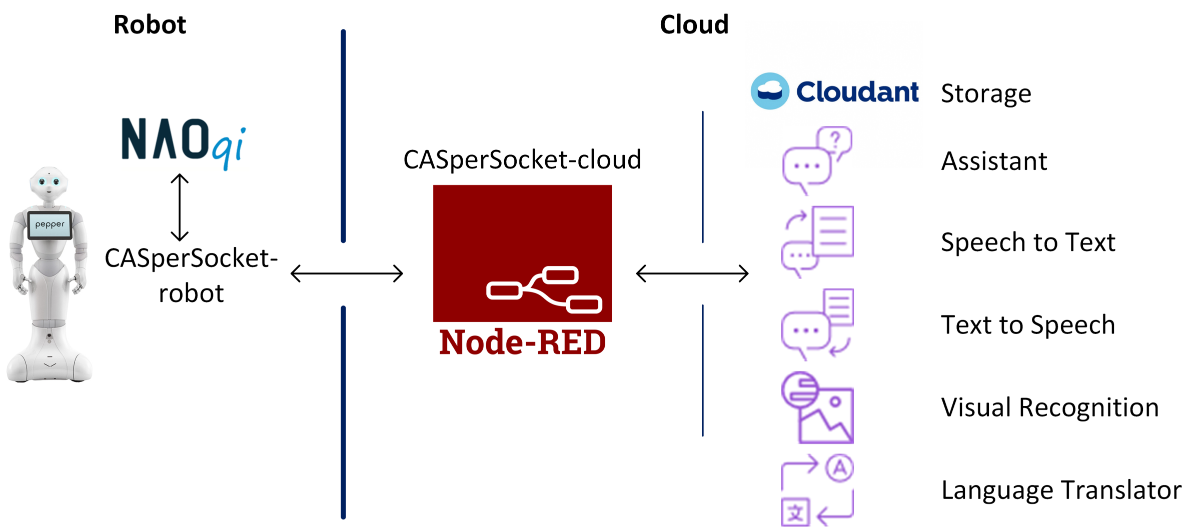


Figure 4.3: System overview of CASperSocket. CASperSocket-robot communicates with NAOqi and other system services on the robot using the managers, and connects to CASperSocket-cloud. CASperSocket-cloud is build with Node-RED, hosting the websockets and connections to the cloud services including storage (and all other external connections).

scaleable. CASperSocket-robot also runs on the NAO robot, except for the tablet components.

The part of code running in the cloud is based on Node-RED and perform logic, state control, connection with cloud services and storage, and any other external connection which might be needed. A basic schematic of the complete system can be seen in Figure 4.3.

4.2.2. CASperSocket

As stated in the overview of this chapter, Node-RED is used as a cloud component to connect with the CASperSocket-robot part running on the robot. First, Node-RED is introduced more extensively, followed by the ways of how to connect the robot's components with it. Finally, the drawbacks of this solution are discussed.

About Node-red Node-red is an open-source event-driven flow-based programming tool for wiring together hardware devices, APIs, and online services. It provides a browser-based editor that makes it easy to wire together flows using a wide range of functionality nodes, including all Watson services, that can be (re)deployed to its runtime in a single click.

Among the long list of available nodes, one can implement a websocket, mqtt, databases, logic, functions, data analysis, email, and social media interaction with a single drag and drop action. Nodes providing other functionality, such as the use of cloud services of other companies than IBM, can easily be imported from an online repository or created by oneself if not available. Node-RED allows for a similar experience compared with Pepper's Choregraph application for high-level robot functions, yet with more generic functionality and also allowing low-level options. Node-RED can be run on various platforms such as a Raspberry Pi, on Pepper itself, a web server or a laptop, which makes it a versatile tool. Its community is active, and the list of available nodes is long. An additional advantage to Node-RED is that a dashboard is included which can be generated with little configuration, allowing to control the nodes and show the output they give.

Comparing Choregraph and the editor of Node-RED (see Figure 4.4) one can see the basis is the same but the overview different. After a day of use, one is accustomed to the different colors for other functionalities, a grid, comment nodes, subflows able to connect with each other, virtual links and status messages creating a better overview of what the components mean and do. What is missing is the functionality to see visually where current data flows are happening, as is with Choregraph showing 'movement' on the lines linking boxes. This is, however, compensated by the use of debug nodes giving the data needed at that point. The same drag and drop functionality can be used, but not all nodes are robot specific.

The normal workflow is as follows: Create an empty flow (the canvas one is dragging to and executed as a whole), drag and drop nodes from the library (on the left) onto the canvas, link the nodes with wires and activate the flow by 'deploying'(right-top). One can (re)deploy this flow, only the changed nodes or all

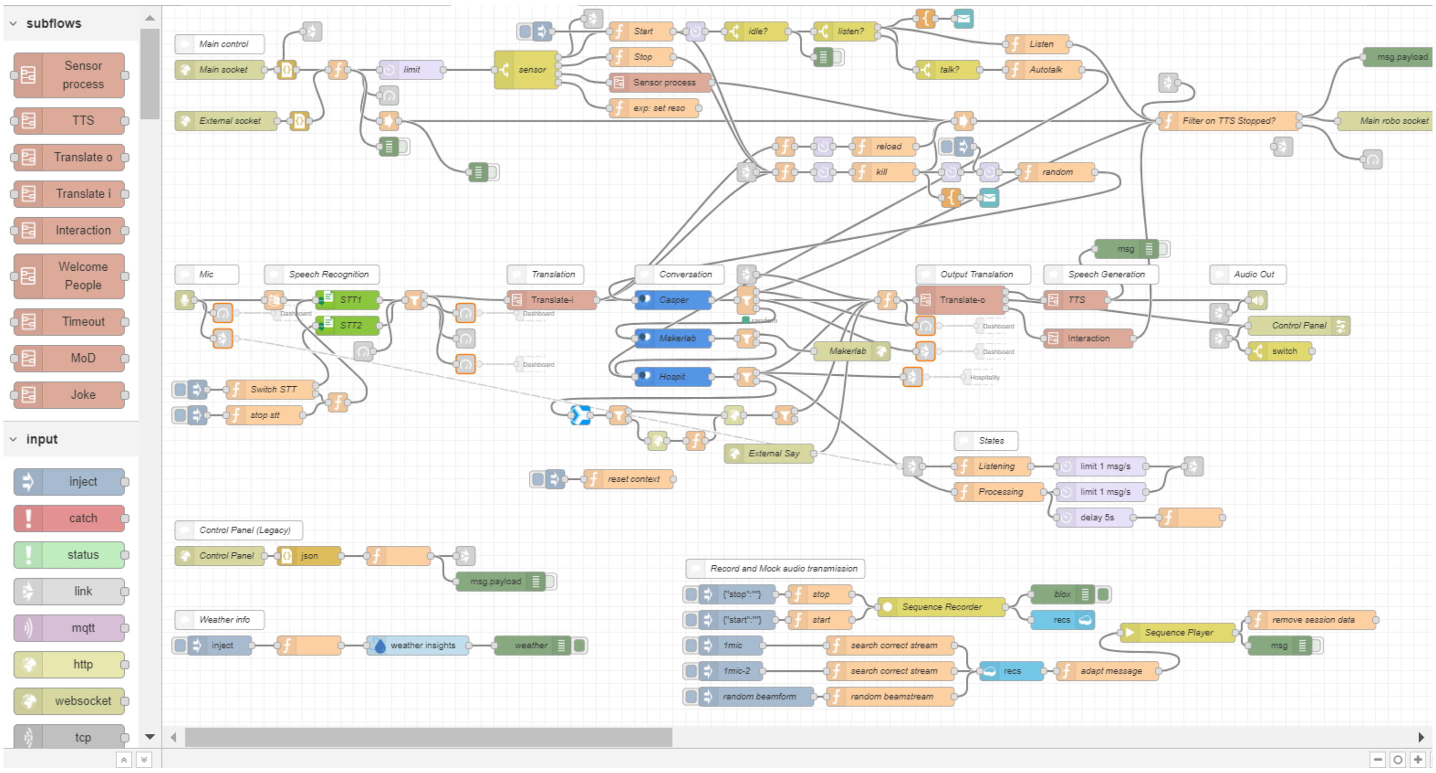


Figure 4.4: The main flow seen in the Node-RED editor. All nodes for logic, connection to services, connections to CASpersSocket-robot storage and alternative configurations are shown. Nodes are extra condensed (put close together) compared to the actual running system, to be able to show it in one overview.

flows at the same time, allowing for dynamic updating of functionality. Debugging the flow can be done by attaching debug nodes and reading the messages in the debug overview on the right. These messages can be filtered to only include nodes needed now. A group of nodes can be transferred to a subflow if the overview is lost. An example of this is the red 'Sensor process' node in the center-top, which processes all sensor-touch related events. Other colored nodes visible in Figure 4.4 are green for websockets, orange for Function nodes (running Javascript, but also Python or another language), blue for IBM Cloud Assistant dialogs and grey-blue for buttons to trigger events manually.

The dashboard module provides a set of nodes in Node-RED to quickly create a live data dashboard accessible via the internet. By placing dashboard-nodes in a flow, one can create buttons, text fields or other types of input, and display output on the same dashboard page again by dragging and dropping the layout, after which the page is automatically generated. This interface was used to control the robot's functions and to perform the evaluation. The interface for the tablet is also hosted this way.

Connecting Node-red with CASperSocket-robot With a Node-RED service available in the IBM Cloud it was decided to host it there. Node-RED can easily be configured to act as a server for several websocket connections. To provide modularity and prevent errors, one websocket is used per component. There is a websocket for Audio for recording output, Text-To-Speech results and Assistant (dialog) responses, a websocket for video/image transmission and Visual Recognition results, next to the main websocket controlling the activation and deactivation of CASperSocket modules, touch sensors, speech commands, behavior commands or referrals to custom functionalities not belonging in any specific module. An overview of which services from IBM Cloud can replace NAOqi functionality is shown in Table 4.2.

CASperSocket-robot can detect a close of a websocket by Node-RED, for example when re-deploying a flow to runtime, and tries to reinitialize it. This means one can change the workings of the system online while not having to change the code on the robot or even restart the running application. A use case for this could be changing the Assistant workspace (containing the dialog) dynamically without having to put the robot out of service. The same holds for changing the Text-To-Speech engine, responses on sensor input or any other functionality supported by the code already running on the robot.

NAOqi API name	Relevant module(s)	IBM Cloud service
NAOqi Core	ALBehaviorManager, ALMemory, ALTabletService	Node-RED Cloudant
NAOqi Interaction engines	ALAutonomousLife, ALDialog	Assistant Language Translator Wheather insights Natural Language Understanding
NAOqi Audio	ALSpeechRecognition, ALTextToSpeech, ALAnimatedSpeech	Speech-to-Text Text-to-Speech
NAOqi Vision	ALPhotoCapture, ALMovementDetection	Visual Recognition
NAOqi People Perception	ALFaceDetection, ALPeoplePerception, ALEngagementZones	Visual Recognition

Table 4.2: NAOqi API names, with the modules relevant for our product. The IBM Cloud alternatives replacing most, or, more, functionality from the NAOqi counterpart.

Data only has to be sent once to Node-RED. After receiving, the results or data can be passed on internally to several services simultaneously. For a video stream, this means that one image can be sent to various recognition services to get different results (face recognition and object detection, while also searching for barcodes, for example). The end result passed back over a websocket to the CASperSocket-robot is only one JSON message containing results or a command for the robot to perform.

People can use the Node-RED tool without having much technical expertise because of the simple interface. However, developers with more knowledge are offered the freedom to build complex systems. Debugging can be performed online as well to ease development.

4.2.3. Audio, connecting Microphones

To circumvent the usual way NAOqi allows access to the microphones[8] (only recording to file or direct Speech to text) we use the Linux tool *arecord*[9]. This utility allows the readout of the microphone data buffers while not hindering existing robot functionality. These buffers can then be streamed to CASperSocket-cloud for further processing, such as Speech to Text or other audio feature extraction. Streaming is performed using small buffers as to keep latency low: saving a file of audio recording, as is possible using NAOqi, would not allow for near real-time transcription.

As a social robot, Pepper will likely be exposed to a variety of social interactions, and users can express commands, queries, or answers in a large number of ways. As opposed to the keyword-based built-in speech recognizer where a recognition vocabulary needs to be defined a priori, the Speech-to-Text service allows for general speech (also called free-speech), providing confidence levels, alternatives, and partial results while processing.

The pipeline in CASperSocket-cloud is as follows (See Figure 4.5): receive audio buffers in websocket -> send data to Speech to Text service -> process and convert JSON with transcription and/or partial results for internal use -> optionally translate into another language -> send result to Assistant -> optionally translate again -> optionally generate speech with Watson Text to Speech -> send speech data or text to say by NAOqi TTS back over websocket. If TTS is used from the NAOqi toolbox, the robot performs moving behaviors contextualized to the input by default, or otherwise no movement or a specific behavior can be executed during talking, as to make the interaction more alive and natural.

Pepper has four omnidirectional microphones in an array mounted on the top of the head[10]. For sending audio to the cloud, there are two options: Send one channel (from one microphone) through CASperSocket-cloud to the Speech to Text service, or send more (two or four) channels. The latter is not supported by the

[8]With an educational license from Softbank, free-speech cloud Speech to Text from Nuance is also available on Pepper in English, Chinese and Japanese. Since we missed this license, this could not be tested. `http://doc.aldebaran.com/2-5/family/pepper_technical/languages_pep.html`

[9]arecord sound recorder: `https://linux.die.net/man/1/arecord`

[10]Four microphone array Pepper: `http://doc.aldebaran.com/2-5/family/pepper_technical/microphone_pep.html`

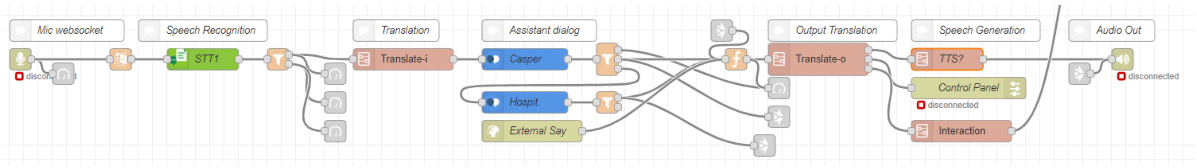


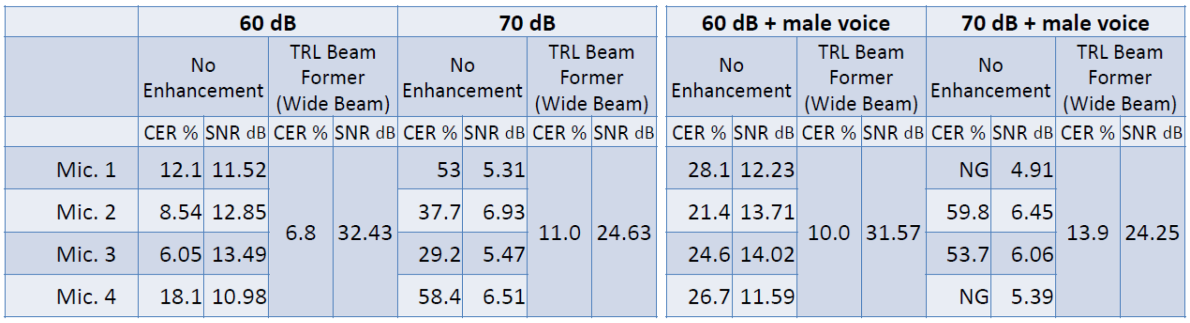
Figure 4.5: Audio pipeline in Node-RED. From audio buffers over a websocket, to a transcription, response by Assistant and optional TTS generation at the end. Other connectors are present to show the (partial) result or certainty on the tablet or generate alternative contextualized movement of the robot based on the input text.

cloud as only one channel is processed and must thus first be reduced to one channel.

While the first, one channel, option was used initially, giving accurate results in a silent environment, this was not the case in noisy surroundings as can be read later in Subsection 6.6.2. Transcribing speech when there was none, transcribing wrong words, no words, and low accuracy, in general, was the case. To improve on this, a beamformer is used, which can improve the signal-to-noise ratio using the four microphones.

Beamformer Background noise can greatly degrade Speech-to-Text performance. The noise comes from Pepper's own motors/movements and background noise from the environment of a reception: sliding doors/-gates, colleagues talking a few meters away, the coffee machines, discussions at reception, etc. High-quality noise reduction is needed to be able to understand the user with speech in this environment. A beamformer using the four microphones can do this. Many techniques exist to perform beamforming, and all are based on principles of phase shift and wave propagation: with milliseconds of time difference between receiving the sound waves in one microphone and another placed some distance further (a few centimeters in Peppers case), a single higher or lower amplitude wave can be created by superimposing the waves on each other. In our case, the goal is only to hear the interacting user in front of the robot.

At the moment of writing, no public Pepper-specific beamformer was available. For CASperSocket-robot, a non-public beamformer created by IBM Research Japan is used. It was developed early 2016 and now in an experimental phase, lacking more than basic documentation in the form of performance slides which can be seen in Figure 4.6, promising high performance in increasing the signal to noise ratio. The use of it requires changing driver settings on the Pepper robot, lowering the microphones input gain and volume settings.



	60 dB				70 dB				60 dB + male voice				70 dB + male voice			
	No Enhancement		TRL Beam Former (Wide Beam)		No Enhancement		TRL Beam Former (Wide Beam)		No Enhancement		TRL Beam Former (Wide Beam)		No Enhancement		TRL Beam Former (Wide Beam)	
	CER %	SNR dB	CER %	SNR dB	CER %	SNR dB	CER %	SNR dB	CER %	SNR dB	CER %	SNR dB	CER %	SNR dB	CER %	SNR dB
Mic. 1	12.1	11.52	6.8	32.43	53	5.31	11.0	24.63	28.1	12.23	10.0	31.57	NG	4.91	13.9	24.25
Mic. 2	8.54	12.85			37.7	6.93			21.4	13.71			59.8	6.45		
Mic. 3	6.05	13.49			29.2	5.47			24.6	14.02			53.7	6.06		
Mic. 4	18.1	10.98			58.4	6.51			26.7	11.59			NG	5.39		

Figure 4.6: Performance of beamformer used in the CASperSocket in noisy environments as provided by documentation. CER = Character Error Rate (lower is better), SNR is the Signal to Noise Ratio (higher is better), NG means Speech recognition did not return any recognized character. The table shows the use of the beamformer provides better results in almost all cases.

These values and other configuration settings have been determined by exploratory searching for the best values by listening to the output while noise is present in the background and adjusting to debug output from the beamformer. From the configuration, one can choose focus region, use of delay-sum or MVDR beamforming, binary masking, (local peak) weighted CSP, post filters and suppressors and de-reverberation, supectrum and flooring factors. We use the MVDR (Minimum Variance Distortionless Response) variant, as this performed best in our tests and outperforms delay and sum beamformers [30], with binary masking and default values for all other options. The focus region is set to 90 degrees, where 360, 180, and 45 degrees are also an option. This means the beamformer finds a target speaker within the 90 degrees and enhances the speech, suppressing sounds from the sides and back. Figure 4.7 illustrates how this works.

A beamformer attenuates background noise by enhancing sound components coming from a direction specified by a steering vector. The robust MVDR beamformer nulls signals coming from any direction other

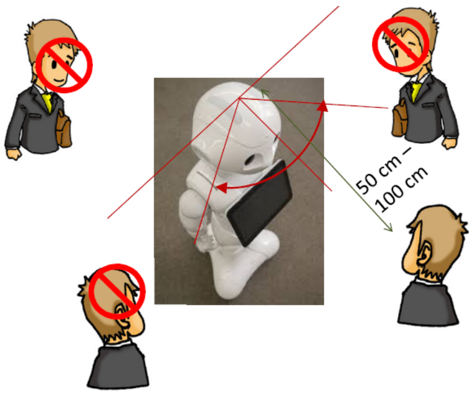


Figure 4.7: Overview image of beamformer working: Sound produced by the person right in front of Pepper (in a 90 degrees angle) is in the audio signal, sound produced by the others will be filtered away.

than the look direction specified by the steering vector [31], in our case where the robot's head is looking to and where we expect a user to be. Determining the steering vector is included in the program.

The program itself is an executable file which requires raw audio data as input (PCM format, sampled at 8000Hz per channel) and outputs single-channel raw audio, which can be sent to the cloud Speech to Text service. It can either run on the robot or in the cloud, yet our implementation did not entirely work in the cloud for unknown reasons. There is little CPU power (an increase of 50% on one out of four cores) needed to perform this on the robot. A configuration switch has been built into CASperSocket-robot to either use the 1mic or 4-mic with beamformer pipeline. The beamformer introduces about 300ms of delay in the total pipeline from recording to transcription. Whether Speech-to-text from the NAOqi toolbox performs beamforming is not documented, but several unofficial comments from employees on forums indicate that it is. Next to that, it would be illogical to build a robot with four microphones and not use them for such a purpose, although this can also only be used for sound source localization to be able to detect where users are to look at.

4.2.4. Cameras

Pepper has two normal 2d cameras and one 3d sensor located in the eyes facing forward and in the mouth facing downwards. The 2d cameras provide a resolution up to 2560*1920 at one frame per second (fps) or 640*480 at 30 fps. The main eye camera is used in our system to determine whether someone is standing in front of the robot, next to gathering data as the number of visible faces, gender and age estimation, face location and size using the Visual Recognition service. The camera could be used for other things which are image recognition related such as detecting objects, describing what is in the image, pose detection, etc.

The general pipeline is as follows: make a picture -> compress it -> send it to CASperSocket-cloud -> send to Visual Recognition -> do something with result in CASperSocket-cloud. A NAOqi-API call which takes images is used which also encodes it in JPEG format. Other options, such as recording a video, are described in Subsection 7.3.1. Visual recognition is implemented by connecting the video websocket to the Visual Recognition node, and select to look for faces, as can be seen in Figure 4.8. The output of the pipeline can be used by scenario's, for example, to detect approaching people and activate a scenario or measuring if they have not left.

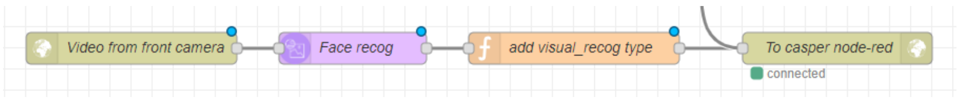


Figure 4.8: Node-RED implementation for Visual Recognition with an incoming and outgoing websocket, Visual recognition service, and a function box adding the message type to the outgoing JSON message.

Engagement For our use case, we use the pipeline at 2 FPS in the second-highest resolution of 1280*960px. This allows the robot to classify people from a maximum distance of 5-10 meter, depending on lighting conditions. The robot tries to draw attention (by doing a small wave) as soon as one or more faces are in sight (the robot's position was slightly off facing the entrance door on purpose). This activates a built-in NAOqi function (FullyEngaged[11]) tracking the face in sight by aiming gaze at that person. This could have been implemented using the position returned by Visual Recognition, but if users do not make sudden moves, the NAOqi face tracking works well and quick. Although it often loses recognition of the face with its built-in recognition for a short time (visible by watching the robot's memory values), the cloud Visual Recognition service still has the person in sight preventing the robot from looking for other people. This happens until a user does move away, which will let the robot look around again if a user is gone for seven seconds (controlled by Visual Recognition) and reset the scenario for a new user (UnEngaged mode).

The two toolboxes working in a collaborative manner for the same functionality is an excellent example of how using utilizing the best properties of both, in this case, low-latency gaze behavior and accurate classification, outperforms the use of a single toolkit, resulting in a better working system as a whole.

4.2.5. Storage

IBM Cloudant (NoSQL) is a scalable, schemaless, and fault-tolerant JSON database built for the web. It allows one to start small and scale up easily, with support for replication and clustering. This document store could be used for all storage needs of the robot, as for example user data, configuration, pictures (think of scanning a name badge) or any other data storage use case. The same holds for keeping the user's state of conversation or data from previous interactions with this person.

Cloudant can easily be implemented in Node-red by dragging the Cloudant node on the workflow and setting a name for the database to use. After this, the connection is established, and the database can be used directly without further configuration.

Recording input for testing One advantage of Node-RED is the ability of a node allowing timed recording of data on any place and node in the flow. By recording the sensor input coming into Node-RED (audio buffers, images, NAOqi memory values) to Cloudant, including a timestamp when this message arrived, one can replay this whole activity back as if it is happening live. Reproducing bugs or performing automated testing is made possible this way and greatly decreasing developer or testing effort.

4.2.6. Scenario and dialog

All previous components make up a framework, similar to NAOqi. All components output JSON messages with the processed results and a message type. One can now start creating a scenario (the point where development using Choregraph normally starts) by creating a subflow and connecting the needed service output results with its connector. How these are processed and how the scenario is built is up to the developer.

In our case, we use Watson Assistant for performing a dialog, which can also be used to perform state management. With every response from Assistant (a sentence to say to the user), a string containing the state is also included in the JSON message. This is received by a function box processing all inputs of the scenario, as can be seen in Figure 4.9. The 'State keep' function performs the logic of the scenario, based on the state input of Assistant. The State keep function has two other inputs: (1) the number of people currently in sight, used to activate and optionally stop the scenario, and start Speech Recognition which activates the Assistant and (2) manual inputs to make testing easier. All test inputs not actually needed for running the scenario are located left down and have an orange border in Figure 4.9.

While the interaction could work already with this setup, we want the robot to do more such as displaying text, suggestions or buttons on the tablet in certain states, perform a wave movement when the robot says hi or notify someone of the visitor's arrival by sending his or her data in some way. All boxes to the right side of the state keep function in Figure 4.9 contain code for specific states where something else than talking needs to happen. This could have been put in the same function box but is split for a better overview. All lines enter in virtual links back to the main flow, tablet, or dashboard.

All light blue boxes relate to showing data on the dashboard or input from a button on the dashboard. Dark blue boxes save interaction data for our evaluation.

Assistant and dialog A schematic overview of the dialog can be found in Appendix B. This is translated into a 'Dialog Skill' in Watson Assistant. Dialog skills use Watson natural language processing and machine learning technologies to understand user requests and respond appropriately. How this is done is also presented in Appendix B. One must define the possible intents and entities and give examples for the system to learn,

[11]Engagement modes: http://doc.aldebaran.com/2-4/NAOqi/interaction/autonomousabilities/albasicawareness.html#albasicawareness-engagement-modes

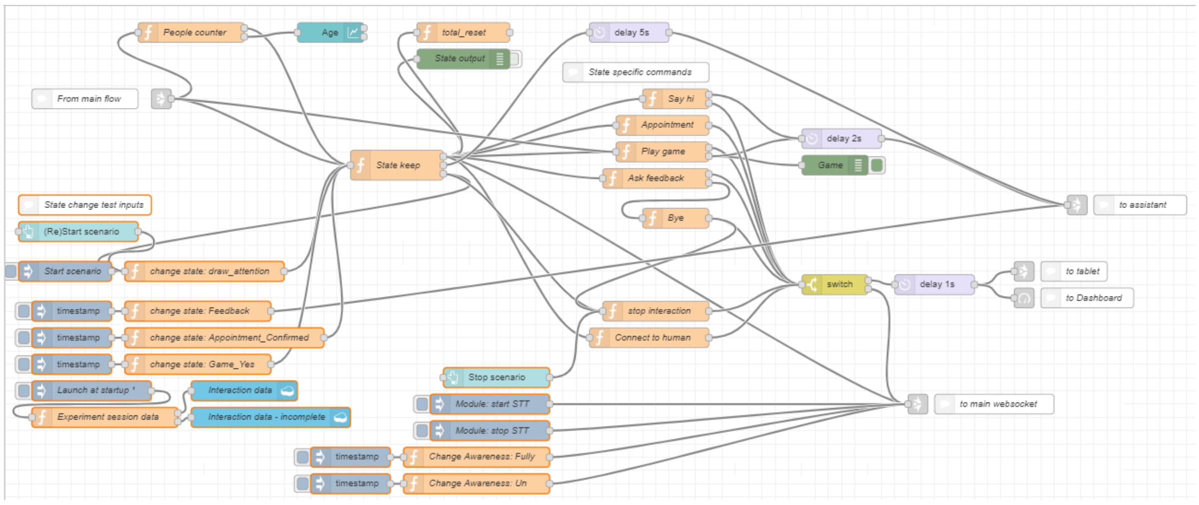


Figure 4.9: The full scenario of the hospitality use case as constructed in Node-RED. All nodes in left bottom, surrounded by an orange border, are testing related.

next to learning from use and corrections made by the developer over time. One can now construct the dialog using the intents and entities. Common collections and examples can be inserted into the skill with a single click to get a helping start.

Every response from the Assistant contains a textual response (what the robot can say) and a context object to track the dialog. This context object needs to be included in the following API call to continue on that conversation. The context object contains previous steps in the conversation, the currently determined intent, and known entities with their values, the 'state' we inject ourselves and metrics on the current conversation.

There are several advantages of using Assistant over the build-in dialog manager of the NAOqi toolbox (using ALDialog). The ease of use increases by providing a simple interface and dashboard to build and improve dialogs. The clickable interfaces are well suited to non-programmers. It has built-in learning capabilities, and if the system wrongly classifies a user utterance, this can be corrected, preventing it happening in future conversations. Its context-aware algorithm performs automatic filtering of intents, entities, numbers, dates, names, etc. The dialog can be changed while being in use and delivers real-time and historical statistics on the conversations. Costs of using it are minimal. Comparing to ALDialog, dialogs are harder to create by having only one file where the dialog is contained, resulting in a lack of overview for larger dialogs. Building the dialog requires a low level of programmer experience. Jumping between conversation-nodes with extracted data is not possible, and intent detection is keyword-based.

Interface Node-RED also hosts the tablet interface, yet not created with the dashboard functionality(the general robot's website and control panel, is). Figure 4.10 and Figure 4.11 show the same interface in different states of the conversation with various display option informing the user of the robot's state. This design has been based on research from Perera[13].

Figure 4.10 shows the tablet interface with the text spoken by the robot, a microphone sign indicating the robot is listening, a list of (clickable) suggestions for the user to say (right-top), an entry bar for text input using virtual keyboard (bottom) and an indication of the interaction progress (empty as this is the start).

Figure 4.11 shows the spoken text by the robot (right-aligned) and user (left-aligned). It also shows clickable buttons for hints on how the user could answer the robot's current question. The progress bar shows the current step of confirming the appointment. Since we do not want the user to digress from this question now, no new suggestions to say are shown. Next to the user's text, a small green number indicates the confidence level of the speech transcription, mainly meant for the observer.

4.2.7. Drawbacks of Node-RED

The developed CASperSocket allows for every functionality needed by this research. There are some drawbacks to using Node-RED compared to a product running only functionality of the Pepper Toolbox or connecting directly to cloud services. To start, the stability of Node-RED is varying: during development, it reg-

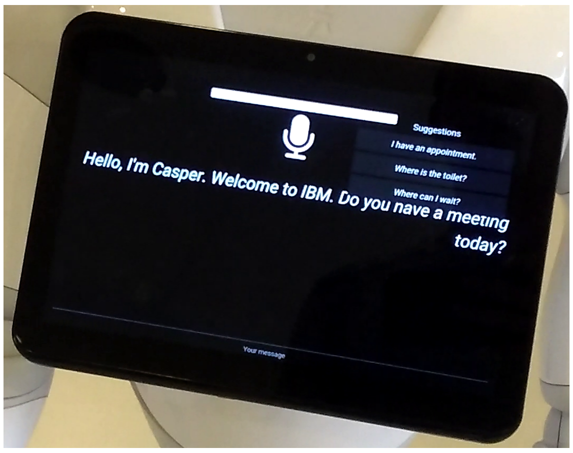


Figure 4.10: Tablet interface showing spoken text, listening indication and suggestions.

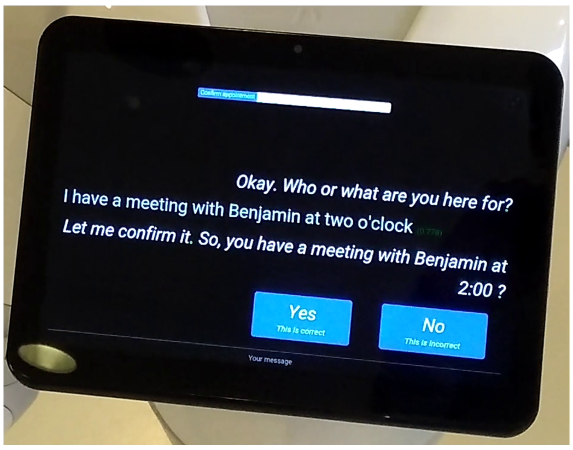

Okay. Who or what are you here for?
I have a meeting with Benjamin at two o'clock
Let me confirm it. So, you have a meeting with Benjamin at 2:00 ?
Yes
This is correct
No
This is incorrect
Your message

Figure 4.11: Tablet interface showing spoken and processed speech transcription of user, buttons, and progress bar.

ularly crashed when starting a high-bandwidth data transfer or disconnecting from websockets. With development done it does not crash anymore, yet dynamically changing functionality may result in a crash. The impact is low as the downtime is on average 30 seconds and it restarts automatically.

Next, as can be seen in the images in previous subsections (see for example Figure 4.4), the overview for the developer did not improve much compared to the Choregraph application. Spending some time with Node-RED makes the eye trained with seeing quickly how it works, but for beginners, such a system is still not clear. Making separate nodes (sub-flows) for components from the existing codebase can help, as well as general restructuring.

Access management and version control is present in Node-RED, yet not visible and easily configurable. Git is used in the background for version control, but this is not directly visible for the user as well as other supported features. Deleting a flow is too easy for general users, as it does not ask for confirmation and there is no restore button. The interface is mostly suited for beginners, but these items make it a more dangerous place to experiment in.

Finally, simultaneous multi-user mode is supported, but not well implemented as updates can only happen on one machine at a time. Other users can throw away another developers work with a single click if not careful.

4.2.8. Alternative: Intu

Project Intu[12] is a former experimental IBM service, open sourced since July 2017. The software is designed as middleware to be used on various form factors such as an avatar, robot, car, space (cognitive rooms, 'Watson in the walls') or other form factor having sensors and (some) actuators, to route sensor's information to all kinds of IBM Cloud services. Initially, this software platform, also available for Pepper and NAO, was used as cloud component for our experiments.

The Intu system is in the documentation often referred to as the cognitive architecture 'Self' to evoke a sense of self-identity. It is agnostic as to the specifics of the platform it is running on, and in that sense, one can think of Self as attending to higher-level cognitive functions that build on device drivers with implementations specific to a particular manifestation, such as Pepper.

The project was primarily focused on use cases which are associated with augmented intelligence, that is, systems that expand individual and collaborative human senses and abilities, systems that work in cooperation and conversation with humans but that are also capable of degrees of self-agency and self-understanding. Example use cases mentioned in the documentation are retail, elder care, manufacturing or transportation - assistant, a concierge, cognitive room, or cognitive companion. Self is meant to augment human capabilities. Safety, privacy, and security are 'baked in'.

At the time of writing, user activity on the accompanying GitHub page is low (no commit has been made to any branch since four months), and thus the software is outdated on certain points. This was one of the reasons to not use this software for our evaluations.

[12]Intu Github: https://github.com/watson-intu/

How does Intu work The internal setup of Intu is shown in Figure 4.12. Four subsystems encompass Intu: agency, perceptions, and sensors, actuators, models. They can reflect on a short and long-term state. Com-

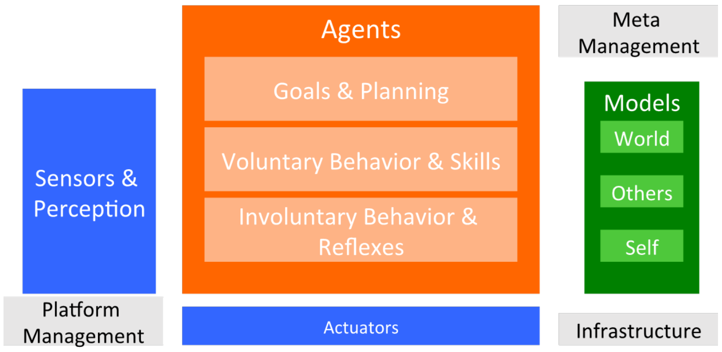


Figure 4.12: Overview of Intu architecture

ponentization is important, Self uses one backbone. Self is based on the following design decisions and principles[13]:

- Self is a hybrid architecture, encompassing explicit symbolic computation in the center with neural networks at the edge The pragmatic reason for this is that there are limits to AI, especially making it reflective: why did you make this decision? This knowledge is better available in symbolic systems. Perception is better done in neural networks (edge). Self combines these two computational models.
- Inspired by Minsky's society of mind, behavior in Self takes place in the context of multiple concurrent agents communicating opportunistically via blackboards.
- Inspired by Brooks' subsumption architectures, behavior in Self takes place in a hierarchy of cognition, from involuntary reflexes to voluntary skills, to agents carrying out plans to achieve explicit goals.
- There is a clear separation of concerns among perception, actuating, models, and behavior.
- As much as possible, a behavior is either taught or is learned, not programmed.
- As much as possible – driven by these separations of concerns, the needs of packaging, and performance – all components are made manifest as RESTful microservices.
- As much as possible, plans, skills, and reflexes are extensible. Extensibility is made using a pervasive plugin mechanism.
- Self is intentionally full of strange loops: components of Self are also parts of the models of itself.
- Self is intentionally fractal: an instance of Self may have models of others, which themselves are other instances of Self.

With the aforementioned form factors in mind, it should be noted that an instance of Self transcends each of the forms itself: logically one can think of Self as single entity, but physically, Self is deployed as a small kernel that resides in that robot/avatar/space/device with all other elements of Self residing in the cloud, delivered as cooperative constellation of microservices. Agents can look inside the model of others: "Not only this is a human, I know who it is".

Not using Intu for this research In terms of our research questions, Intu offers several advantages. Little software needs to be developed as it is already made and only needs to be extended for our specific use case and tests. Right after installation, it connects Pepper's sensors and logic control to the cloud where data can be exchanged, and IBM Cloud services can be used with no effort. A tablet interface is also provided as well as an extensive dialog to talk to the robot and give commands and ask questions. Connecting Intu with other sensors such as other microphones or cameras in the room can be done with little effort. The documentation suggests it could work faster than using Node-RED with various optimizations in place. The ideology of Intu

[13]Based on personal communication with the original designer/developer IBM Fellow Grady Booch and the following Youtube video where he discusses Intu: https://www.youtube.com/watch?v=ADnCUgrEt0U&list=PLZDyxLlNKRY8l5r7sbYbmtB-7N0PwPEzD&index=2

works in our use case' setting, as this software was used before in a Hilton hotel in America using the NAO robot.

Disadvantages are also present, such as less configurability. Although it is open source and can be changed as wanted, there are clear code patterns present and going in against that often not works out. There is a steep learning curve since documentation is outdated, not consistent with the code implementation, not all is publicly available, and with no clear overview as it is spread over multiple places. The Intu gateway, an essential cloud part for Intu, is offline and deprecated an will not be replaced according to the main developer. A hotfix is implemented to circumvent this, but no permanent solution is being made at the time of writing. Next, compiling the code is not possible on every platform (only on Mac). Every little change requires recompilation and reinstallation of the system, which is not a pleasant workflow for experimental research. Finally, creating a full scenario as intended by our use case is possible, yet not what Intu is designed for and thus more effort to build. Intu is designed for small behaviors, responding to sensor input events. Starting a gesture built in the NAOqi system is not directly possible, as well as other standard options, as the API for Pepper is not complete. The Intu architecture makes performing specific tests (for quality) hard to do since there are all kinds of processing chains in between which cannot be circumvented.

With the slow adoption of the open-source community and tight coupling with fast-changing cloud services, it is expected that this software can stop to work in the near feature. With the author having little experience with C and it being a large project already, it is expected that needed changes of the software might take more effort than building a custom application completely fit to our requirements. Together with the aforementioned disadvantages, the decision was taken to build CASperSocket.

4.3. Lines of Code

Various parametric developer effort/cost estimation models exist. One metric often used in mainstream models is the LOC (Lines of Code) because it is highly correlated with effort[32]. Variants of LOC giving more precision are SLOC, looking only at Source code lines skipping comments and other non-executable text, ESLOC (Equivalent SLOC), taking into account new and adapted (modified, reuse, generated) code, and KESLOC; Thousand Equivalent Source Lines of Code.

As this metric gives meaningful insight into the complexity, maintainability and effort of the two software products, it is compared in Table 4.3. Although SLOC is shown in the table for the robot's python code, there is no tooling available to calculate SLOC for the cloud and Choregraph counterpart. Because of this, the comparison is made using LOC, which is less precise but still indicative. From the Choregraph product, all XML code (see Subsection 4.1.4 on the use of XML) is removed before determining LOC since this is no source code but only used for styling.

To get the total LOC for the CASperSocket, the LOC (total lines) of the 'on robot' and 'on cloud' columns are added to get to a total of 2832. This is significantly lower than the Choregraph counterpart with 4660. A sidenote is the significant amount of code duplication in Choregraph projects as described in Subsection 4.1.4, which increases the number without adding functionality. Lines in the dialog are not counted as they do not contain code instructions.

The comparison is unfair for several reasons. First of all, the Total lines for both products do not fully

	CASperSocket - IBM			Health Insurer
	On robot		On Cloud	Choregraph
File	**Lines of Code**	**Source code lines (SLOC)**	**Lines of Code**	**Lines of Code**
audio_manager.py	397	*269*	210	-
setup_system.py	359	*245*	150	160
video_manager.py	250	*175*	10	-
behavior_helper.py	107	*70*	-	-
tablet_manager.py	112	*67*	-	-
websocket_client.py	66	*48*	-	-
websocket_manager.py	71	*48*	-	-
Scenario	-	-	1100	4500
Total #lines per source	1362	*922*	1470	4660

Table 4.3: Table showing (S)LOC for the two developed product using the Watson Toolbox (1362 + 1470 = 2832) and Pepper Toolbox (4660)

represent the same functionality. The IBM product has more functionality in terms of the added capabilities of Speech to Text, Dialog and Vision and remote control, although it can serve the same purpose as the NAOqi product. Second, the Watson product contains code running on the robot and in the cloud, together forming a Framework comparable to NAOqi. As can be seen in the table, the actual amount of code required for the scenario is much lower (1100 lines). Functionality wise, the LOC of On Cloud - Scenario is a more accurate comparison showing a big advantage for the cloud solution. However, boxes of functionality in Choregraph are easily dragged in the flow without seeing any code. This counts for the LOC, and thus as effort, yet it does not cost any effort for the developer to actually program, significantly decreasing actual effort when using NAOqi. Building the Health Insurer product using the Python NAOqi API would have resulted in a better comparison. It is still expected that scenario development using this way will still cost more developer effort.

Concluding, when looking at both the total LOC of both products and looking at the LOC for programming the scenario only, the cloud requires less code (and estimated less effort) for (at least) the same functionality. The framework for the cloud option only has to be made once after which it can be used for any scenario.

5 Evaluation at a Health Insurance Store

For testing the interaction quality of the first prototype using only standard functionality from the Pepper Toolbox, an evaluation was performed at one of the regional offices of a Dutch health insurance firm Located in Heerhugowaard, this office is used by clients who do not want to or cannot use the online services offered by the firm. Also, clients come in to drop off physical documents or be advised on products. Personnel and clients usually enter through different doors, resulting in the main entrance for the most part being used by clients only. A picture of the office with a view towards the entrance is provided in Figure 5.1.

The evaluation was performed during nine non-consecutive days in different weeks to allow for improvements to be made in between. The results of this evaluation are as a baseline and comparison for the more extensive evaluation to be performed with the cloud-enabled robot later. The companies goal of the evaluation was to determine whether the robot can add value to the existing client-contact and to identify what clients think of it.

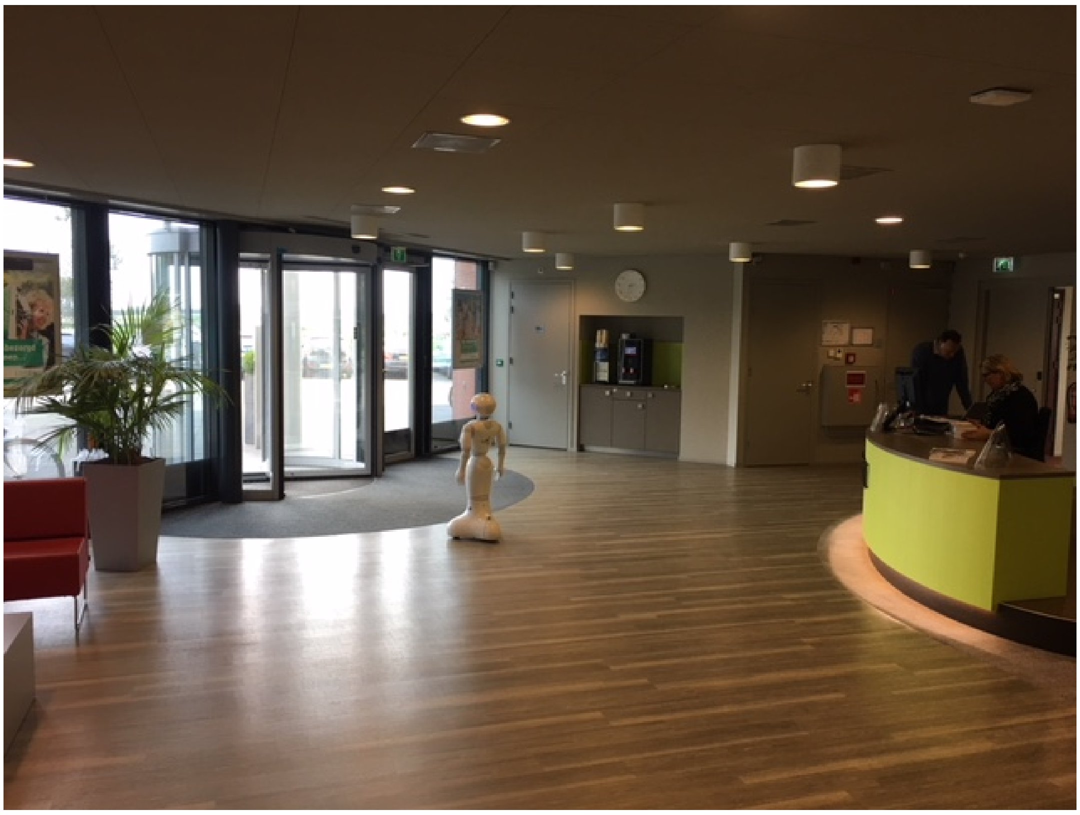


Figure 5.1: Evaluation location with Pepper positioned directly after the main entrance and before reception.

5.1. Materials

The Pepper robot running an application built with standard NAOqi functionality (from the Pepper Toolbox) was used during the evaluation. The locations of the robot were various positions between the main entrance turning doors and the reception area, as the possible placement in the space is part of the research. The supported language in the application is Dutch.

Next to the robot, a wireless 4G (MiFi) router was used for internet connectivity, offering a local network to control the robot.

5.2. Participants

Since this office provides services for clients not being able or not wanting to use the internet for help, the average visitor is not technologically skilled, and the age range is more often than not between 40-80 years old. No quantifiable data on this was gathered.

Participants were not notified upfront that the robot is operating, although Pepper being used in an experiment had been in the local news in the days before testing, resulting in some clients being aware beforehand. Almost all participants had an appointment on a specific day and time with an employee. A total of 56 persons have participated in the evaluation.

5.3. Procedure

At the beginning of the day, the observer places the already booted robot between the entrance doors and the reception. The appointments of that day are manually loaded on the robot using a list in a pre-defined format. The observer takes place in a corner of the room so that the robot, entrance, and reception are still in visual sight, but the observer is not noted directly by clients when entering. The hospitality application is activated, and the reception is notified of the start of the evaluation.

Clients entering through the turning doors find the robot immediately in their path and thus must choose to start interacting or go past the robot to the actual reception. Visitors making the latter choice are asked by the receptionist to try registering their visit using the robot. Interaction is initiated by standing in front of the robot so that the participant is in eye (camera) sight.

The observer takes notes on the behavior of interacting clients. A specific observation sheet or question list is not used, but notes are taken on estimated age, what went well and not so well during the interaction, what could be improved, which position in the room seems to work, and so on.

The observer can intervene on autonomous functionality by making manual state changes if needed, e.g., continue from 'detecting people' state to 'start of interaction' state if the robot does not do this by itself. The observer can also restart the system, tell the interacting client to speak back (as this sometimes is not understood immediately) or use the tablet to continue in the conversation.

The following data is collected automatically:

- Time of start (person detected) and stop (feedback is given, the person leaving) of interaction
- Score of mini-quiz at the end of the interaction, scale 0-3
- Which steps in the interaction flow have been reached and in what order
- Which modality has been used at every interaction step (state: speech/tablet)
- Feedback score (on the total interaction, scale 1-5) on the question (translated): "How would you rate this interaction"

5.4. Results

This section presents the results of the evaluation. Due to a bug in the data collection, only starting times of interactions have been recorded. The stop time is not present in all logs, and thus, the duration of interactions could not be determined.

Positioning Various positions for the robot have been tested surrounding the walking path, from the door to the reception. It has been observed that clients are less likely to walk around the robot if it is placed directly in their walking path, although no quantifiable data has been obtained for this. The same holds for the observation that younger people (estimated age < 40) tend to prefer approaching the robot instead of the reception.

Engagement With the positioning now fixed in the direct walking path from the door to reception, about half (43%) of the participants showed proactive engagement, meaning this percentage of people approached the robot by themselves. The remaining percentage of people was stimulated to try using the robot by the receptionist, and about half of those people (so a quarter of total participants, based on observations) did that. The remaining quarter indicated only wanting to speak with humans, or they were in a hurry and of no interest in this.

Communication modality and client experience One could either speak to the robot or use the tablet to perform the interaction. 52% of interaction steps are performed using speech, and the remaining amount used the tablet. It must be noted that one on the interaction steps, giving feedback, could only be performed using the tablet. The client experience is 75% positive. This value describes how clients value the addition of this Pepper robot to the office services compared to regular service. This is obtained by interviewing the clients directly after the interaction or after the appointment. The feedback score on the interaction itself, asked by the robot at the end of the interaction, is that 79% responded four stars or higher out of five. Finally, 64% of clients indicated Yes or Maybe on the interview question whether this robot implementation fits the company.

Interview learnings The following learnings have been obtained by summarizing the interview results with the participants. Table 5.1 shows the general learnings and an explanation.

Learning	Explanation
The interaction between client and Pepper must be started by the robot using proactive speech.	This makes it easier for humans to approach Pepper and know what to expect of it.
The clients expect an open question dialog, seemingly resulting in a mismatch while communicating with Pepper.	This prototype has an extensively guided interaction resulting in closed questions and a limited amount of possible answers from which cannot be deviated. Humans naturally respond in different ways, and thus Pepper does not always recognize the user's input.
There must be a handover moment available from robot to a human agent.	Pepper can now only perform a part of the interaction and the company always want to offer a form of human contact. Referring to a human agent is a natural way of solving problems or processes these robots cannot handle (yet).
The way of placement and visibility of the human receptionist determines for a part whether and how people approach and interact with Pepper.	If getting to the human receptionist is easier (closer, no queue, friendly smile) than getting to the robot, people tend to prefer that.
For a follow-up implementation of this prototype, a real-time link between robot and client-appointment-system needs to exist to ensure quality.	Sometimes an appointment was missing due to inserting that on the same day, while appointments were loaded at the start of the day creating confusion for the user.
Pepper can add value to the client interactions of this company for less complex client interactions byways of speed, fun and wow element.	For simple interactions Pepper can add a positive experience and help for clients for lower cost than a regular human receptionist. Replacement of a receptionist is a no-go though since the robot still must be backed up by a human for interactions or tasks it is not programmed for and thus cannot handle.

Table 5.1: Learnings from the questionnaire with participants from the first evaluation

5.5. Discussion

The created prototype is capable of completing its task, but more than often needs some human assistance to achieve that. Speech to Text often does not work by falsely recognizing (an already limited amount of) keywords and sometimes activates without any voice input at all. Since a large part of the interaction is hardcoded instead of being dynamic, the interaction is not easy to adapt.

Positioning The greatest engagement and participation has been achieved by placing the robot in the middle of the direct walking path. This way, one has a short time to decide whether to approach the robot or continue to the (human) receptionist, as one is confronted with a robot which starts a conversation proactively. For customers (especially at this location) this could feel like the technology is forced onto them and the human receptionist is only there as a backup, while using the robot should be a voluntary move for them to accept this way of interacting. Younger people act different, as placement outside of the direct walking path suffices and (after a quick visual check with the human receptionist to see if this is expected behavior) clients approach and use the robot.

Engagement Engaging proactively is especially useful for the average client base (40+) since they initially tend to avoid 'this piece of talking plastic' (actual quote of a client). They are not used to this form of communication. After participating, they usually find that the same goal has been achieved without much effort and indicate that they also would use it next visit. This could indicate an aversion to the unknown and new technology, so a less ’aggressive’ approach can be used after this form of technology and communication is more common and accepted. For the younger people walking in, this is less of a problem and often initially even prefer talking to the robot instead of the human receptionist.

Communication modality The relatively low amount use of speech can be explained by the low recognition rate of the built-in speech recognition. Once people encounter a speech-to-text transcribe error, they continue the rest of the conversation using the tablet since the buttons on the screen provides a clear and more known alternative. They do not switch back to using speech after using the tablet.

5.6. Conclusion and summary

An evaluation has been performed in a Health Insurance store for nine days. The goal was to see whether the robot serving a hospitality scenario adds value for the client-contact in the office. It also provides a baseline for the cloud-enabled robot evaluation. A Pepper robot was used with an application build using standard functionality from the Pepper Toolbox.

The robot was placed in the walking path from the door to reception as this gives the most significant chance that users choose to interact with the robot, next to pro-actively engaging them. The robot engages a person when in eyesight. Data on the interaction is saved automatically but is also observed and noted down from a distance. The data shows that users use speech about half the time as communication modality. The tablet, meant as a backup input option, is used by users when speech commands fail. When using the tablet once, users rarely switch back to using speech.

Fifty-six users participated in the evaluation. Clients are generally positive (75%) about the robot hospitality experience, with 43% engaging the robot by themselves.

6 Evaluation at the IBM Client Center

For testing the interaction quality of the second (cloud-enabled, see Section 4.2) prototype, an evaluation was performed at the main entrance of the IBM Client Center in Amsterdam. The same use case and requirements from Section 3.3 were used for this evaluation.

This evaluation was needed for answering Research Question 2 on how the use of Watson's versus Pepper's toolbox compares with respect to the quality of the human-robot interaction. Research Question 4 on the up and downsides of using a cloud solution for robotics when looking at business values was also partially answered using costs and experiences of this evaluation.

The IBM Client Center is located on the ground floor of the IBM Netherlands headquarters where IBM technologies are demonstrated, and various events and meetings are taking place. This results in a dynamic and challenging surrounding for the robot to operate in. Through the entrance regular IBM personnel, contractors and clients with appointments are entering the building.

The evaluation was performed on six (non-consecutive) days. The aim was to gather at least 10 participants per day, which would result in at least as many participants as with the first evaluation. A pilot was performed first to observe the real-life performance of the robot and improve the interaction, fixing technical difficulties, and confirming that the evaluation procedure works. The evaluation focused on observing and gathering data only. A single interaction was expected to have a duration of around three to five minutes. With the gathered data, a comparison was made with the results from the previous evaluation. This chapter describes the setup and way of executing the second evaluation, next to providing the results and discussion of the results.

6.1. Materials

The Pepper robot running the cloud-enabled CASperSocket was used during the evaluation. CASperSocket is the middleware running on the robot as described in Section 4.2, combined with logic and data processing in Node-RED using IBM Cloud services. An active internet connection is required for the robot to connect with the cloud. For the pilot, a Mifi router was used to set up a 4G mobile Internet connection and provide Wifi connectivity on the 5GHz band. The evaluation itself uses the IBM intranet as this has no sudden latency lags, which happened using 4G.

To count all the visitors entering the building (some of which participated in the evaluation) a sheet of paper displaying the date and 15-minute counting windows were used.

A sign holder is used to inform people of, and how to participate in, the evaluation. Informed consent is taken care of by placing the information sign directly after entering the turning doors of the reception area, before being in the direct neighborhood of the robot. The text on the sign describes the goal of the evaluation (testing a hospitality cloud robot scenario), the fact that no personal information will be saved and that you only participate with the evaluation when you approach and stand in front of the robot on the left or can continue to the reception on the right. If people pass and do not approach the robot, no interaction is started, and the robot stays in idle looking-around mode. Finally, the sign says that everyone can opt-out of the evaluation, also when someone has already interacted, simply by indicating this at the reception.

6.2. Positioning

The position of the robot is next to the walking path from the entrance turning doors to the reception desk. See Figure 6.1 for an overview of the robot's placement and environment.

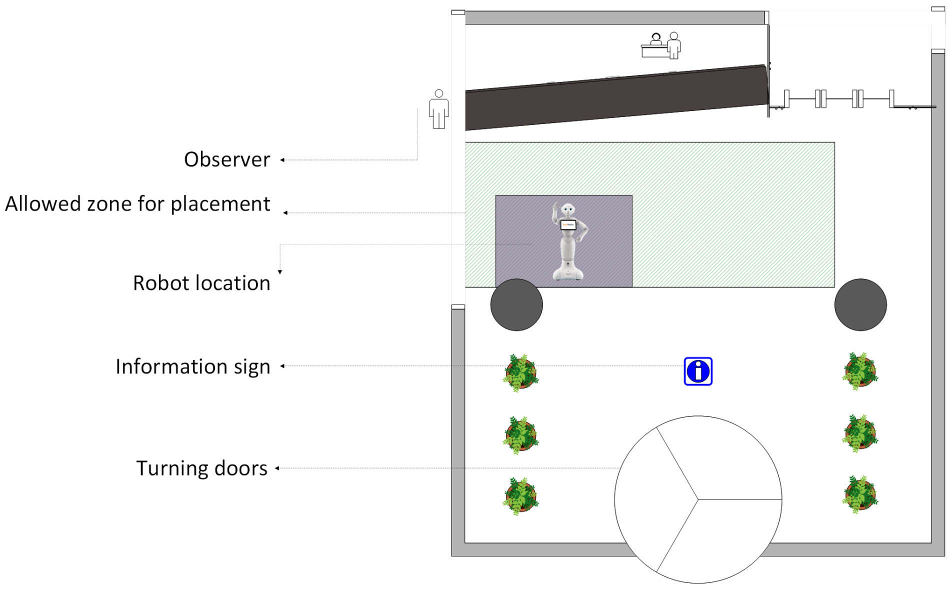


Figure 6.1: Overview of evaluation location depicting the reception, turning doors, automatic gates, placement of the robot, the location of the observer and an informational sign.

This position is meant to make people aware of the robot's presence and thus attract as much visitors as possible to the robot, while still offering the option to walk around it if a visitor wants to talk to a regular human receptionist or does not want to participate in the evaluation.

The choice for this location has been made based on earlier experience, and related work indicating is not favorable to put the robot in a direct line from the door to reception as this could discomfort people[20]. The use case targets visitors of the building and not personnel or others who are using the automatic gates instead of going to the reception. The latter group is not excluded from the evaluation as they can still try other functionality. If the robot would be placed further from the walking path, people tend to ignore it faster and opt for human assistance. For security reasons, the robot could only be placed in specific areas indicated in green in Figure 6.1. The evaluation was only allowed if the position is not in a direct line to the reception desk. These inputs combined result in the final position, as shown in Figure 6.1.

If someone decides to approach the robot, it will engage a conversation automatically. Figure 4.2.4 further describes how engagement is implemented. By making an active choice to approach the robot, more general engagement of users during interactions is expected than the situation where everybody is requested to make use of the robot without making an active choice as in the first evaluation. We do not expect this influences the results as this was also the case in the other evaluation.

6.3. Participants

As the use case (Section 3.1) describes, the target group is clients/visitors who have an appointment with an employee. The target group can be visually identified by badge color or the initial lack of a badge, which they receive at the reception required by company policy. This group has not been filtered or selected beforehand for the evaluation and are unknown to the robot when walking in. The author informally observed that the visitors are technically skilled and of working age (20-60), with a near 50/50 divide between genders and mainly having the Dutch nationality but internationals are no exception. Quantifiable data on this was not obtained.

Not only the target group will pass the entrance and reception area, as the same entrance is also used by personnel, contractors, and 'other': people such as pizza delivery, taxi drivers, florists, etc. These groups may also interact with the robot, and data is also gathered from them as some useful insights were also drawn from this.

Everyone who enters is made aware of the robot's presence by use of an information sign, depicted as an "i" in Figure 6.1. This is required by human experiment rules from the TU Delft, when not giving explicit consent beforehand. The sign also helps to stimulate people interacting with the robot and to prevent a surprise effect as robots are not common in reception areas, yet.

An observer will count how many visitors pass without noticing or choosing to interact with the robot. Visitors are not forced to take part in this evaluation. No compensation is given for participating.

6.4. Protocol

In order to perform the evaluation consistently, a strict protocol was set up. The protocol consists of setting up the environment and robot, positioning of an observer, ways of gathering data, and what to do with irregularities.

An observer is present during the evaluation to collect impartial data which cannot be fetched automatically. Also for safety reasons and guidance for receptionists, an observer is wanted.

To start the evaluation, the observer moves the robot (already booted) to the reception area. The robot is placed on its designated location facing the entrance as can be seen in the middle left of Figure 6.1. Also, the sign is put on its location just behind the turning doors. The receptionists are notified that the evaluation is now taking place. The scenario in the CASperSocket system is activated, and the start time is noted down on the sheet. The observer now makes sure to test the system once to check if everything is operating as expected.

The observer sits down at the designated location seemingly 'working' as if not part of the evaluation. This means the observer is visible but not in direct sight of the user when interacting (see the top left of Figure 6.1 where the observer is seen next to the reception behind two plants). This is to ease the experience for the participants since during the pilot it was indicated that it feels uncomfortable to interact with the robot while being observed by multiple persons. In order to reduce bias, the observer will not guide people to the robot, give instructions, or interact with the participant unless asked to.

To measure engagement, the observer also starts counting visitors (reliably identifiable by badge as described in the previous section) walking by and not interacting with the robot per 15 minute intervals on the counting sheet. If someone approaches the robot, the interaction is started, and the observer also starts watching the participant's behavior using the observation sheet shown in Appendix A.2.

The expected behavior is that a participant walks in, sees the sign, and reads the content. The person then approaches the robot after which the robot initiates the interaction (if it sees the person) by saying hi and asking how to be of help. It is expected that sometimes the robot is looking in another direction, and thus, the person must first say something to attract the robot's attention (item on observation sheet). The user then continues to answer the robot's questions and give feedback at the end of the interaction, after which the participant is forwarded to the receptionist to actually register and receive a badge.

The output given by the robot is fully visible for the observer, as the robot is in eye-sight and within hearing distance. Also, the screen of the tablet, displaying the full conversation seen by the user, is visible on the laptop of the observer in real-time. This means that the observer can see and hear when a Speech to Text result is right or wrong, and thus, why the robot performs as it does now. These observations are marked by the observer on the observation sheet for manual data gathering (see Subsection 6.5.2).

Irregularities If there is some irregular event, engagement is lost, or more information is wanted from a participant's interaction, the observer may approach them for a questionnaire afterward (see Subsection 6.5.3). The robot may continue to help another participant in the meantime, but no data of the interaction is processed.

If the hospitality functionality stops working and the robot becomes inoperable, either by losing connection, a software error or an error of the robot, the application (or robot) is restarted, and data of the current interaction will not be used. If the participant is still around the option to perform the interaction again from the start is offered, after explaining what just occurred.

6.5. Data gathering

Data is gathered automatically by the robot and manually by an observer during an interaction. This section describes what is saved and what purpose this serves.

6.5.1. Automatic data gathering

Table 6.1 contains the data types which are (automatically) collected during an interaction. This data is all saved in a cloud database, performed by Node-RED.

To comply with the GDPR, the data is not linked yet to other data by, for example, an ID. Instead, as soon as the robot is engaged and says hi to the user, a timestamp is saved and used as an identifier. This also means that there are more automatically gathered results than manual, as sometimes there is a false activation (and the scenario stops shortly after, saving the empty result) or the observer is busy performing a questionnaire and no data is manually gathered. The results can, however, be filtered easily later by matching it with the manually gathered timestamps.

Data type	**Reason for saving**
Full text of the interaction from the user and robot	This is a built-in feature of Watson Assistant for improving future interactions by checking (and if necessary correcting) classifications and responses.
Modality used at every interaction step (is the tablet or speech used)	Find which modality people use first and stick to. This can be compared with the previous evaluation.
Time measurements (using timestamps)	To determine - interaction length (beginning of the interaction at saying hello, until the end when the user leaves), - total running time of the evaluation - idle time (robot not interacting with anyone but during evaluation time). This can be used to see if users can perform the interaction in about the same time or whether this differs, or which steps take (too) long to perform.
State changes	This can be used to determine in which order users perform an interaction and thus if they follow the happy flow, as well as finding places where users get stuck or walk away.
Feedback score on the interaction given by users when asked the question: "How would you rate this interaction?".	This gives an indication of how the interaction is experienced by users. This is displayed on the tablet and has a scale from 1-5 stars.
Whether the interaction is terminated (e.g. by walking away) before finishing it (without error).	The interaction step where this happened is saved for later analysis. If users leave often at a step, this may indicate a tedious or unpleasant experience.

Table 6.1: Automatically saved data types including reason to save.

6.5.2. Manual data gathering

Next to the automatic data gathering, interactions are observed. Observations are noted down in a structured way using an observation sheet that can be found in Appendix A.2). How to perform these observations is described on the second page of the sheet. Why these items are chosen to observe is described below.

Participant's details First of all, the participant's details 'gender' and 'age range' are estimated by the observer, considering how people react to the robot and perform an interaction seem connected to these variables. Age estimation is obtained through facial analysis with the Watson Visual Recognition service, which is mainly used by the CASperSocket-cloud for detecting faces but also gives an age range as extra data with the result. This is displayed realtime in the dashboard of the observer used to control the experiment It has been observed before (during the first evaluation and from the author's experience) that age is of influence on how people cope with the robot.

Ranges for age are: younger than 20, 15 - 40, 30 - 55 and 40 - 65 and older. These ranges are expected to be found in this location and are descriptive for according behavior; younger generations have grown up with new technology around them, are generally more open to that and are experienced with how to approach it while older generations seem to prefer humans. Even with the help of Watson, age is an estimation if not asked, and as a result, the ranges overlap so the observer can tick the most fitting age range with high certainty. Gender is a subjective matter as one cannot always be sure of this by just observing, and thus is not to be ticked on the sheet at all if unsure, to prevent entering wrong data.

Easier to see is if a person is alone or in a group and whether this person is a visitor. Determining whether a participant is alone is important for the robot's gaze and hearing functionality. The robot focuses on the first face it recognizes and keeps locked on that person during the entire interaction. If this is the wrong person, the likeability of the robot for the actual interacting person may be lower since the robot does not make eye contact and speech recognition may work worse since the robot only listens to persons right in front of it, possibly degrading interaction performance. Determining the person's type can be done according to the badge the person wears (or receives) and is important for data filtering since the use case targets visitors. It takes minimal extra attention of the observer to check whether a person receives a visitors badge later on whilst observing the next interaction.

The robot's behavior robot, as well as the user, are observed to see what influences a pleasant and fluent interaction. How do participants interact?

Behavior of the robot The robot is observed to check whether the cloud implementation works as expected. The robot has no way of knowing whether the Speech to Text result corresponds to what is being said in real life. Since the observer can hear the participant and see the output on the tablet in real-time, it is possible to check for errors, thus explaining wrong behavior or responses from the robot). This was written down and not communicated to the user. If these errors happen often, one may conclude that Speech to Text, even using the theoretically better performing cloud services, is not yet working good enough for this type of unrestricted interaction. It may also be observed that the robot's response time is too slow for pleasant conversation, which could be because of late STT results, a slow Internet connection or for example a beamformer with high latency (>2 seconds delay). This is another relevant factor for doing this kind of interaction for businesses as we expect response times must be at a minimal level (for example a maximum user wait time of three seconds after the end of a speech utterance) for a pleasant experience.

The same holds for Visual Recognition results, which could influence engagement by detecting (and thus greeting) a user too late or too early which may give a bad start of the interaction resulting in lower feedback ratings. Inadequate visual recognition could also lead to the robot looking the wrong way. When a person is not in direct sight of the robot the beamformer's advantage becomes a disadvantage by filtering away speech coming from that direction, and thus the user cannot be heard anymore ending an effective interaction. This is one of the reasons why the use case only focuses on conversations with one person as with multiple users the robot could possibly lock onto looking at only the non-interacting person.

Behavior of the user The user can also be a cause for troubles in the interaction itself. This often could be because the user does not know how to behave with this new kind of technology; either not adapting to an automated system at all by for example saying lots of extraneous sentences, speaking with low volume or speaking slow. Another reason is the user adapting too much to an automated system by, for example, speaking only with keywords, suited for keyword-based speech recognition systems, which is not how Watson STT works. All these elements could confuse the Speech to Text resulting in a changed and possibly unintended interaction.

Also, the user could not know how to handle this technology, observable by the person asking others what to do, leaving the robot mid-interaction or continuously looking for confirmation in other people. All these items are therefore an item on the observation sheet since these are of strong influence on how well the interaction goes.

If engagement is lost for some reason, it should be noted down why this probably happened to see if the cloud influences this. Finally, we can find the preference of modality of a user (speech or tablet) by looking (automated) which modality is used per interaction step. However, when we nudge a user to use the tablet (as is asked for giving feedback), is the user then still inclined to continue using the tablet or does one switch back to using speech? This 'switchback' is, therefore, also monitored by the observer.

To capture other interesting unforeseen behavior, a large general comment field is added to the sheet. Things to note here are off-script events (when either robot or user deviates from the expected behavior) or environment variables other than usual such as loud noises, a large group of people near the robot talking or windows cleaners in sight of the robot.

6.5.3. Questionnaire

A Questionnaire is performed to get qualitative insights, such as determining what elements need improvement or how participants experience the interaction. If an interaction is observed with irregularities (not a standard *happy flow*, premature loss of engagement or the interaction was for a large part not fluent), the observer approaches the participant afterward to conduct a post-interview. The observer estimates the likelihood of a participant's willingness to complete the questionnaire and approaches only those that are estimated to have time to do so. Also for interactions without irregularities, participants are selected for the questionnaire if they are estimated to have time to perform it. In the questionnaire, there is room for an open response. Questions asked can be found in Appendix A.1. Six scale-based questions are asked on the experience, based on the metrics in the paper by Bartneck [33], a standardized method to measure users' perception of robots. As the list of possible metrics is long, and the goal is to keep the questionnaire short, only the most relevant metrics have been selected based on relevance to the use case. These metrics, using a 5 points Likert scale, are: *machinelike - humanlike, artificial - lifelike, fake - natural, unpleasant - pleasant, unfriendly - friendly, no control - in control.*

6.5.4. Privacy and GDPR

Data gathered (by the robot) will not leave the IBM cloud, compliant with GDPR rules. The name of the client is only used during the interaction for personalization and simulating the gathering of appointment data. This data is never saved on the robot. Gathered data is only coupled based on the starting time of the interaction, so no link to the participant exists. A participant can opt-out at any moment after which that data will be removed.

6.6. Pilots

Two pilots were conducted. The focus of the first pilot was on testing, verifying, and improving the software system as a whole and see how users respond to the dialog. The focus of the second pilot was on evaluating the experimental procedure.

6.6.1. Pilot in office room

This first pilot allows testing the integrated system with participants, including all software modules and dialog used for the evaluation. This pilot shows whether the system works as expected, whether people understand how to interact with the robot and show possible errors in dialog or interface.

The evaluation of this pilot is different in context by that it is performed in a silent room with one observer guiding the interaction. This influences the behavior of the participants, and Speech to Text results are expected to be more accurate than with the use case at the reception.

The CASperSocket system with the same dialog (Watson Assistant workspace) has been used in another study, which is about the effect of different conversation recovery strategies on people's perception towards the robot's social attributes [34]. A detailed description of how the dialog is constructed and choices behind wording can also be found here. Since the experimental setup of that Thesis is close to the interaction from this paper's evaluation, these results are used as pilot data and to improve the interaction experience and software before the actual evaluation at the reception.

Participants were allowed to meet the robot before the actual evaluation by chit-chatting for a few minutes. The Hospitality scenario was not activated while getting to know the robot. The user is not guided in the conversation and cannot confirm appointments but can talk about general topics. This was done so the participants know how to speak with the robot and are not displaying first-time-use behavior such as taking photographs, giving comments on the experience while giving the robot instructions (having the effect of creating bad speech transcriptions and thus wrong robot responses), starting to speak too early (robot is not listening yet) or not knowing that the tablet has a touchscreen. After this, the hospitality scenario was activated.

A total of 39 participants, all IBM employees and from different nationalities, were given a story in advance: ' You are called <name>, you have a meeting with <employee>, and your goal is to register your visit using the robot'.

Observations and resulting adaptations During the pilot, the observations below were made. Some adaptations have been made to the system to solve these problems and observations and improve the fluency of the interaction.

Watson Speech to Text works with high enough accuracy for this type of interaction. A large part of sentences are transcribed correctly, and Assistant corrects small errors such as missing words. Accents do create some troubles with wrong transcriptions, but by repeating the sentence more pronounced, the interaction could still be finished following the *happy flow*.

Users do not experience problems in how to use and approach the robot given the scenario. This observation is drawn because people did not doubt what to say, did not have to repeat themselves (often) and people were able to finish the conversation until the end, following the *happy flow* of the hospitality scenario, thus finishing the given task. It must again be noted that people were allowed to get to know the robot before activating the actual scenario, and they were given a goal, whereas in the evaluation at the reception this will not be the case possibly influencing the participant's approach. To improve this more, the dialog has been changed to have more structure and be more precise in how questions are formulated, leaving less ambiguity what the robot wants to know from the user. Also, more suggestions and buttons on the tablet have been added at various interaction steps as it seemed to help people what to say. Next to that, buttons on the tablet have been enlarged and made more clear.

English Speech to Text does not work well in a Dutch oriented office. Due to English Speech to Text, half or more of Dutch names are not understood correctly and converted into likewise sounding existing English words, which makes a scenario identifying the client hard to do. People could use English pronunciation for their name to solve this problem, but this has to be told in advance. As a solution, an alternative for name entering has been made in the form of a software keyboard on the tablet. If there was no correct name after the first try, a keyboard would open automatically, containing the pre-filled text "My name is .". Participants used this option often after implementation.

Transcribing single letters does not work. During the quiz game, users have to answer 'option A', 'B' or similar. However, Speech to Text did not transcribe single letters correctly resulting in wrong input for Watson Assistant. This sometimes made the conversation go in wrong and unintended directions. To solve this, the answer space at the quiz has been made less broad. It wasn't indicated whether people could say for example 'Option B' or the actual answer to a question and furthermore 'Option B' was often not correctly transcribed. Now instructions are given at the start of the game to either say 'option <number>' or click the answer on the tablet. 'One, two, three, and four' can be transcribed by Watson STT, while 'A, B, C, and D' cannot in our case.

Need to use a clear separation of actors on the tablet. Some people were confused with the text on the tablet since it was sometimes unclear which side (left for user transcribe, right for robot response) represented who. This has been improved by making the difference between speakers more clear and remove (fade away) transcriptions with low certainty sooner.

Finally, saving of data has been fixed as this failed during the pilot. Now correct timestamps and timezones are used, and missing data for the experiment has been added.

User feedback The automatically gathered feedback score given by these users (scale 1-5, 36 out of 39 participants gave feedback) is on average 3.8.

Every participant was also asked to perform a questionnaire on how the flow of the interaction was experienced, as well as speech, retrieval of names, delay in (robot) reactions, movement of the robot, tablet use, the game, and other items participants came up with. This feedback has been used to improve the system.

6.6.2. Pilot at reception

The second pilot focuses on testing the experimental procedure and is performed at the reception as our use case defines. This pilot will show possible errors in placement, procedure, or system performance at this location.

After two hours of testing spread over two days, some changes have been made to the procedure of the evaluation and the software of the robot. The first 15 interactions did not result in a completed *happy flow*. The following observations have been made:

Speech to Text does not give adequate transcriptions in the reception area environment. The accuracy was low or a transcription was returned when the user did not even speak. Bad accuracy was expected as the CASperSocket (at this point in time) uses only one microphone out of four available in Pepper. In the first pilot, where the room was silent, and only one person was speaking this did not give problems, yet in the noisy surroundings of a reception area, this made the interaction go in other directions than expected.

Users change the way they speak after seeing the transcribing process. Showing in between Speech to Text transcriptions (see the transcribing process word by word while it is happening, so-called 'partials') seems to let users adapt their speech method during the interaction. Some users spoke slower, more pronounced, or waited a bit longer to start a sentence over the course of one interaction. This confirms the hypothesis that this method is a good feedback mechanism for users to understand more why the robot does what it does.

Wrong influences. In one occasion a cleaning lady noticed the robot (in the middle of an interaction with a user) and liked it so much she tried to communicate with it by shouting Dutch sentences(translated "Hello!" and "What are you doing here") loudly and positioning her head 10 cm in front of the robot's head. This resulted in the actual participant stopping the interaction and walking away. After getting no wanted response from the robot and seeing a few people looking at her, the cleaning lady also moved away.

Extra items on observation sheet

As a result of these observations, the following items have been added to the observation sheet:

"User says extraneous words for dialog": although the dialog system accepts a large variety of input sentences it still can return strange responses when saying non-relevant utterances, especially when these are said in Dutch.

"Switchback": if the user switched to using the tablet, will the user switch back to speech? The modality of input per interaction step is already captured by automatic data collection, but because the input is also given by 'ghost transcriptions' caused by surrounding noise, this data is not too reliable and needs to be manually confirmed.

"Engagement": whether the robot starts the interaction too soon (nobody in front of robot or still far away) or too late (someone wants to start a conversation, but the robot has not activated the scenario yet).

Changes in protocol and software Some changes in the evaluation protocol were implemented according to the feedback of users and other observations.

First, the observer should not be too visible. Feedback was given that users feel not at ease when talking with a robot while three people (two receptionists and one observer) are watching. Next to that, a counting sheet is added to see how many visitors are passing (and thus not interacting).

Second, to improve sound quality and thus speech processing, a beamformer has been implemented on pepper making use of all four microphones. Without this extra step, the full evaluation would not have been possible as designed because of too many input errors with the noise from the surroundings. More on the beamformer and its implementation can be found in Figure 4.2.3.

The software has been adapted to not ask for the name of the employee after this was already determined. This seems like a simple check, and in the pre-pilot, this problem did not occur. However, the added extraneous Speech to Text input due to surrounding reception noise resulted in the dialog system not being able to handle all the extra added digressions resulting in off-script answers. The Watson Assistant workspace has been adapted to include multiple checks for not asking the same information more than once. The use of a beamformer resulted in an improvement of this behavior by decreasing the amount of random input.

At this step, engagement measuring has also been improved; during this pilot, the interaction was started manually, and the robot could stop it by itself if no user was present at a point. Now the robot could also initiate the interaction on its own when seeing a face in the engagement zone, meaning a person has approached within 1.5m in front of the robot.

6.7. Evaluation

In total, the robot has been in function at the reception for 18 and a half hours, spread out over six days. A total of 61 interactions were monitored, of which 39,3% completed the happy flow. 32,8% of the participants who completed the *happy flow* were a visitor, giving an average rating of 4.45.

The manually and automatically gathered data points per participant have been matched on timestamps using an automated script and merged to create one dataset for further data analysis. We will call this the *basic dataset* in the rest of this document. The script also converts some data points to more accessible and processable data, from JSON format to CSV, for data analyzing purposes. An example of this conversion is the list of states the user went through during an interaction which is reduced in complexity by changing <*state name* + *timestamp*> to an integer indicating the total number of states.

The automatically gathered data of a small part of interactions (6 out of 61 or 9.8%) is missing in the database for an unknown reason. To reproduce the missing data into the basic dataset, the logs from Watson Assistant were used which contain the complete text of the interactions. Most data points could be reconstructed by counting manually or deduced from the text and otherwise left empty.

6.7.1. Results

In this section, the results are shown which have been obtained by running the evaluation at the reception described in this chapter. The results are calculated from the basic dataset.

Since the interaction was designed for visitors and not employees per se, most of the figures and calculations focus on the subset of data with only visitors. If otherwise this will be explicitly stated.

Feedback scores The feedback scores are given at the end of the interaction, on a scale from 1-5. The average feedback score from visitors who followed the *happy flow* is 4.45. 65.5% of all users have given feedback, where this is 81.9% for only the visitors' group. Employees often left before finishing the interaction (feedback is asked at the end) resulting in a lower percentage of feedback given by them. 95% of the visitors, which followed the *happy flow* gave feedback of 4 or higher. All feedback scores are represented in Figure 6.2.

As can be seen from the data, almost all participants, especially when falling in the targeted visitor group,

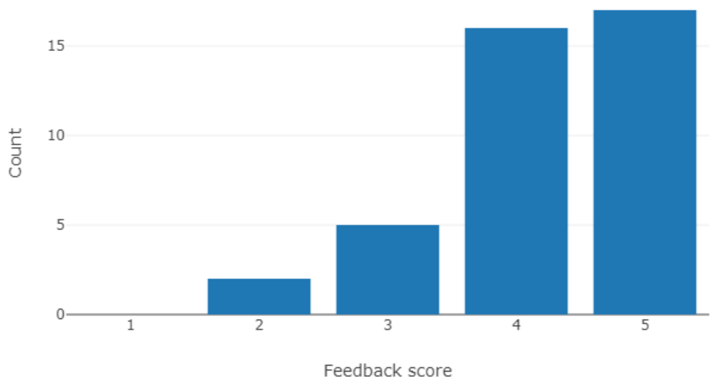


Figure 6.2: Feedback from all participants, if given(n=40). Not given n=21

give a positive feedback score when doing the full interaction as intended. When failing to finish the interaction following the *happy flow* participants give lower scores or do not give feedback at all by walking away.

Average duration of interaction People spend varying time performing an interaction. A boxplot of the interaction times in minutes is provided in Figure 6.3 where three groups can be compared; all conversations, only visitors, and visitors having completed the *happy flow*.

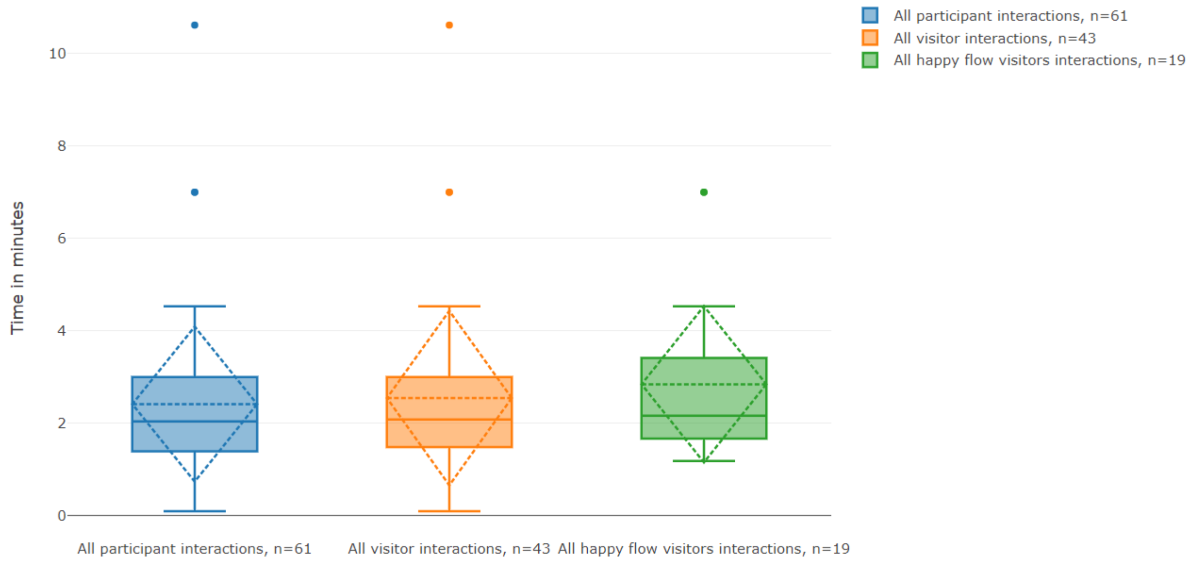


Figure 6.3: Boxplots of duration of interactions in minutes, from three data subsets

The figure shows that the intended target group which follows the intended flow in the interaction (see 'all *happy flow* visitor interactions') performs the interaction in about the same time with an average of 2 minutes and 50 seconds and a standard deviation of 1 minute and 40 seconds. The outliers in the other two groups are bigger, yet we could not find any significant differences.

Communication modality: Speech vs tablet Speech is used the most as communication modality. If we look at the percentage of speech per interaction; *count of speech inputs / (Count of speech inputs + count of tablet inputs)*, and average this number over all interactions we see an 86.99% use of speech. People performing the *happy flow* were forced to use the tablet at least once when giving feedback. Participants playing the game were also stimulated to use the buttons on the tablet for giving answers, and thus the speech/tablet percentage changes significantly.

13 out of 61 (21.3%) participants performed a switchback, which means that the user talked again to the robot after using the tablet. The option to enter input through the tablet's keyboard (except for names) was used less than five times.

Happy flow The happy flow is completed when a user performs the interaction following steps as intended by the designer. 24 out of 61 participants (39,3%) followed the *happy flow*. Pearson correlation analysis shows that various measured data points are likely to influence the completion of the *happy flow*. The most significant ones with r larger than |0.3| are mentioned in Table 6.2.

Data point	Pearson value	Explanation
Leaves during interaction, engagement lost	-0,65	This negative correlation states the obvious; if you leave before finishing the interaction one has not performed the happy flow
Says extraneous words in dialog	-0,44	This negative correlation shows that if a user says extraneous text in dialog, the interaction will probably not result in a happy flow.
Incorrect Speech to Text results resulting in wrong behavior	-0,32	The same holds for failing Speech to Text results, which negatively influences completing the happy flow
Interaction complete	0,78	Completing the interaction (giving feedback or say goodbye) does not always mean following the happy flow, as this could skip the appointment making process.
Tablet used (count)	0,80	Using the tablet correlates highly with completing the happy flow. Feedback is given on the tablet, which is the last step in the flow, and thus this is to be expected.
State changes (count)	0,83	More state changes seem to correlate with completing the happy flow. This does not correlate fully, as one can 'change state' often by being stuck in the appointment making phase but not being able to continue on until the end. More state changing is thus not better per se.

Table 6.2: Happy flow correlation

Visitors count A total of 188 visitors have been counted passing by or interacting with the robot during the evaluation. Figure 6.4 shows the average number of visitors per half hour interval spread over all measurement days.

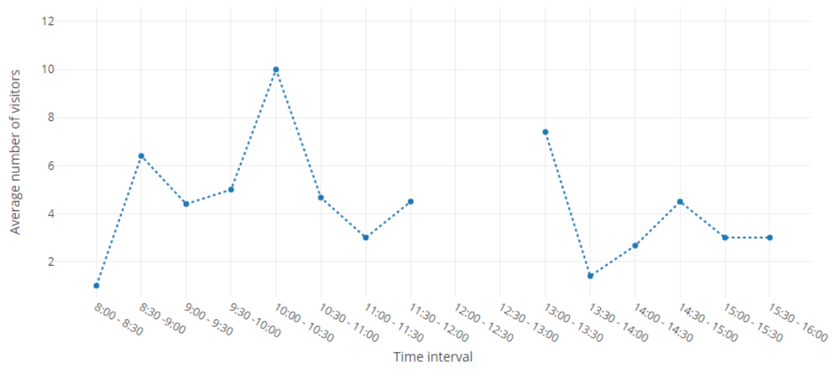


Figure 6.4: Average number of visitors per half hour interval over all measurement days.

6.7.2. Questionnaire

To get qualitative insights on experiences and points of improvement, a questionnaire was performed under 14 participants, as described in Subsection 6.5.3. In general, the participants like the robot itself and how it moves and talks. A general dislike is that the robot cannot understand users well and that names are hard to enter. Users also indicated multiple times that the interaction would go better if done a second time now that they know what to expect and how to use it. Finally, many indicate that Dutch language support would be instrumental in having in a Dutch office.

The first question of how to describe the experience of this interaction is answered with fun, cute, straightforward and humanly because it looks you in the eyes, but listening capabilities of the robot should be improved upon.

The next question on what went well and what not during the interaction has the clear answer that Speech to Text should be improved to reach an accuracy level that is acceptable to the users, but people notice that names is where the system generally fails. Participants like and dislike that the robot does not accept a wrong answer when it asks for something and that the robot is persistent when gathering appointment data. Further, the following user comment describes the average response on this question: *"Communication from the robot is great; continuous body language, feedback on screen is clear, yet communication to the robot could use some improvement"*. The following comment could also be relevant for future designs: *"The voice sounds nice and what she says indicates that she does not know all yet <" I'm still learning ...", observer>; that was nice to understand or accept the little mistakes she makes. You feel like you can teach her something, almost like a child, but I still find it useful and fun to do."*

When asked where the interaction went wrong (if that was the case) and why that probably happened, 71.4% responded Speech to Text and recognizing Dutch names as the cause for troubles. Also, accents, speaking too soon and using Dutch in general was mentioned by the majority of respondents as a reason for bad understanding by the robot.

When specifically asked what participants liked and disliked on the robot the favorable properties were shared among most of them; it looks friendly, the interaction is fun to do and nice, the voice and accompanying body language (including making and keeping eye contact) are convincing, and the activation works well. One person is happy that the robot keeps being calm while the participant itself was fed up and reacting irritated.

Negative properties are more person-specific: *"The movements of the arms really helped me think this is an actual person, this is almost creepy. I must say I startled one time because it seemed to move toward me when I touched the tablet."*. A few participants are confused by the coloring of the shoulder LEDs. These colors change depending on the engagement mode (red is not interacting, green is engaged, yellow is person lost but still engaged) and make it clear for the observer in what state the robot is. The red color was interpreted by these participants as 'robot out of service'. The built-in checks for name confirmations as extra interaction step <"Let me confirm, your name is ...?">, and the small tablet text are also mentioned as dislikes.

Some (21.4%) participants indicated that the interaction was too slow and/or too long to be used for people in a hurry and that it should get to the point quicker. In contrast, an almost equal amount said the robot should explain more about itself at the beginning of the conversation, like how to talk to it, what its name and purpose is and how it can help before asking questions.

Participants were also asked whether they find this robot solution adding to the hospitality experience at the IBM reception. Generally, participants agree with this, added that some flaws should be worked out and it should not replace the receptionists. Other responses include that it is a lovely way to show technology, that this only helps in busy offices and that using this should depend on the type of company (technology or others). One comment from a user is: "I like this since we live in modern times, but personally I find it freaky".

Suggestions for improving the robot are various. This includes multi-language support (which is built-in up to 10 languages except for Dutch, but not used during the evaluation), quicker responses on user input, more instruction by the robot at the start of the interaction, getting attention earlier by already greeting when coming through the door and introducing a 'go quick' button for recurring users saying only the essentials. Outside of scope but also named as improvement is letting the robot guide visitors to their meeting room. Multiple suggestions are on improving Speech to Text:

- The robot should provide better feedback when it is listening and when not; this could be done using the shoulder LEDs or properly timed microphone sign on the tablet since the sign is present now but sometimes displayed too late or too early for an unknown reason.
- likewise, it should be more clearly indicated that speech is being processed now, to prevent repeating when the robot did actually hear what was said, but the delay is larger than expected.
- Have a minimum waiting period between asking something and responding on (wrong) input, as the robot now sometimes repeats the question or gives the general 'I don't understand' answer almost immediately after asking the question
- Should indicate it does not speak Dutch if it recognizes that since this is a Dutch office
- Should indicate that the user should speak slower if it does not understand

Finally, participants were asked to rate their impression on the robot on a 5 points Likert-scale based on two selected keywords from the Godspeed Questionnaire [33] as described in Subsection 6.5.3. The averaged scores can be found in Table 6.3

Keywords	Average score	stddev
Machinelike – humanlike	3,07	0.59
Artificial – lifelike	2,79	0.93
Fake – natural	2,43	0.82
Unpleasant – pleasant	3,79	0.94
Unfriendly - friendly	4,64	0.71
No control - in control	2,50	1.05

Table 6.3: Average 5-scale likert score on given keywords selected from Godspeed questionnaire.

6.8. Discussion

The robot is not robustly usable in its current form, yet this prototype proves that the concept could work and is probably very useful if some more time is spent on refining the software.

Engagement worked as expected with the robot quickly initiating an interaction if the user is in sight. The only problem seen is that the current prototype looks around semi-random when Unengaged and is thus sometimes not looking at an approaching user. The user then first has to draw attention with his voice after which activation takes a few seconds. This was confusing to some and should be reasonably easy solvable by letting the robot look around less. After the face is locked, the gaze behavior of the robot works well.

The questionnaire shows that currently unimplemented features (either due to time or unavailability of the feature) are mentioned to be missing. Among this is the support for the Dutch language which is to be expected in a Dutch office with Dutch employees. However, as IBM is a multinational, international guests are likely to come in. Regardless of how well the Speech to Text engine is performing, when Dutch names are used transcriptions will be wrong with English recognition parameters. This has little to do with the robot itself, but the use of a wrong language or location, as in an English office this problem would not have occurred this often. Also, transcribing names is harder to perform than regular sentences using grammar or other algorithms for word prediction. Because of GDPR constrictions, we could also not spot names known beforehand in a sentence, which could have helped significantly.

The use of a beamformer seems essential in this use case. Without it, it is almost impossible to complete an interaction as wanted as the robot interprets noise as commands from the user. With the beamformer, the robot only hears the user. This, however, adds a little bit of delay, which is a downside in fluent interaction.

The percentage of completed interactions following the *happy flow* is quite low. This can partly be explained by the much busier lifestyle of the visitors at this location, as even when all functionality worked correct and as expected some still walked away, just because they had something else to do. Other users understandably gave up after not getting one step further when entering a name after trying for long. This step is essential for the specific use case presented in this Thesis. If names were not part of the interaction, the robot would probably operate much better overall resulting in a much improved completed interactions percentage. Giving the STT service hints of which names to expect will also dramatically improve results, as the STT service was now completely unaware of this as compared to the first evaluation where the STT had to choose the best match out of a couple of names.

Participants have answered a questionnaire on their impression of the robot on a five points Likert scale. Scores were lower than expected, as most people responded neutrally on the given keywords except for 'pleasant' and 'friendly' which scored relatively high. This means we succeeded in giving a pleasant social robot to work with and improve people's mood, or at least not decrease it. However, the goal of letting users think they have control because they can say whatever they want to the robot instead of constricted answers has failed with a low score of 2.5. This could be explained with the design of the dialog for appointments, as the robot demands answers to the given questions and does not stop asking until it is satisfied. There is no comparative data from the first evaluation to see whether the score would even be lower there, and it is thus unknown whether this score is an improvement or not.

Comparison with first evaluation The feedback score has improved compared to the evaluation at the Health insurer. Although some people complained that the interaction took too long, we see that the average time to finish it is 2:50 minutes, not far from achieving the same result with a human receptionist. The biggest downside of the current implementation is the lack of support of the Dutch language, which significantly influences how the interaction went. The dialog contains two steps where a name is required, but Dutch names are seldom understood. This frustrated users resulting in them walking away before finishing the interaction or giving lower feedback due to this. While this is a technical limitation which can be fixed when the Dutch language becomes available (or by using another cloud service provider which supports Dutch such as Google), it does show how important it is to have multiple input options such as the tablet.

A comparison between the two evaluation is displayed in Table 6.4.

Metric	Exp 1, %	Exp 2, %	Comments
Feedback score of 4 or higher	79	95	Given at end of interaction on the robot
Use of speech (vs tablet)	52	86	Counted over all interaction steps
Client experience	75, n= 56	71, n = 14	Are the users positive about the use of the robot in general
Does it fit the company?	64, n= 56	93, n = 14	IBM n=14, of which four answered 'yes, but only if Dutch'.
Proactive engagement	43	32.4	Do people walk to the robot by themselves or after a nudge from the receptionist

Table 6.4: Comparison in numbers between both evaluations

The feedback scores given in the second evaluation are high. This is however mostly because of the formulation of the metric (score is four or more), since at IBM users rarely give a 3 star feedback as can be seen in Figure 6.2 where the distribution leans to 4 and 5 star mostly. This could also be because users walked away before giving feedback more often if the interaction did not go well.

A goal of the cloud-enabled prototype was to increase usage of speech instead of the tablet by offering broader speech input possibilities. This indeed resulted in a substantial increase in usage, possibly also as the clickable buttons to answer on the tablet were less prominent visible as with the first evaluation. The location at an IT company and age could also have contributed since the userbase is probably more in contact with speech technology in their daily lives. However, from speaking with the users afterward and by observing, it is also clear that users must be steered in the right direction on what to say as otherwise users will expect to the robot to be able to handle almost everything. Displaying example sentences to say at the beginning of the conversations on the tablet helps the user with this, as well as during other decisive moments in the dialog, as was also found in the related work.

Since IBM is a tech company, it makes sense that the score for 'does it fit the company' is higher than at the Health insurer, also because of the age differences in participants. Next to that, the sample size is much smaller. Remarkably, without knowing the background of the research, one participant mentioned 'that this robot works here -IBM- obviously, but certainly not at, for example, a Health insurer'.

The lower number at 'Proactive engagement' might be explained by the difference in positioning and type of customers walking into the office. Whereas at the Health insurer, the robot was positioned directly in the walking path to the reception, at IBM, this was just aside from it, so people might not directly notice it. Also, at IBM, a couple of times, groups of 10+ persons walked in simultaneously, whereas this was one person at most in the other case. In both instances, it was observed that younger people (<40) tend to approach the robot quicker compared to older persons.

Overall, looking at all numbers, users seem to like the second prototype better. This is mainly because of the unrestricted speech input offered by the cloud-enabled prototype. However, this is also one of the biggest downfalls. When the dialog agent does not know how to handle the given question or correct transcription fails, users get frustrated quickly. The prototype from the first evaluation using the Pepper Toolbox is more robust, and even when offering less functionality, this prevents disappointment. Also important to mention are the costs; where the first prototype is free for use after the development of the software, this does not hold for the second prototype where there are monthly costs and possible needed updates to the software if the Cloud service changes.

6.9. Conclusion and summary

The second evaluation of this thesis utilizing a cloud-enabled hospitality robot as a receptionist in the IBM client center has been described in this chapter. New software has been developed, which uses IBM Cloud services and infrastructure to perform the logic and process sensor data of the robot. This software is used to see how this changes (the quality of) an interaction with a user compared to the other prototype without these functionalities. We also learned what hurdles there are to build the software this way, what changes in the development approach and how users experience it.

A total of 61 unselected people participated in the second evaluation, which lasted for six days. Data on the interaction has been gathered automatically and by observing. With this, we know technical data of the conversations such as timings, state changes and modality of communication per step but also which user behavior influences the performance of the robot to finish the interaction as intended. For qualitative insights, some participants were also asked to perform a questionnaire.

Two pilots have been held before performing the main evaluation. One in an office room without noise, where mostly improvements in the dialog and interface have been made. The second at the reception itself, where the evaluation protocol could be tested and the effects of the final environment where the dialog was build for. Here the observation was made that Speech to Text using a single microphone is not workable as there is too much random text input due to surrounding noise and conversations at the reception. This led to the use of all Pepper's microphones in combination with a beamformer to filter the audio coming only from the user's direction. This has resulted in much-improved Speech to Text accuracy and precision, with the downside of a slightly bigger delay in processing(300ms+). It is, however, a requirement for using the robot this way in such high noise environments, preventing lots of frustration with users.

With most improvements from the pilots implemented and the protocol confirmed to work, the final evaluation could start. The results show improved performance in multiple categories compared to the first prototype, but robustness is still low. The feedback given by users is more positive, with an average of 4.45 out of 5. Users take an average of 2:50 minutes to complete the interaction and mostly use speech to communicate with the robot (87%).

As the discussion shows the concept of a cloud-enabled robot works and can significantly improve robot performance. The biggest downsides of this method are costs and currently also the robustness of the system as a whole. The latter can be improved significantly with more development of the software as the cloud services themselves are rarely at fault. Whether the costs are worth the increased performance is a decision varying per company and are expected to decrease over the coming years, making this cloud-enabled toolkit more interesting as time passes.

7 Service/module metrics

The robot can be observed to determine how well it performs in an interaction, but quantifiable results are also needed to claim that a system component, cloud or NAOqi, works better. This chapter contains the setup and the results of performance and accuracy measurements for both systems so their quality can be compared per component, as is needed to answer Research Question 1. This includes metrics such as the speed of transfer, time to process data, accuracy, and others. The approach to these tests is included so others could perform the same tests, as these apply to other social robots. Next to that, these results are expected to be outdated relatively quick, and thus, the tests must be able to be rerun.

We will look at the specific services per modality; Audio, Dialog, and Vision. As performing an exhaustive test set would take more time and effort than available, Speech to Text and Face detection are discussed in more detail as these are expected to be used more extensively than others.

Unless stated otherwise, the tests use same materials as in the evaluations from the previous chapter (see Section 6.1); the pepper robot (model 1.8a, firmware 2.5.5.7) connected using 4g and IBM internet over a 5Ghz network. It runs the CASperSocket connected to Node-RED using the IBM Cloud Frankfurt availability zone while Pepper does not have any usual running services (out of the box) disabled. Tests start automatically after starting the CASperSocket on the robot.

7.1. Audio

For the audio modality, two components can be compared with metrics and functionalities, Speech to Text, and Text to Speech. Both can be tested for latency, and for STT, we also look at the Word Error Rate.

7.1.1. Speech to Text

Many metrics can be used to describe how well a Speech to Text system works. We will analyze latency and Word Error Rate as these are significant indicators, relevant to the cloud, and measurable for both systems.

Latency

What is to be investigated The latency of Watson Speech to Text against NAOqi ALSpeechRecognition is tested. Independent from the content of the returned response, the latency of the transaction is measured as the difference in time between the creation of the request and the arrival of the response. Latency arises from time to package the request (in this case audio data containing speech), optional time to convert the audio into another encoding, time to send the data to a server (being rerouted through several other servers) if there is no network congestion, time to transcribe the audio, time to send the results back and time to process the result. This is extra influenced by location and current load of the data center where the data is sent to.

NAOqi STT does not have to send to and receive data from an external server but has less processing power available. Low latency is important for pleasant interactions as otherwise, people may walk away or start repeating themselves before the robot can answer, resulting in other undesired behavior or leading to frustrations.

Setup

1. Record speech utterances increasing in word size e.g., first a sentence of one word and a pause, followed by two words and a pause, up to sentences of 10 words.
2. Start retrieving current dB levels on the robot every 10 ms with timestamps. This way, all time measurements take place in one location, and clocks do not have to be synced. A sudden rise in dB level indicates the start of speech.
3. Start the playback of recorded speech on a speaker, located 1 meter in front of and facing the robot, where a head would normally be. Also, activate both STT systems.
4. When a Speech-to-Text result is received from either system, save this with the current timestamp and STT source.
5. Match timings of large dB increase with the STT responses to calculate delay.
6. Repeat this test three times and average results.

As the Watson service also returns partial results (word by word early prediction before corrections, using for example sentence grammar), this can also be measured for latency. Also, the cloud STT service can be run with and without beamformer, changing the latency.

Discussion Although this test has not been performed, observations by the author show that all systems have a low enough latency to provide fluent conversation, but it is clear the beamformer adds a small delay (around 300-500 ms). This cannot be attributed to the cloud itself as the beamformer runs on Pepper itself. It is, however, a requirement to use the beamformer for the cloud in noisy surroundings, an expected use case for social robotics, and thus the added latency also counts for this component.

The transfer rate during audio transmission over the websocket is around 32kB/s and should thus not cause added network latency. If the network latency itself is large, this test will give false results and should thus also be tested.

EXTRA:
When performing Speech to Text test over 4G instead of Wifi, an observation was made that the delay of transcription results was growing over time. After a few minutes, the delay in transcript response could increase to a few seconds, and even to 30 seconds or longer after 10 minutes. A solution to let it work properly was restarting the stream of audio data to the cloud every few minutes. Otherwise, consistency in latency is not guaranteed. Whether this the robot's software fault, Node-RED or Watson Speech to Text service could not be determined for sure. Since this mainly happened on 4G, we suspected an issue with the connection itself, which after some research indeed showed a high latency of a few seconds (based on ping) once every minute or so, probably caused by a busy tower. The transcription service would add this delay every time to the results resulting in this behavior. A stable wifi connection is thus not required but certainly recommended to prevent this kind of errors and add robustness.
EXTRA 2:
Also on Wifi, the latency of these services is not consistent as they differ slightly per time of day. An example of this is the Speech to Text service, which was initially (during first development) instantiated on a data center in America (US South). Despite the latency added by simply crossing the ocean (200 ms latency [a]) the response was fast enough for holding a regular conversation with the robot in the morning (UTC +1). In the afternoon, however, with the East Coast in the US waking up, a variable and substantial increase in response time could be noticed, up to seconds of delay, unsuitable for fluent conversation. It is expected that, with the different time zones and thus different usage statistics of the whole data center, there was too much congestion by American usage of the data center in the afternoon to respond quick enough for robust usage. Transferring to a nearby data center solved the problem, as would using a private node in the cloud if available.

[a] `https://www.cloudping.info/` Latency to Amazon Web Services US-based servers

Word Error Rate

What is to be investigated To determine how well Speech to Text systems work in terms of accuracy, often the Word Error Rate is calculated. This WER is computed by comparing output transcriptions with previously determined reference transcriptions. The number of errors produced by this comparison typically belong to 3 categories: INSERTIONS where the output of the ASR outputs a word not present in the reference, DELETIONS (D) where a word is missed in the ASR output and SUBSTITUTIONS (S) where a word is confused with another one. The final WER can now be determined by $WER = \frac{S+D+I}{N}$, with N the number of words in the reference transcription.

Since the NAOqi STT system is keyword-based and does not support free speech while the cloud providers do, a comparison between the systems is made using the Keyword Error Rate (KER) metric. KER is similar to WER, but as the name suggests, it is based on keywords instead of full sentences. According to [35] there is not a significant difference between using WER and KER, except KER increases more rapidly than WER as the accuracy of the transcription deteriorates.

Setup For our test, we record speech first and play it back on a speaker to the robot to be able to perform the test again using the same data. This can be skipped if this is not needed. Follow this procedure to compare two Speech to text systems using KER (partly based on the STT test method given in [18]):

1. Create a list of sentences for people to read and write down various keywords from the sentences. Input these keywords in NAOqi STT to recognize, and, if the option is present with the cloud STT service, enter the same to-be-recognized keywords there.
2. Use a studio microphone in a silent room to record these spoken sentences. Instruct speakers to use their normal voice, thus without purposefully changing voice with speed, intonation or articulation.
3. Put the robot in a silent room.
4. Activate both speech to Text systems on the robot. Alternatively, one can run only NAOqi STT and record the audio from all microphones to a file, convert it to one channel, optionally perform beamforming or other audio improvements, and process it later with the cloud STT.
5. Start the playback of recorded speech on a high-fidelity speaker, located one meter in front of and facing the robot, where a human head would normally be when interacting. The keywords and transcriptions have now been obtained.
6. Calculate KER for both systems using:

   $KER = \frac{F+M}{N} x100$

   Where F is the number of incorrectly recognized keywords, M is the number of missed keywords, and N is the Number of Keywords. A lower percentage (usually) means a better performing STT.

Results This test has been executed using NAOqi and Google Speech to Text using Dutch language and may, therefore, differ from other languages and cloud providers. The test has been performed in collaboration with J. Sparreboom during his thesis on a beamformer for the Pepper robot [36]. Note that for this test no beamformer has been used but since the room is silent during testing (there is no noise to be filtered), this does not affect the numbers. The results are shown in Table 7.1.

Provider/Speech type	**No dialog**	**With dialog**
NAOqi	32,4%	34,5%
Google	27,0%	28,3%

Table 7.1: Keyword Error Rate for NAOqi and Google STT using the setup described above. 'With dialog' means speaking full sentences in turns with the robot and recognizing a word from that sentence. 'No dialog' means telling a larger story to the robot without robot response.

Interpretation From the table, it can be observed that Google is performing slightly better (around 5 percent) in both conditions, either with or without using a Dialog. When using a keyword-based recognition system, the choice between NAOqi and Google (or another cloud provider) should thus not depend on KER as the difference is small. Instead, this choice should be made based on other properties like the ease of use, costs, or amount of available languages.

Note that during this test, Google did return full transcriptions, yet these were not used except for the keywords in the text. The usage, accuracy, and performance differ when using general speech full transcript Speech to Text services.

Speech in noisy surroundings
The WER test is performed in a quiet surrounding. Various use cases, including our own, can be thought of where this condition is not the case such as a train station, airport, museum, etc. containing background noise. This has a significant impact on STT performance, as described in Figure 4.2.3. Although we did not perform this test for our use case, this section provides a possible test plan and test data sets to assess the performance of (free-speech) STT on the pepper robot. To evaluate Speech to Text systems challenges have been held over the last 30 years. Examples for STT challenges with reverberated speech from a distance (think of a lecture, meeting, or conversation scenarios) are RWCP-SP, CHIL, AMI, PASCAL SSC2, and REVERB, in quiet conditions [37]. Other challenges aimed at command scenarios in the presence of background noise are Aurora 2 and 4, HIWIRE, and DICIT.

The Third Chime Challenge [38] tries to combine this, aiming at high levels of background noise with speech recorded live in noisy environments. They describe a challenge for STT systems using a balanced dataset containing various working conditions. The dataset contains utterances from varied speakers; 12 US English speakers, six male, and six female, ranging in age from 20-50. Audio levels were held constant. This challenge targets the performance of STT systems in real-world commercially motivated scenarios, just as our use case does.

This same dataset can be used to test a robot's STT performance. A similar setup as for WER can be used, except the speaker is replaced with a speaker installation. Note that if the error rate is high and the application is a dialog system, the system may still be able to achieve an acceptably high completion rate since it can fix errors through clarification or asking confirmation.

7.1.2. Text to Speech

As stated in the intro, (the cloud version of) Text to Speech is expected to be used far less than Speech to Text. This is mostly because the built-in voices of NAOqi work well with no errors in pronunciation of input words and quick voice output after giving the command. Downsides to using NAOqi, looking at bare properties, are the simultaneous availability of languages (restricted to a maximum of two on the Pepper robot), the inability to easily switch between these or pronunciation errors in infrequently used words. The cloud offers many types of voices for both genders and some accents for more languages. Some services offer adaptations of voice (speed, pitch, emotion) or could even simulate a specific voice after training. Next to that, these options can be changed dynamically during interactions.

Although clear considerations can be made here for choosing one system or the other, detailed comparisons on specific aspects of TTS are not useful to help in this decision as the systems are so alike and work well. One comparison which is of more significant influence is response time (latency) from giving the talk command to the actual speech being heard.

What is to be investigated The latency from giving the command to speak and actual speech coming from the robot

Setup

1. Create text utterances increasing in word size e.g., first a sentence of one word, followed by two words, up to sentences of 10 words.
2. Create a button in the browser which lets the robot speak the text from a text field, just like the standard web interface of the robot.
3. Start an external recording of audio using a microphone, located next to the mouse.
4. Click with force (to generate a clicking sound) on the mouse button to let the robot speak, for both systems.
5. Analyse the audio recording (using a sound visualize, for example, Audacity [1]); the dB increase of the click can be seen as the 'start' command after which the next increase in dB levels shows the first spoken word, and thus a completed TTS.

[1]Audacity: `https://www.audacityteam.org/`

Discussion This test was not performed, however, observations were made by the author while building the system. Speech produced by NAOqi has a negligible delay, which is its biggest upside. Speech from the cloud is quick enough for most conversations but a delay is present, especially for larger sentences due to the buffer of audio data which has to be generated and received in full before starting to speak (the function to play an audio buffer is supported on the Pepper robot, but resulted in a kernel crash). Although some software fix could be implemented to reduce this delay, it will inherently be present with speech coming from a cloud service. Caching recurring sentences avoids this problem partly.

7.2. Dialog

Ascertaining performance of a dialog system requires more manual labor than the other components in the system. Spoken dialog systems 'performance', can be described as the ability of a system to provide the function it has been designed for[39]. Also important is usability, subjectively measured by metrics as user satisfaction or likelihood of future use, which are difficult to measure and are dependent on the context.

A variety of frameworks is available to measure the quality of human-system dialog interaction [40]. Subjective metrics can be used, such as perceived task completion, user experience, to be collected via surveys and questionnaires. Objective metrics such as word error rate, dialog length, dialog turns counts, etc. can be obtained through message logs. With Assistant, some of these objective metrics are part of the JSON messages with a response for the user so a system can act on this if wanted .

PARADISE is an example of a framework for automatically evaluating (the usability of) dialog systems[41]. It seeks to optimize a desired quality by making a linear combination of various metrics. For example, to optimize user satisfaction, metrics such as task success and dialog length could be used. Several models are created for predicting system usability (as measured by user satisfaction), which results show to generalize well.

In non-task-oriented dialog systems, it can be harder to develop robust evaluation metrics compared to task-oriented dialogs. It is not clear what 'success' means, and thus, task-specific objective metrics are not fitting. Subjective evaluations for appropriateness of responses can be much more meaningful in this case, leading to the development of coding schemes for response appropriateness and scoring in such cases [42]. Our cloud system is mostly task-oriented (as it has a goal to complete registering a visitor and notify employees after accomplishing this), but also offers non-task oriented dialog by giving general information or answering unrelated questions, which is also a general use of the robot. The main focus is task-oriented.

Another attempt to automate the evaluation of dialog performance is made by Georgila [39], who takes first steps toward developing low-cost evaluation metrics that are predictive of user perceptions about dialogue quality in the IOT domain, similar to the task-oriented domain. By low-cost is meant that these metrics should be based on automatically extracted features or, if this is not possible, rely on simple annotations that can be performed by non-experts in linguistics or dialogue. To do this, they calculate correlations between features and human ratings to identify which features are highly associated with human perceptions.

Automatically gathered features include: number of system and user turns per dialogue, number of total words from system and user per dialogue, average number of words per system and user utterance in a dialogue, and number of occurrences of specific words and expressions, e.g., 'yes/yeah/yep/yup', 'no/nope', 'ok/okay', 'alright/all right', 'good/great', 'sure', 'got it', 'no problem', 'sorry/apologize/apologies' etc.

They perform linear regression and derive a variety of dialogue quality evaluation functions. Results show that these functions are highly predictive of human ratings and outperform standard reward-based evaluation functions as discussed before. The best working functions are functions that include 'misunderstandings', 'system confirmation requests', 'system requests for more information', and 'conversational style.'

What is to be investigated The performance of a dialog system running on a robot, where performance is as earlier described defined as: the ability of the system to provide the function it has been designed for. Other items one can investigate is usability.

Setup How to perform this test depend on the end goal. For comparing two systems, one can gather and compare metrics calculated from the logs of a conversation. In the case of the NAOqi toolbox, one must first save this data. Metrics to use are (largely from [39]):

- Occurrences of silence
- Occurrences of specific words and expressions such as 'I mean/I meant', 'I said', 'no/nope' or similar

- Number of system turns per dialogue
- Number of user turns per dialogue
- Number of all turns per dialogue
- Number of system words per dialogue
- Number of user words per dialogue
- Number of all words per dialogue
- System does nothing
- System does something invalid
- Task success (in our case this is performing the happy flow)

When comparing the same system with variations, it depends what to optimize on. To optimize user satisfaction, metrics such as task success and dialog length could be used.

In short, the setup of this test is:

1. Save the full interaction transcriptions. This can also be from simulated users.
2. Gather dialog metrics suited for the dialog type and use case from the transcriptions.
3. Compare the results and see which systems perform better depending on the goal (e.g., should the conversation be kept short, or do we optimize for task success).

Alternatively, use the manual annotation scheme (from [39]) if automatic data gathering is not possible. Gathering subjective feedback from users could also be used as a measurement. One should be aware that speech recognition errors can be a source of noise in all interaction, regardless of the dialogue context, influencing these metrics.

7.3. Vision

While NAOqi can run functions as face/object detection/recognition fairly low level in the system, having direct access to the memory values containing raw camera data for the algorithms to use, this is not the case for similar functions in the cloud. For the cloud, the camera data must often be converted into a compressed image format by the robot (requirement of services), send over the websocket if it is not busy, be processed by one (or more) external servers and send the response back. As this pipeline influences all cloud vision metrics independent of the used function or cloud provider and takes extra time to process, we first look at how fast images can be sent to the cloud with the metric Frames Per Seconds (FPS).

Next, specific metrics can be considered for face/object detection/recognition as can be seen in Subsection 7.3.2.

7.3.1. Frames Per Second

When processing a video or image stream in the cloud to determine whether faces or another object can be seen, these frames must first be uploaded over some internet connection. While generally nowadays an internet connection is fast enough for most media applications (think of a highly compressed UHD video stream as utilized by Netflix running at a minimum throughput of 25Mb/s), there are other factors to take into account. Upload speed can be influenced by for example a busy router, slow networking hardware on the robot, a full or busy wifi channel (think of 2.4Ghz vs 5Ghz and their various channels), internet line congestion, a slow responding server or even a busy mobile tower when using a 4G internet connection.

What is to be investigated The Frames Per Second with which the robot can upload images taken by its camera. Thus the amount of times in one second using the pipeline: *Take image with camera -> (convert to .jpg) -> upload over websocket to Node-RED -> get confirmation of receival on robot.* An image can be sent whenever the websocket is free, and thus before receiving confirmation of earlier uploads.

There are various ways to retrieve an image or stream of images (video) of a Pepper robot, of which the following are the most obvious (ordered slow to fast):

- Using NAOqi: ALPhotoCapture proxy: High-level approach, requiring one built-in function performing all operations.

- Using NAOqi: ALVideoDevice proxy: lower-level option which works faster but requires more developer effort by doing manual image conversions. One gets the raw bytes of the image from the Pepper camera and need to convert it to a readable format using OpenCV (can be installed locally on the robot) for cloud providers to understand the data.
- Using GStreamer [2] (or another method working on the same low-level system access as NAOqiOS itself): This method directly accesses the camera's on Pepper from the operating system. This disables NAOqi camera usage, meaning standard modules do not have access to the video feed at the same time and cannot perform functions like automatic face tracking anymore.

Setup The test is automated on the robot. First, the highest resolution is set for image capturing. The image resulting from the first (ALPhotoCapture) capturing option is uploaded over a websocket to Node-red. Confirmation of receiving the image in Node-RED is then sent back to the robot over the same websocket. The number of confirmations is then counted within a 1-second window bin. The test is run for 10 minutes, followed by the same setup on lower resolutions. This minimizes temporary variations such as short high CPU usage on the robot, a congested internet connection or other outliers. During the test, no services which normally run on the robot are disabled, so the websockets, alive behavior, basic awareness, and even some small behaviors, etc. are kept running as usual.

This test is performed twice to know the range of possible values. Once using a 4G connection on a 2.4Ghz wifi network, and once on a wired internet connection on a 5Ghz wifi network connection.

Results The results are displayed in Figure 7.1. Overall the FPS is stable with a deviation of around 1 for all resolutions with a small sudden peak for 640*480 on Wifi.

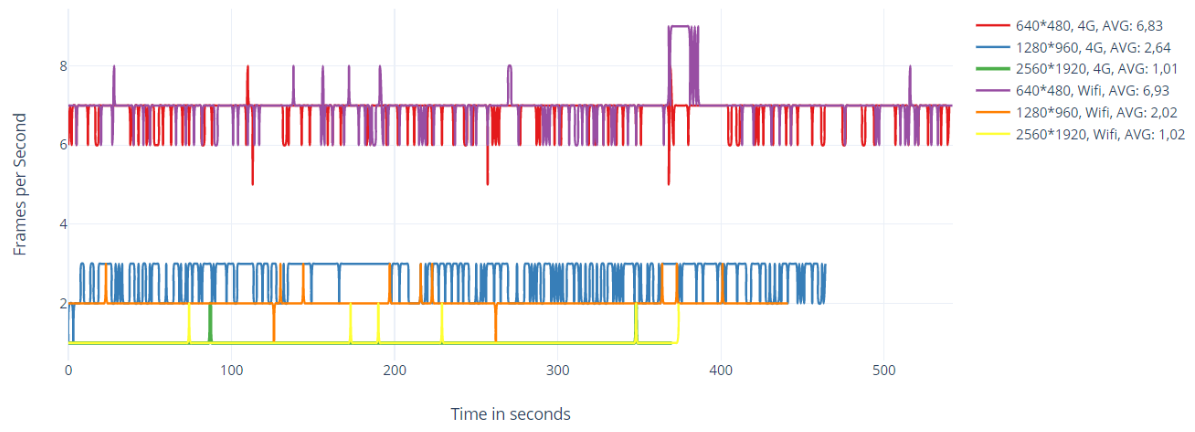


Figure 7.1: FPS over 10 minutes of sending images from the robot to a cloud server and receiving confirmation of that. The same test has been run for different resolutions on both 4G and Wifi.

The transfer speeds during this run averaged on 850kB/s for 2560*1920px (around 1.5MB per image), 550kB/s for 1280*960px (450KB p/i) and 450kB/s for 640*480px (110KB p/i). When running a speedtest from the robot[3] on the wifi connection itself is able to reach 1358 kB/s download and 871 kB/s upload speed. For 4G, this is 385 kB/s download and 887.5 kB/s upload.

Interpretation The achievable FPS is high and constant enough for most social use cases with the robot. Especially the middle resolution of 1280*960px offers reasonably high quality for advanced image processing against a stable minimum of 2 FPS and even regularly achieves 3 FPS over 4G. The middle resolution also performs better than the documentation suggests with 2560*1920px and 1280*960px being rated at max 1 FPS and for 640*480px and below max 30 FPS. Different CPU's and memory amounts are available on the different Pepper models, so this one example may not work the same for all. Even with later image processing steps in the cloud, this means a reliant pipeline for most interactions.

[2]gstreamer media processor: `https://gstreamer.freedesktop.org/`
[3]Speedtest from terminal using speedtest-cli: `https://github.com/sivel/speedtest-cli`

With the CPU load being around 40% during this test, and memory using 600 out of 4000 MB available, hardware performance does not seem to be the limitation for achieving a higher FPS. We therefore think, but cannot confirm, the camera or storage is the limiting factor.

If needed, performance could be improved by using one of the other image capturing methods as described at the beginning of this subsection, as currently the slowest but the easiest option is used.

7.3.2. Object, face and person classification and recognition

The cameras of the robot provides the programmer with high confidence information about the environment. They can confirm a person is standing in front of the robot, compared to, for example, the lasers giving information that 'something' is near the robot. The algorithms used to analyze the images can vary from low computational (Something is moving) to high computational (recognize an employers face moving from a specific location to somewhere else). What can be processed locally on the robot within a specified timeframe is limited, yet there is no increased latency due to the cloud. Which component -local or cloud- to use is thus dependent on the use case but could require performance numbers to base the decision on.

For this thesis, the tests to obtain these performance numbers are not performed due to the broad scope, and we can not be exhaustive. Instead, we do provide a structure to obtain these numbers. The structure for testing these vision techniques (object/face/person classification and/or recognition and/or tracking) is comparable, and thus, only one method is given for object detection.

Object detection test method To obtain comparable metrics for object detection, a similar method as with Speech To Text WER (see Subsection 7.1.1) can be used. This means using two pipelines: (1) perform object detection through NAOqi and (2) send (and save) images for object detection to the cloud for analysis. Both pipelines are then run simultaneously while objects are being presented to the robot's cameras and the output of both systems is saved with a timestamp in a log on the robot.

After running the tests, one can determine performance (time to obtain the result on the robot) by comparing timestamps for presenting the same object. Accuracy of detection can be determined by comparing the timestamped results against the known value; was there an object at all, is the correct location (and/or orientation) returned? In the case of face recognition; did the system return the correct name belonging to a face, and does it consistently (and what percentage of total time) keep reporting this person and not someone else? Per use case of the robot, it differs which metric for accuracy is more important to look at.

Stepwise, the test is performed by:

1. Put the robot in front of a non-changing background. This does not have to be a white wall as that is usually not a real-life situation the robot would work in, but for comparability, the background must not change in order to test with the same conditions.

2. Put the robot in a mode where the head and body does not move.

3. Enable both pipelines; the Naoqi toolbox and the cloud service.

4. Save the video of the cameras or make an external recording to compare the data against later.

5. Present an object (or face) in the vision of the robot's cameras. Let the robot save its result with a timestamp. Optionally, a monitor positioned in front of the cameras can be used to display more varying faces or objects to the systems.

6. Stop the test and compare timestamps of the result for speed and the accompanying data for accuracy. Which metric is used is dependent on the relevancy to the use case.

For face detection, the WIDER FACE dataset [43] can be used as a benchmark, offering a labeled set of faces with a high degree of variability in scale, pose and occlusion. For face recognition, including names and different pictures of the same person, the Labeled Faces in the Wild dataset can be used[44].

Again, circumstances differ, as the software from the Pepper Toolbox cannot detect as much as the cloud service equivalent. Other software could be run on the robot offering more functionality for these capabilities such as OpenCV. However, the limited processing power offered by the robot's hardware does not change and limits the choice in algorithms to use. For the cloud, the complexity or amount of algorithms to use does not matter for the robot's performance as it is only sending images, which the server then takes care of by distributing it to other servers needed for further processing.

7.4. Concluding

Our results provide several indications of the performance of various system components. We have aimed to illustrate the overall approach that we propose in this thesis, not to be exhaustive (which would require significantly more work to evaluate other system components).

We found that for audio, specifically Speech Recognition, the difference in keyword recognition rate between the cloud and NAOqi systems is not significant and that this factor is thus not relevant for comparison, while other properties such as latency, the ability to give free-speech transcriptions or amount of languages are relevant. This does not hold for when the robot is in a noisy surrounding, where the cloud variants (including beamformer) perform significantly better, however, no quantifiable data is available for this. Text to Speech is harder to test, but we found that the latency of cloud services can be too long for some use cases.

For Vision, FPS has been tested. Although our implementation for the test uses the slowest image capturing option on the robot, a stable output of FPS high enough for most use cases (>1, >2, >5 FPS for maximum to lowest resolution) can be achieved on both Wifi and 4G.

The approach discussed in this chapter has several limitations, which we briefly discuss. Relevant metrics and methods to obtain them are described, but it does not show the whole picture. Several extra modules are missing from the test scope, e.g., sensors as touch or laser/sonar. These do not produce such extensive amounts of data compared to, for example, video or audio.

What is also missing are results for when modules are run simultaneously. What happens to the throughput, do multiple modules slow down or only one? Do the number of frames drop? Can a 4G connection still hold up?

Generally, looking at the speed of data transfer (900 kB/s for high-quality audio and video simultaneously) any regular internet connection will succeed in transferring all at the same time. On slow connections though, it might be smart to prioritize audio, touch and general commands over video, which could be done by lowering the frame rate or the use of auto-throttling.

Choosing which type and provider of Speech to Text to use depends for a large part of the use case. On the one hand, one has low latency keywords recognizers as NAOqi offers or low latency free speech recognizers and classifiers as Watson. Google offers high accuracy free speech with the downside of higher latency. Watson, on the other hand, gives more accurate transcriptions with background noise. Since the cloud services continually update and improve on latency, lower prices, and accuracy, the advantage is expected to shift more to the cloud over time. NAOqi offers free operation for all modules, which may be enough, more stable, and easier to implement for simple use cases.

8 Cost structure and estimate

Costs are an essential aspect for businesses to base decisions on and an analysis of this is relevant for answering Research question 4. This chapter holds the analysis of costs (or effort) made for *developing* and *running* both systems, an all-NAOqi, and an all-cloud robot, as used in our use case. An indication of the running costs per month for one robot can be deduced from this.

The numbers and calculations in this chapter are mostly a paper exercise. Though based on real data and educated assumptions, real usage is hard to estimate. This can be seen later in the chapter with running cost calculations where actual usage -and billings- are much lower than calculated.

8.1. General approach

The structure of this chapter is as follows: for every component (Audio, Video, Dialog, Logic and Other) we look at the cloud costs structure, total costs from running the evaluation at IBM, and an extrapolation to monthly costs for running it on the cloud, next to the costs for development of both systems. We also discuss possible options to reduce running costs. The chapter ends with a discussion of how total costs for the complete lifecycle can be extrapolated from the data gathered before.

All cloud providers offer varying services and capabilities with their own unique pro's and con's, for various prices in various price packages. As it is hard to formulate a cost indication for all these providers, we focus on the overall cost structure. The product created in this thesis is fully based on IBM Watson services, which is why the current pricing for IBM Cloud (standard plan) is used in this chapter.

Almost every IBM service offering has three pricing plans; *lite* plans are free and offer a small amount of processing time, sometimes with limited configurability. This gets one started but is usually not enough for production environments. *Standard* plans are for small to bigger businesses and often works with tiers getting cheaper when used more. Custom models (training) and other add-ons can be used at extra cost. The *Premium* plans allows for options as Private Usage and Training Data, Storage in an Isolated Environment, High Availability and Service Level Uptime Guarantees, Single Tenant Environments and Mutual Authentication. This is all at a bigger cost but may be required for security or regulatory reasons.

For all components it is assumed that the local running version (from the Pepper Toolbox) is free to use after implementing/developing it: except for possible security fixes or maintenance because of a software update of NAOqi or external services used, there is no price tag on using this local solution for all of its operation lifespan. This will not be mentioned again in the following sections but is important for comparison.

For estimations where time cannot be used as a metric, we use the range of 30 - 100 interactions per 8 hour day, as to offer a medium intensity reception and one where the robot is handling a user almost all time.

8.2. Audio

Transcribing audio is not costly nowadays. For Speech to Text, the price is determined by the amount of minutes of audio data sent to the server, whether this contains speech or not. For Text to Speech, the price is calculated per thousand characters.

8.2.1. Speech to text - Costs

The first 250000 minutes (Tier 1) costs €0.015 per minute, followed by €0.0113, €0.0094, and €0.0075 per minute for the following tiers increasing again after 250000 minutes of use (per month). Exceeding the first Tier roughly corresponds with six robots sending speech data simultaneously for a full month. When one wants to use a custom language model, the price per minute increases with €0.0225. For the premium plan, the costs for use are the same, next to a fixed rate of €3,761.00 per instance.

The cost of using the STT for one full workday (eight hours) is €7.20. The costs of running the evaluation, during 18.5 hours and short testing, is €25,-.

8.2.2. Speech to text - Cost reduction

Looking at the price scheme, there is one obvious solution to reducing costs: sending less audio data containing no speech. The most extreme solution is only to listen (send audio data) when user input is expected or needed, or even using the local STT for easy speech utterances and only use the cloud service when full sentences or hard to transcribe words are expected to be used.

In the CASperSocket, audio streaming is activated from the moment a face has been detected, and the scenario is started until the scenario is stopped, and the user has walked away. This was done to prevent possible errors when setting up the streaming, such as initialization errors or starting to listen too late and thus missing vital speech from the user. For cases where no visual recognition is (or can be) performed as a starting method, a sudden increase in audio level (dB) on the microphone can be used as a starting signal.

The trade-off in interaction quality is not missing input text from waiting users versus not streaming all day. If the developer activates the streaming the moment a user is near, but no interaction has been started yet, one can make sure responsiveness if high as soon as the user starts talking.

8.2.3. Text To Speech - Costs

TTS is charged per character. The first 10.000 characters per month are free, followed by €0.0151 per thousand characters. Custom TTS dictionaries containing their own word pronunciations for up to 20K words can be used for free.

For the month where the evaluation was performed, 49654 characters were send, resulting in €0.08 of costs. It should be noted that Watson TTS was not used as the standard voice method. However, if the full scenario would be performed using cloud TTS, the costs would not go above 3 cents per interaction. With 30 interactions a workday would not exceed €1,-, or €3.00 with a 100.

8.2.4. Text To Speech - Cost reduction

Although the costs of TTS are not high compared to the other components, smaller companies still might want to reduce costs. There is almost no trade-off for interaction quality when implementing this. If a local TTS is available on the robot, which is the case on Pepper, this could be a good alternative to using cloud TTS. Only for special cases, for example when a specific voice, language or voice transformation is needed, cloud TTS can be used (if allowed by the robot provider) and local TTS otherwise.

If the choice is made to use cloud TTS for everything, still a cost reduction can be achieved, which even increases responsiveness. If a sentence is recurring, for example, "Hello, welcome to our company, how can I help?" the audio response can be cached. Before sending the request to the TTS service, a check is performed whether this sentence has occurred before and can be played immediately without even contacting the cloud service. The additional advantage is less latency in starting to speak, as the audio data generation and transfer steps are skipped. Note that this does not work for dynamic sentences when the sentence is almost the same but contains a name or time, for example. This requires the regeneration of the whole sentence.

8.3. Video

The cost calculation of the Watson Visual Recognition Standard Plan is event-based. An application can send ten images to the server to analyze, but if only one of them returns the presence of a face, only that event will be charged on the account.

The price for one 'general tagging event', in our case the *face* classification event is €0.001504. For specific (trained) faces this increases to €0.003 for recognition and €0.0752 per training image. We can calculate the costs of running it one workday with the following assumptions:

- video (a stream of images) is sent at 2 FPS (see Subsection 7.3.1 for details why).

- A total of 2fps * 60s * 60m * 8h = 57.600 frames are sent on a typical 8-hour workday.
- 20 percent of the images contain one or multiple faces in the images; this is an estimation based on observations of the evaluations performed in this thesis and resembles 30 interactions per day. For the 100 interactions per day, we use 60 percent.

The total amount of frames to be paid per workday is then 0.2 * 57.600 = 11.520 frames. Multiplied by the costs per face classification event (€0.001504) results in €17.33 per day if in continuous use and with mentioned assumptions. For 100 interactions, this is €51.98 per day.

This does not correspond with the month of the evaluation where the *monthly* costs are €37.87 (€1.89 per day). An explanation for this can be that during testing, the 'face activation' was simulated and thus faces were not seen that much. Also, the robot was not running continuously, but only a couple of hours per day on average.

8.3.1. Cost reduction

Several ways can be thought of to reduce costs for visual recognition. This includes reducing the frame rate and/or not sending images at all with the use of simple local sensor processing.

Frame rate Since one does not pay for sending images itself if no face is present, it may not help much to reduce the frame rate. This also helps in making the trade-off on the speed of recognition when someone arrives in front of the robot and starting an interaction. Reducing the FPS send to the server increases the time to respond to a waiting customer, which should most likely be avoided "at all cost" to prevent loss of engagement.

When a person is recognized in the image (and thus billing starts), it can be an option to decrease the FPS after this. The system can determine whether the user is still engaged by several factors; speech input, the presence of an object in front of the robot (using laser, 3d camera) and detecting a face using the cameras. While the latter is very indicative, it does not necessarily need to be quick. When running on 2 FPS, stopping an interaction after half a second, when no face is detected, is dangerous; it could be the user is still present, but the robot just looked away or just moved out of sight. A safety interval of a few seconds (in our case six seconds) is usually built in to decide the engagement is lost, and thus 0.5 FPS (every 2 seconds) would work well enough. This reduces the images to be paid for with 75 percent while functionality is almost unchanged. The numbers here are arbitrary, but the reasoning holds for similar FPS decreasing solutions.

Local sensor activation In the case that other cloud services do charge for every image send, an alternative is to locally run a simple (low computational power) image analysis technique on the camera output, such as movement detection, sudden light intensity changes or similar. One can refrain from sending images to the cloud service as long as this simple image analysis returns nothing.

Tradeoffs Both methods do not have a large trade-off with quality; when high precision detection is needed, one uses the cloud service activated by using beforementioned methods. Otherwise; image sending is halted or reduced in speed. One important downside is an increase in the response time of the robot detecting people in front of it. It depends on the application how important this is, but obtaining engagement from a person walking by is harder if the person has already passed at the time of response, resulting in a missed opportunity for interaction.

8.4. Dialog

The pricing plan for Watson Assistant, the dialog system used in CASperSocket, is based on the number of API calls to the service, or, easier said, the number of user input sentences requiring some answer or action. The first 10.000 messages per month are free, after which one pays €0.00188 per API call.

The interactions of the evaluation had an average turn counter of 13, resulting in €0.02444 per interaction or €0.73 per day with 30 interactions or €2.40 with a 100. This only holds if the first free 10.000 API calls are already passed.

The month of the interaction, including extensive testing, resulted in €0.84 of costs. To give one general number, one could state that for one robot, costs will not exceed €1,- per day.

A cost reduction on this is not reasonably relevant as the costs are already low and insignificant compared to other services, but also hard to achieve programming-wise. Every API call needs a context which differs per interaction, which makes caching not useful. Reducing the number of API calls cannot be done unless the conversation is made more efficient using fewer dialog turns.

8.5. Node-RED and other

Node-RED, which in our case performs all logic processing and data distribution for all previously mentioned components, is an open-source web solution without costs when running on an own server. When running in the (IBM) cloud costs for processing power and managed hosting is involved.

The metric in which this is billed is 'GB per hours per month'. This means that the costs depend on which virtual machine is used. Several choices can be made in the IBM dashboard, but the type of machine is usually decided by the amount of RAM a user wants. An increase in RAM often means an increase in CPU power. It is billed with €0.0526 EUR/GB-Hour, getting the first 375 GB hours for free.

For CASperSocket, two pre-build managed (automatically maintained and configured) virtual machines in the cloud were used. One handling all logic and traffic (Main) and one purely used for a link to the Visual Recognition service (VisRecog), as restarting the stream of images to Visual Recognition sometimes crashed the Main VM resulting in a motionless robot. Splitting it in two VM's solved this problem and added significant robustness to the system. The two virtual machines run with different configurations:

- Main, 4GB RAM, monthly cost: €75.74
- VisRecog, 2GM RAM, monthly cost: €37.87

This results in a total of €113.61 per month. Whether the robot is used or not, the VM keeps running and allocating RAM, which the cloud provider cannot use otherwise. The total cost deviates only with about €5,- in the months after the evaluation where the usage was lower.

8.5.1. Node-RED - Cost reduction

The easiest solution for reducing cost is disabling the Virtual Machine when not in use. Only activating it during regular work hours means only 160 hours out of 730 in a month, decreasing the cost by 78 percent. There is no trade-off in interaction quality from using this method.

Other options are using only one VM for all components or decreasing the RAM, effectively getting a cheaper VM. The robot in the evaluation has run on this lower configuration, but a drop in robustness could be noted, probably as a result of not well-tested code (resulting in buffer overflows) which we could not confirm. We expect fixes in software used (Node-RED updates, updates on Node-RED modules) could make this a standard solution, as we now over-provisioned the machines as a fix.

Other options are to use other cheaper VM's, a company-provided server, or even a Raspberry Pi as alternative hosting locations. Node-RED itself is not computationally intense and runs on many platforms. A downside of this could be stability, maintainability, and response/processing times, but this all depends on the chosen solution, and it is hard to make a clear statement about this in general.

8.5.2. Other sensory data

Next to the sensor data for previously mentioned components, several other sensor types can be used in interactions. Think of touch, sonar, laser, depth sensor, or actuator movement values (current body position). Other data send back and forth from the robot to the cloud (logic) server is tablet content and touches and command messages.

All these data types are in the form of small JSON messages. Due to the nature of the sensors, the quantity is very high as they need regular updates. Using the setup of the evaluation, no extra costs are billed for the data transfer as this is included in the Node-RED instance. If some cloud provider bills for the use of the connection (bandwidth) it depends whether this is message-based or volume (size) based if it is significant for the cost aspect. Looking at total costs, the costs for processing these sensors can be neglected.

8.6. Total cost of ownership

Next to running costs, also development and maintenance add to the Total Cost of Ownership (TCO). This is a financial estimate intended to help buyers and owners determine the direct and indirect costs of a product or system during its intended lifecycle. This section contains the calculation of the TCO.

8.6.1. Monthly costs

With the costs from the components known we can calculate a total running cost per month. All costs mentioned in the previous sections are shown in Table 8.1. The table shows the calculated costs for one workday of usage, meaning the service is used for the full workday without smart cost reductions and thus not only running when a user is interacting with the robot. Only Video (Visual Recognition) takes the assumption into account that 20% of the time a person is in view (and thus billed). For Node-RED, no calculation is made as it is not visible in the developer's dashboard what activity attributes to the total cost, and these costs do not deviate more than €10 in the months after the evaluation. See the specific section for all other assumptions.

The (extrapolated) costs for a full month (business days) are displayed next. As an indication of how theoretical calculations can differ from real-world usage, the last column shows the actual billed costs for the month of the evaluation using the beforementioned cost reductions. Usage here was equivalent to 10-20 interactions per day and not representative for full workdays as used in the calculated columns, which is more relevant for real-world usage numbers as is aimed at for answering the research question.

Component	**per Workday (8h)**	**per Month (20 workdays of 8h)**	**Actually billed (Month)**
Speech to Text	**7.2**	144	6.45
Text to Speech	**1**	20	0.08
Visual Recognition	**17.33**	346.52	37.87
Node-red	5.68	**113.61**	113.61
Assistant (Dialog)	**0.73**	14.6	0.84
Total	*31.94*	*638.73*	*158.85*

Table 8.1: All numbers are Euros. Bold numbers are calculated and retrieved from the respected section for which certain assumptions hold.

We see that the Visual Recognition service is responsible for more than half the total costs with the calculation. Server costs also attribute much to the total. This is also reflected in the actual billed costs, yet here a significant cost reduction is achieved for Speech to text.

We now have an indication of the actual costs per month for running a cloud-enabled social hospitality robot and how the components compare in functionality versus price. This does not mean the same ratio holds for every solution for this use case as implementations, usage, and cloud service provider likely differs.

8.6.2. Development costs

To determine development costs, one needs the hourly rate of a developer and the number of hours to work on the project.

Hourly costs The hourly costs of a developer vary greatly. Companies could build the product in-house or outsource development to other countries (in other continents) and save significantly on costs, with the trade-off of varying quality and time of development. To calculate product development costs, a realistic hourly rate is needed. To obtain this, we take the average hourly rate of (medior, python) software developers in the Netherlands. Several websites offer an average of this but provide different numbers. As we only need an estimation, we average the results: Glassdoor.nl €26.80, Payscale.com €17.97 - €49.43, codementor.io (Python-specific) €54.3 - €71,-, Average €43.96. With little overhead costs, we settle on €50 for an easy to work with number.

Hours of work - Effort estimation model Hours of work on both projects have not been steadily tracked, but The Health insurer application (using the Pepper Toolbox) took about four months to develop (from concept to tested product) and the IBM application (using the Watson Toolbox, having no base framework) about five months.

A model is needed to estimate the development time, as the trade-off for toolbox type needs to be made before actually developing the product. A simplified cost estimation model based on COCOMO is used to approximate project cost [32]. The older effort estimation model COCOMO [45] has been extended, adapted, and improved multiple times since 1981, resulting in better estimations for current software with new development languages and methods. However, recent versions have more than 20 parameters and are thus not useful at an early conceptual phase if one does not have a logical approach for specifying the input values.

The simplified model's R^2 value shows that 89% of the variation of software development effort has been explained by the regression[32]. The model has been tested against empirical data collected from 317 projects from 2004-2013. Although the model is basic and not entirely accurate, it is easy to use and therefore a useful estimator for our goal. The provided equation is applicable for project size ranging between 1 and 842 *KESLOC (Product size in thousand Equivalent Source Lines of Code)*, 12 different application types and different business sectors (military, government, and commercial). We apply the equation with the 'Intelligence and Information Systems' application type as provided in the paper. The equation to calculate PM *(Engineering Labor in Person Months)* then results to:

$$PM = (2.047xKESLOC^{0.9288})x1.917$$

This equation should only be used for early estimates and if more data is available it is recommended to use the full recent COCOMO models using more parameters.

Applying the equation to the projects from this thesis, using the LOC as described in Table 4.3, we obtain:

- IBM, Watson Toolbox, 2392 LOC (2.392 KESLOC): **PM=8.8. Actual: 5 months.**
- Health Insurer, Pepper Toolbox, 4660 LOC (4.660 KESLOC): **PM=16.4. Actual: 4 months.**

These numbers are significantly higher than the actual development time. The IIS category already has the lowest multiplication factor of all categories in the paper. Several factors could explain the difference. The paper does not specify which programming language is used and python is usually more concise (lower LOC) than, for example, the same functionality in C. Also, for both products, a lot of testing has been performed, but no automatic tests have been implemented. The products also have limited documentation, and the code is not fully production-ready. Also, for the Pepper toolbox, the code has duplication and syntactic sugar. The amount of code lines is thus higher than with usual projects, as earlier described in Section 4.3. A conservative approach for estimating the Lines of Code is thus advisable for Choregraph based projects.

Total development costs Using a simple but clear calculation, we can calculate the (hypothetical) costs of both projects by multiplying the number of hours with the hourly developer costs. Since we know actual development time in this case instead of the COCOMO based method (see bold text above), we use those numbers.

*IBM: 4 months (4*20d*8h = 640 hours) * €50 = €32.000*
*Health Insurer: 5 months (5*20d*8h = 800 hours) = €40.000*

A side note is that IBM's development time includes building a Framework to run on, which can be reused for other scenarios and use cases. Looking at scenario only if the framework was already there, time and costs would be more than halved. Not developing the product in-house but externally, where a framework is available, could be cheaper in this case, especially if it is not expected to reuse the framework for later projects.

8.6.3. Maintenance

As to keep the product running during its intended lifetime, maintenance is required in case new features are needed, API's are updated, the cloud provider stops a service, the robot receives new firmware, etc. Estimating maintenance effort is affected by a large number of factors such as the size of the application, type of maintenance (enhancive, corrective, reductive), the programmers' experience and familiarity with the system, documentation, company processes, complexity and quality of source code. A large proportion of effort is devoted to program comprehension [46]. Again, LOC (added, modified, and deleted) is a metric with which maintenance costs can be estimated.

[47] states there is no best practice method available to estimate the maintenance costs effectively but suggests that both the size and complexity of an application influences the maintenance costs significantly based on the actual rate of defects. Therefore, another model is suggested based on reported software defects, the number of programming languages, basic maintenance effort, and several other factors. This model is made for larger projects than used in our use case.

Since both products developed in this thesis are relatively small, based on specific systems, and we do not have the metrics available to fill in the models, we propose a more straightforward maintenance cost estimation.

NAOqi toolbox 1 hour per month
For NAOqi variants, the biggest influence on maintenance is a firmware update of the robot. This usually results in an API change deprecating some functions and introducing a new method working slightly different, leaving the deprecated function working for one or two years. An experienced programmer and/or someone familiar with the system can usually correct this within a few hours, including testing. Firmware updates of this size happen about one to two times per year. Therefore we state a maintenance cost of 1 developer hour per month. An exception to this is an upcoming firmware update for the Pepper robot where a large part of the API is replaced without deprecation, creating the need for redeveloping large parts of applications. As this is an exception to earlier updates, we do not take this into account for calculations.

Watson toolbox 2 hours per month
Since the cloud part is mainly managed, no maintenance is needed for this. However, the API of the cloud services change more often, as well as updates on the logic server. Both can run for a longer time without needing an update, about once or twice a year. Also, the framework is based on NAOqi, requiring almost the same upgrade path, as mentioned in the NAOqi toolbox. We expect, therefore, that the system as a whole requires more maintenance costs than only NAOqi. We estimate this at two developer hours per month, resulting in 3 workdays of maintenance per year.

8.6.4. TCO for both products

Now all data is available to calculate TCO. We calculate the TCO (excluding hardware) for a period of one, three, and five years of use. The calculations are made for the product made using the NAOqi toolbox, for the Watson toolkit, and a mix of toolkits displaying the costs if all cloud services are used except for the most expensive option: Visual recognition. Price of development for the mixed product is assumed the same as the full cloud version as no specific data is available for development time of this component, so it is unknown how much cheaper that would be. We assume a developer costs 50 euros per hour. The cloud (IBM) monthly costs use the most expensive solution without smart cost reductions, and real costs are thus expected to be lower. All other data is gathered from this section with corresponding assumptions. Table 8.2 shows that for one year the TCO of all variants does not differ much, with mixed being slightly cheaper. Over three years, the difference between full cloud and full NAOqi is much larger. The use case must require higher precision than build-in software to justify the cost increase. Naturally, this difference increases even more when the five years is considered. However, as stated, usage and thus costs are expected to be lower for cloud as these represent maximum values.

		1 year			3 years			5 years		
Cost type	Cost element	IBM	Mixed	NAOqi	IBM	Mixed	NAOqi	IBM	Mixed	NAOqi
One-time	Development	32000	32000	40000	32000	32000	40000	32000	32000	40000
Recurring	Maintenance	1200	1200	600	3600	3600	1800	6000	6000	3000
	Cloud	7665	3507		22994	10520		38324	17533	
Total		**40865**	**36707**	**40600**	**58594**	**46120**	**41800**	**76324**	**55533**	**43000**

Table 8.2: TCO calculation for hospitality application created with both toolboxes and a mixed variant which does not use Visual Recognition. Total lifecycles are calculated for one, three, and five years. All numbers are in euro and do not include hardware costs.

This table misses one important element, which is the same for all variants: hardware. The price of buying a robot and hardware to run it (router, charging station) also needs to be included in the total sum, but since prices can differ this is left out of the calculation. Likewise, independent of the used solution, the customer must have the personnel to position the robot at the start of the day and bring it back at the end, with costs depending per company. The prices for cloud include developing a Framework to run on, which could be reused for later projects, decreasing initial development investment. This calculations from this section can also be made interactively by using the decision sheet for choosing cloud versus on-board constructed in the next chapter, Chapter 9.

8.7. Discussion and Conclusion

This chapter holds an analysis of one-time and recurring costs for developing and running a social robot using both toolboxes. We have shown that for cloud, video analysis and (logic) server costs contribute most to total running cost while audio and dialog are not a big factor. For the video and audio components, easy and significant cost reductions can be achieved. In our case, this resulted in one-third of the calculated costs from real-life usage.

The numbers and conclusions from this chapter do not apply to every solution built this way. When the application is developed in another country, very different developer costs may arise. The same holds for the hours of work needed to finish the application, the use of another cloud provider with different pricing, the use of another robot, the amount of usage during work hours, etc. However, an insight into how these products compare in costs has been shown, as well as how to obtain these numbers for oneself. A reusable structured approach is given. The details will differ, as these are strong context-dependent, but the approaches are generic and basic, suited for testing various robot and application setups. A side note to the cloud numbers is that building a Framework is included in the development price, while this is not the case for the NAOqi toolbox. A fairer comparison would be looking at scenario-implementation only, reducing the cloud development costs to about half the price.

Whether one solution or the other is a valid business case can be interpreted differently depending on the kind of company. A small company with less to spend could be interested more in developing full cloud, as the one-time startup costs are lower with the recurring costs being very dependent on usage and amount of cost reductions. For larger companies a large investment is easier to make but can also result in higher development and maintenance costs (due to more expensive personnel), affecting the decision. An average receptionist in the Netherlands earns about €31.000 per year [1]. Although not all receptionists can be replaced when using a social hospitality robot, since the system cannot operate fully on its own, it still could save a company 34.000 to 51.000 euro (minus robot purchase) per employee in three years.

Other cost structures are also possible. Larger companies developing a social robot application can also decide to offer the product as a total solution to smaller companies in the form of a lease agreement, providing robot, maintenance and (managed) software as one, reducing the initial investment costs.

As shown, not every component needs to use a cloud service but could work on the local system from the Pepper Toolbox. Chapter 9 gives a method to help determine per component whether cloud or non-cloud is advisable for the design of a robot application.

This answers the fourth research question of this thesis on the up and downsides of using a cloud solution for robotics when looking at business values. When the product is used for a shorter period (one year or less), the upside for cloud is high-quality processing for the same or even less of a price than the non-cloud version. For more extended periods, the NAOqi toolbox offers a cheaper but less performing option. Where this balance lies depends on many factors, including the availability to a pre-existing framework for the cloud, the required abilities and needed performance and amount of expected use.

[1] Source: indeed.nl, Average salary of a receptionist per year

9 Decision method

Previous chapters provide background, software, and experience and thus a basis to evaluate the use of cloud functionality in social robots. When trying to determine whether to implement a social robot product using the cloud or using built-in functionality (in this thesis called the Pepper Toolbox versus the Watson Toolbox) one needs to know and understand the trade-offs involved in this decision. As not everyone is fully-fledged into this matter or aware of the options, costs, or expectations to have, this chapter describes the construction of a matrix for companies and developers to make an educated decision. It is intended to be used without needing all methods, tests, and data provided in this thesis, but if more information is needed, this could be used as reference material. Even after the quantifiable results from this document are outdated due to quick developments in this field, the matrix would still be relevant to use since the method is given to obtain the numbers to conclude from. Two prerequisites should hold before filling in the decision matrix:

- The cloud needs the availability of fast (>2mb/s transfer rate/upload speed) and stable internet connection. Without this, a NAOqi/on-board based product is the only way.
- The robot interaction is social by nature

Every tab in the matrix contains instructions of how to enter the data. The decision matrix constructed in this chapter can be found in Appendix C. It is pre-filled with data based on the robot application developed for the second evaluation: the CASperSocket. With our configuration of architecture and by using a mix of the toolboxes for the components, the matrix indicates this is indeed the best solution compared to using one single toolbox. It scores the highest amount of points in our scoring system with a lower Total Cost of Ownership.

9.1. Approach

Finding the right balance between the trade-offs mentioned in this thesis can be done in the same manner this report is constructed. By categorically choosing the importance of indicators for an intended interaction, the decision can be made step by step. We use the same categories as found in the Research questions: Interaction indicators, Quality and robustness indicators, Software engineering indicators, and cost indicators. We calculate a weighted score or summarizing number for these categories for three types of products: a product built using only the Watson toolbox, a product built using only the NAOqi toolbox, and a product built using a mix of the toolkits based on the choices made in the matrix. These can be compared after filling in the matrix.

We assign a multiplier per toolbox per indicator, showing how well this item works per toolkit. The multiplier has a scale of 1-10 and is based on the experience of the author, checked by other experienced programmers of the Pepper robot. As there is often more than one solution to build features in software engineering, the minimum multiplier is one instead of zero. This means the feature can work or be programmed with this type, yet the performance is low, costs are high, or it takes much developer effort to make it work as intended. A multiplier of 10 should be interpreted as the best suitable option which is easy to implement, configure, has low costs and/or has the best performance/accuracy.

The person(s) filling in the matrix can then assign an importance value per indicator, which is then automatically multiplied using the given multiplier resulting in a score per toolbox. A choice for a toolkit for that indicator can now be made, and one directly gets feedback on that choice. A 'Not needed' option is included if the indicator is not used or not relevant and will not count a score. Summing the scores gives us a total score per toolkit for the intended interaction.

This is followed by a development hours calculation and cost calculations based on that. With all categories filled in, a summary can be constructed from which an informed decision for a toolbox (mix) can be based on.

The multipliers are intentionally hidden in the sheet for clarity and not tempt users to think about their decision instead of choosing straight away for the option with the highest multiplier, as sometimes the highest multiplier might not be the best choice.

9.2. Categories

The matrix is split into different tabs for the categories to keep a clear view of the current indicators. All tabs are discussed below. Every tab contains instructions on how and where to enter data and how to choose the values. All values needed to be filled in have a green color in the matrix.

9.2.1. Interaction indicators

The matrix contains indicators for various types of elements to take into account when deciding to use or not to use cloud functions in social robotics.

The interaction indicators are categorized per component and primarily based on the observations made during the evaluations of this thesis which can be found in Subsection 6.5.2 as well as the metrics from Chapter 7.

The most influential indicators are included in the matrix. These define how an interaction or use case would work and thus, which components are essential to work well and therefore choose a particular toolbox. An example is the first item on the list: "Use of unrestricted/free speech input" (Speech component). When it is mostly unknown what users are going to say to the robot one almost certainly needs the cloud (with Speech to Text) to obtain the actual transcription, as this is currently not possible on the Pepper robot as it is keyword-based (with an exception for specific robot licenses and English language). Since it is technically possible to implement a Speech to Text running on Pepper itself, yet requiring more developer effort with (expected) lower STT performance, the multiplier for this indicator is 2 for NAOqi and 9 for the Cloud. With the same reasoning, all multipliers are determined.

At the bottom of the tab, the total interaction score can be found for the three types of toolbox. One can continue to the next tab.

9.2.2. Quality and robustness

This tab's setup is similar to the interaction tab but focuses on the quality and robustness of the robot application. The list includes indicators as the ability to perform live updating of software, (near) realtime processing and the required level of autonomous operation. Again the end scores are presented at the bottom of the page, and the next tab can be opened.

9.2.3. Software engineering

This tab tries to estimate the developer effort needed to build the application. To achieve this, one must first select whether a base framework is available for connecting with the cloud. If available, this reduces the needed effort for the cloud significantly. Therefore, if 'Yes' is selected a part of the sheet will grey out as this is purely framework development.

The user's task is to estimate how many Lines of Code the components and scenario development will take to build. It should be filled in for both toolboxes to be able to compare costs in the end. The same holds for the expected complexity for this component when using this toolkit if one think it differs from the default values there. A choice must now again be made which toolkit to use per component which can also be helped by looking at the same category choices in the previous tabs.

With the LOC available, the effort estimation model from Subsection 8.6.2 is applied to achieve the number of developers hours needed per toolbox. If one thinks the estimation is off, the value can be changed here as this will be used for later calculations. The average complexity is also automatically calculated. Default data in this tab is gathered from Section 4.3.

9.2.4. Costs

In this final tab, the Total Cost of Ownership for all three variants is calculated using the same method as in Section 8.6. One enters the available budget, period of use, number of robots going to run the application, and developer costs per hour. The operating costs per month per component follow next. If the defaults do not seem correct, these can be changed. Finally, one enters the costs for robot purchase, the optional costs for a framework, and accessories needed for the robot.

Now all data is available to calculate all costs for the full running period automatically and is displayed below all inputs. An indicator shows whether the chosen solutions are within budget.

9.2.5. Total score

Returning to the Total Score tab, an overview is displayed summarizing all data. After comparing the numbers, one can interactively fiddle with the numbers in all other tabs until satisfied with the solution and able to make an informed decision. The TCO includes all costs for the chosen period of use.

9.3. Method limitations

The matrix constructed in this chapter has some limitations. It has a balance between focusing on a Pepper robot interaction while still being generic enough for other kinds of robots. As a result, some indicators and default values may not be fully applicable to one's intended robot application or model. However, the basic principle can be reused or adapted.

The multipliers used in the first two tabs are based on the author's and other Pepper programmer's experience. Others in the community did not validate it, which could improve its accuracy.

The effort estimation used in the matrix has the same limitations as mentioned in Subsection 8.6.2 and was not accurate for our case. This could influence the final costs negatively, which is an integral part of the matrix. If a better model or more information for estimation is available, this calculation can easily be replaced, improving the results.

10 Discussion

In this work, two software applications have been built and evaluated on their workings and development method. Using a broad approach, the goal was to find and discuss trade-offs for the components, rather than an in-depth analysis per component. The result of this work is a first step in the development of a framework for assessing cloud versus on-board performance, costs, workings, speed, and way of development.

The cloud solution sees an improvement on many levels but is not yet on the robustness and performance level for full-time use. The cloud solution was found to improve capabilities, performance, and accuracy compared to the built-in toolbox but is not yet on the robustness and performance level for full-time use, as current other similar systems. This can be seen in practice as the robot still needs guidance and cannot always understand people due to lousy Speech to Text performance. Another example is when a quicker response is initially expected, which causes users to repeat or speak early, reducing the user experience as unexpected robot behavior often follows due to this. Some more fine-tuning in programming and more evident indications of when the robot is processing and when it is listening should help. Since this is a research prototype build by one developer instead of a team, these kinds of faults are to be expected and should not influence the view on such a system in general.

Next to this, other state-of-the-art systems similar to this one have their restrictions. Applications already running 'in the wild' usually focus on doing one element outstanding but have their errors when looking at overall (interaction) performance. Some systems are running reliably by only using tablet buttons as the input method, severely restricting user freedom and not utilizing the robot' full potential. Others only respond to yes/no questions, or do not accept user input at all and are just activated by vision. Some are used as passive marketing stand mostly unaware of what users in front of it are doing. Robots have many tasks nowadays, but it is also essential to be aware of what they cannot do.

Current similar systems to out CASperSocket implementation (such as Intu) use many other concepts for programmers to understand, in specific languages, which are often not needed for simple scenarios and adding developer effort. Compared to currently available state-of-the-art robot applications with a similar goal, the product we created in this thesis performs adequately on all components instead of just one or two. Especially the beamformer's good audio output results in a better performing Speech to Text than other systems and delivers broader conversations at the cost of higher (300ms) latency. The CASperSocket can and has been programmed by people with little knowledge of the system and is easily adaptable and modular, while other systems require reading copious documentation or specific platforms before one can get started. For the video component, better and especially quicker software is available but often cannot work side-by-side with the regular running robot system.

We expect that in five to ten years these now emerging state of the art systems, usually built as an experimental setup to see what works, will mature into more generic setups with medium complexity. They run on top of the systems provided by robot manufacturers, agnostic to which platform it runs on using a hardware abstraction layer. The system can read the properties of the robot it runs on and displays the capabilities of this model. Control of the robot can be done online or locally. When programming the robot, a choice can be made for local sensor processing or using a cloud service, as the approach to achieve this is similar on every platform. With at least the same technology on board as home automation devices such as Google Home (microphone array and speakers), conversations will happen in a similar manner as those systems and mature in the same pace. This means the robot's understanding of the user will improve but have the same limitations as every other Speech to Text system at that time, resulting in less but similar ’weird’ robot behavior as we have now. More research will be needed to achieve such a system as stated here and improve on the current situation since there are no off-the-shelf solutions to make this yet.

Stability and robustness also depend on how one looks at it: the cloud exists of many components and thus more places for failure to occur. On the other hand, they are all externally managed, and if one fails, the rest can continue to work so only a small part of the robot stops working, offering better stability when compared to the whole robot halting.

Useful comparisons of the evaluations have been described. In a next iteration, the validity and completeness of the comparison could be improved by creating a more similar test situation. Another test location was used, with other types of clients speaking a different language. Other software was used, with different capabilities for the same goal, with other evaluation metrics. This makes the comparison valuable as information for insight. However, more data points could be analyzed if the test setup was more similar. Still, client feedback and the percentage of use of speech can be used to compare the technologies. Also, one clear comparison can be made from a developers view, as the development of the same basic functionality differed much in ease, with the cloud being easier. This could then again differ by developer experience.

Language has been an important factor in the acquired data from the evaluation. Interaction completion rates would have been much higher if no Dutch names were used in the interaction, while English was set as Speech to Text language. The foreign names were seldom understood by the system but were needed to complete the interaction, leading to frustration or loss of engagement. However, it is not an essential factor for comparing the systems aside from the useful fact that Dutch currently is not supported by most cloud providers. Had this element not been included in the use case, we expect that overall, the results for cloud would be better.

When moving all processing to the cloud, the robot becomes similar to a thin client. Should robot manufacturers stick to this concept and build new models with cheap hardware as no processing power is needed on-board anymore, or should the robot’s hardware improve, so the cloud is not needed anymore as it is available locally? Both concepts have their pros and cons. For robot’s like Pepper, we expect an in-between solution as it is now. The cloud costs money to use so forcing customers to use the cloud can make the robot less attractive to buy, while hardware costs at this performance level do not cost that much compared to the rest of the robot. On the other hand, the level of hardware needed to make its performance comparable to the cloud is a substantial investment and not needed for many use cases, next to using substantial more energy thus battery time will decrease significantly, which is unfavorable. The current hardware allows for both with good battery life.

11 Conclusion

Given a robot freshly out of the box, how does one start programming and using it? One can create a robot application using manufacturer' software such as Choregraph or using API's in different programming languages. This works well for small projects and basic applications can be made, but can more be achieved with the same robot, for example, using cloud software? What trade-offs have to be taken into account when doing this. This thesis contains a broad view on this topic. We have made the following contributions:

Contrib 1. an overview of relevant metrics and test methods for a robot's components, with acquired data for some components of the Pepper robot

Contrib 2. architecture overviews of such systems, what upsides and downsides these have, and finding a stable maximum throughput for the cloud system.

Contrib 3. evaluations of how users experience the same social hospitality use case with two differently developed products based on cloud and on-board software.

Contrib 4. a cost structure and running costs overview for a (social hospitality) robot using both systems

Contrib 5. a decision matrix combining all information and experience of this thesis in a compact form accessible to decision-makers who are not necessarily knowledgeable in social hospitality robots and/or the cloud (See Chapter 9). It aims to have the right balance between thoroughness, genericness, and understandability.

In the search for trade-offs when building a social hospitality robot product using cloud services, two applications built with different toolboxes have been evaluated based on the same use case and requirements. The NAOqi toolbox consists of all applications belonging to or installed locally on the Pepper robot, and the Watson Toolkit consisting of cloud services. Although both products succeeded in helping clients at the reception, the robustness in doing this was generally low.

The first evaluation showed that the built-in speech recognition is often not working as intended resulting in clients using the fallback tablet input option for most of the interaction. This problem was largely solved in the second evaluation using Cloud Speech to Text, but here problems arose with the dialog system. Another main issue was reducing noise in the audio signal sent to the cloud. At both locations, clients responded positively on the (experience with the) robot. To obtain more insight into the difference between cloud-enabled and local applications, we researched the quality of cloud components, interaction quality, software engineering complexity, and cost aspects.

The first research question focuses on *the component quality of the built-in Pepper Toolbox compared to IBM Watson Cloud services.* Relevant metrics for every component are described, including the method to obtain and test these. *(Contrib 1.)* Some of the tests are performed, generally resulting in the conclusion that cloud service results are better in terms of accuracy and performance than built-in software of the Pepper Toolbox, yet having an increased latency of 300-500ms. Whether this latency influences an interaction negatively depends on the application and/or use case. For our hospitality use case, the latency of Speech to Text and Visual Recognition was low enough to work with. In the questionnaire, participants mentioned a delay, which was also observed during the second evaluation, where people started to repeat their answer to the robot occasionally before the robot could respond. *(Contrib 3.)* A lower latency would reduce the negative side for the cloud on this trade-off between high latency and high performance versus the opposite using built-in modules.

Head tracking on the robot worked with enough accuracy and low latency being perceived naturally by humans, and thus this component outperforms the cloud service, as the head's reaction speed using the cloud is often too low to be perceived convincing. Finally, running all cloud components simultaneously does not give performance issues anywhere on the pipeline; the robot, cloud server, and connection between them have enough processing capacity. *(Contrib 2.)*

Answering our first research question: the IBM Cloud services win in comparison on quality in terms of performance and accuracy and capabilities, yet lose to the Watson toolbox with the added latency. In the case of Speech-to-text this means that if the robot must respond quickly (with a limited answer space), the built-in recognition service is recommendable. If general speech needs to be recognized and time is of less an essence, the cloud service is ones only reasonable choice.

The second research question *compares the two toolboxes with respect to the quality of human-robot interaction.* Data gathered in the two evaluations presented in this thesis showed an increase in feedback score (79% to 95% with a feedback score of 4 or higher) and use of speech (52% to 86%, instead of the tablet) using the cloud components. *(Contrib 3.)* Observations and questionnaires show that the ability for users to use unrestricted speech input (STT from cloud) improves the ability for users to steer the conversation and perform the interaction in a more natural way but this also introduces less robustness in finishing the conversation correctly, as sometimes the dialog agent (Watson Assistant) did not know to get back to the developer's intended flow of interaction after a digression. A loss of engagement is the result. Improving the dialog could solve this problem as this 'chatbot' technology is used widely nowadays without these types of errors.

Next to the cloud dialog service resulting in higher quality interactions, also eliciting user engagement early is helped by the improved vision: the ability to detect an approaching user from a ten-meter distance instead of two. This system works well together with built-in face tracking for gaze behavior: detect users quick and track engagement using the cloud's Visual Recognition, but perform gaze tracking using the fast built-in software.

Finally, direct feedback on the tablet to the user, showing partial Speech to Text results while users talk and text spoken by the robot, helps users understand the robot's behavior better, as their way of speaking often adapts after one or two sentences, improving this input method's results. If the dialog system does not understand the user, it asks for clarification on the specific item it misunderstood, further improving recovery of a failure.ft With the NAOqi toolbox, quick reactions can be given by the robot due to fast working Speech to Text leading to quick conversations if the user is understood well, but the latter is often an issue.

Concluding on this research question, the quality of human-robot interaction is better using the Watson toolbox due to better insight in the users' intents (by use of general speech to text and an intelligent dialog agent) and increased robust sensor processing capabilities. Some items, such as head tracking (gaze behavior) and Text-to-Speech works well from the built-in toolbox where the cloud offers no to little advantages or even adds disadvantages.

Software engineering complexity differences between non-cloud and (partial) cloud solutions have been analyzed as part of the third research question. Building an application using only built-in components offers an easy learning curve and quick setup by use of Choregraph, yet debugging, version control, and parallel programming is harder to perform. Unexplainable errors do also happen with this way of programming, next to code instructions not being executed by the robot and not having regular IDE-like features for developers. For small projects with little requirements, or quick demo's, this may be the easiest solution. Instead of using Choregraph, one can also program the application using the API's provided by Softbank in various programming languages, among which Python. This would remove some of the negative sides such as version control, IDE features and parallel programming and adds options for automatic testing. For larger projects with more requirements (especially concerning higher accuracy, connection to various external services), the cloud or a mix of toolboxes can form a better solution.

With a base framework for the cloud in place, many advantages for a developer are present such as the support for easy and precise debugging, live-updating during operation, coding using drag & drop and text combined like Choregraph, easier testing due to being able to 'replay' recorded sensor input and the availability of many programming languages to work with. A downside is having multiple codebases (on the robot, cloud server, in a cloud service) and thus also requiring developer knowledge of multiple systems instead of one, next to more systems where an error can occur, reducing robustness. *(Contrib 2.)*

Summarizing: with a base framework available for the cloud the effort for building a product using this toolbox is relatively low and offers similar complexity over the full project than a project of the same size built using Choregraph or using the API's. Both toolkits have their pros an cons, and it depends on the programmers' experience, which suits best. When the framework has to be built first, the complexity and effort are higher.

The upsides and downsides of using a cloud solution for robotics, when looking at business values, forms the fourth and final research question. The answer depends on the availability of a base framework to work on for the cloud, comparable to the CASperSocket developed for this thesis. If available, the required effort for scenario design is low (also due to extra developer tools as described above) and the cloud is cheaper when looking at Total Cost of Ownership, especially for products with a life cycle of a year or shorter or for proof of concepts. The performance to cost ratio of the whole product is better using the cloud. Although development for cloud is cheaper, the continuing operating costs are its pain point, mainly when the product is used for a longer time (>3 years).

This is not the case with the NAOqi toolbox, where there are low operating costs except for a little maintenance or low operating costs when a mix of toolkits is used. If no framework is available and this has to be developed first, the cost balance shifts away from the cloud: the product costs from the two toolbox are about the same or the cloud is more expensive. *(Contrib 4.)*

Performance and accuracy are also important for the companies, which is a side the cloud usually wins. Other upsides important for companies is no downtime with live updates and the ability to reuse code initially developed for the robot in other company products. The dialog the robot uses can, for example, be reused as a chatbot on the companies' website with low effort.

Which of the options to take depends thus again on the goal one wants to achieve with the product. In the end, all solutions provide a cheap, innovative, and at the moment medium reliable alternative for *augmenting* hospitality-related jobs which could save companies a substantial amount of salary costs. A vital remark with this statement is that the robot cannot fully replace, for example, the entire receptionist staff. A human presence is always needed as the system is still limited in its functionality and offering solutions for problems it is not designed to solve will not happen (yet?).

In conclusion: If the robot is used for a short time, the cloud has more upsides by offering high-quality services for the same price as the other toolbox. For a more extended period, the NAOqi toolkit is cheaper. It is up to the (budget of the) owner of the system, whether the increased costs are worth the higher quality.

Having addressed all these questions means that we can answer our primary research objective: *Which trade-offs have to be taken into account when making a choice between using the Pepper Toolbox versus the IBM Watson Toolbox for creating a social hospitality robot?* Many of these trade-offs are present, each with their importance. The main argument to answering this question is taking the middle road: why choose one or the other platform if one can have a little bit of both; a combination of systems using the best of both worlds for that specific application. If the application is a static robot, only there to help answer a broad array of simple questions, one can use Speech to Text and Assistant from the cloud, while using the built-in functions for all other components, running logic from either the cloud or locally on the robot.

The main trade-offs to consider are the following. General high accuracy and performance versus low latency and no running costs, low learning curve drag and drop programming versus the availability of many programming languages and possible configurations but needing developer knowledge of cloud (services), making a framework first versus starting scenario development from the beginning, live updating of code and latest state-of-the-art cloud services versus internet-free and coding in one codebase for free. For costs, the use period and availability of a framework for the cloud are important factors. *(Contrib 1, 2, 4.)*

11.1. Future work

What will the social robotic landscape look like in five to ten years? Aside from this thesis, several companies are developing or already using cloud-based social robotics. Inhabitants of countries like Japan are not surprised to find a robot assisting them daily. We expect growth in the use of these robots around the world but with a slow adoption rate. Next to the innovative aspect, the solutions currently created do not always add value for a company and where these robots have their most value is not yet known. As the cloud offers higher performance at a reasonable cost (depending on configuration), we also expect the use of this will continue and grow. To achieve this future vision, further work is needed to discover the best possibilities to use this. We have set significant steps towards this by building two complete social robotic products used in the wild outside of lab environments or demo spaces and evaluating it with real users.

The data gathered in this thesis is expected to change relatively quick, yet it gives enough indication now to be able to draw conclusions. The advantage will slowly shift to the cloud for numerous reasons as discussed before, but robot hardware may also improve in the future, shifting the balance back to local processing. This means the trade-offs will largely stay present. A repetition of this work in a couple of years could make this clear.

Although we have set big steps in finding trade-offs for using the cloud in social robotics, the picture is not complete and needs further research. The CASperSocket system supports the use of multiple robots from one logic server, but we did not test the effects on performance, shared knowledge, or issues arising from this setup. The performance and accuracy of other cloud providers could differ much per component and improve or decrease the metrics we gathered, next to having a completely different cost structure. The balance for choosing cloud and/or NAOqi could shift again for specific or all components.

An argument against the cloud is the latency it introduces in the pipeline, but this effect can be reduced by using a private cloud hosted locally. How much is the difference in latency, what are the up and downsides of this solution, what is the effect on costs versus performance, and why should one still choose a cloud provider instead of a self-owned server setup?

The Pepper robot was used to perform all measurements. How does this generalize for other robots? Several other manufacturers produce robots having similar characteristics and can be programmed for the same use case. It would be interesting to see how they compare to the conclusions drawn here. The test setup in this thesis would be generic enough for most other types of robots like these.

While in our cloud product, the user is automatically analyzed for gender and age (and optionally emotion), nothing changes the robot's behavior based on this data yet. Research shows that based on these parameters, voice, spoken text, and gesture behavior can be changed to be fitting for that user: for example, speak louder and slower for older users. The user's speech could also be analyzed and used to mimic the users' current state of mind easing the interaction: speak fast and enthusiastic if the user does the same. This would help the users in a hurry as seen in our evaluation and questionnaire, where some users wanted the robot to speak quicker and less verbose, and some wanted more explanations and a slower experience. All services and modules used in our product would support doing this in the pipeline with low effort and could improve the robot's interaction significantly.

A Evaluation

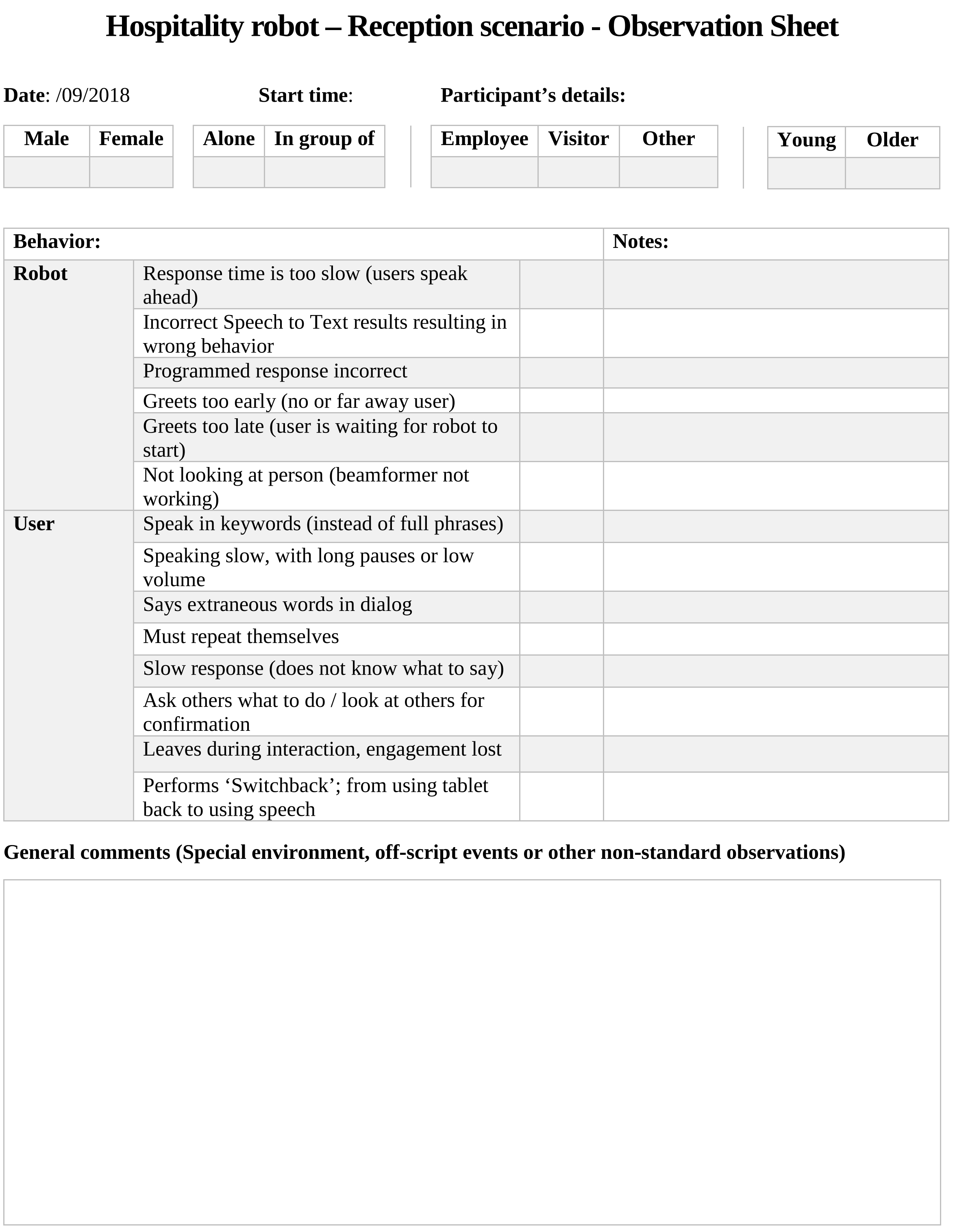
This Appendix contains the questions asked during the questionnaire of the second evaluation and the user observation sheets.

A.1. Questionnaire

These are the questions asked to participants when performing the questionnaire.

- How would you describe your experience during this interaction?
- What do you think went well with the interaction, and what did not go well?
- (Where did it go wrong and) why do you think this happened?
- Which suggestions do you have for improving the robot?
- What did you like/not like about the robot?
- (How) Do you think this robot add to the hospitality experience here?
- On a scale of 1-5, please rate your impression of the robot on:
  - machinelike - humanlike
  - artificial - lifelike
  - fake - natural
  - unpleasant - pleasant
  - unfriendly - friendly
  - no control - in control
  These measures are selected from the Godspeed questionnaire [33], a standardized measurement tool for human-robot interaction developers, to measure people's perception of a robot.
- Do you have any other comments?

A.2. Observation sheet



Hospitality robot – Reception scenario - Observation Sheet

Date: /09/2018 **Start time**: **Participant's details:**

Male	Female

Alone	In group of

Employee	Visitor	Other

Young	Older

Behavior:			**Notes:**
Robot	Response time is too slow (users speak ahead)		
	Incorrect Speech to Text results resulting in wrong behavior		
	Programmed response incorrect		
	Greets too early (no or far away user)		
	Greets too late (user is waiting for robot to start)		
	Not looking at person (beamformer not working)		
User	Speak in keywords (instead of full phrases)		
	Speaking slow, with long pauses or low volume		
	Says extraneous words in dialog		
	Must repeat themselves		
	Slow response (does not know what to say)		
	Ask others what to do / look at others for confirmation		
	Leaves during interaction, engagement lost		
	Performs 'Switchback'; from using tablet back to using speech		

General comments (Special environment, off-script events or other non-standard observations)

How to perform the observations:

Male/Female: If this cannot be determined clearly, don't tick a box

Alone/ in group of: tick 'in group of' when multiple persons are in front of the robot, within 1.5 meters, at the start of the interaction, also when these people do not talk, interact or walk away after the start of interaction. Otherwise, tick 'Alone'.

Employee/Visitor/Other: Check the badge of the person engaged with Pepper to determine Employee/Visitor. If the person does not carry a badge or does not receive one from the reception, tick 'other'.

Young/Older: Ranges are Young (20-40) or Older (>40 - 65). If this cannot be estimated with high certainty or estimated age lies outside of these ranges, do not tick any box.

Response time is too slow (users speak ahead)	When users start talking again after a given phrase before the robot gives an answer. Only tick if STT result had a high confidence, otherwise the robot does not answer because the confidence was too low resulting in 'no input given' and thus awaiting a new user response.
Incorrect Speech to Text results resulting in wrongly behaviors or responses	• When users are interrupted by the robot while talking because STT detected a false end of sentence (pause detected, but user keeps on talking) and the robot already responds on that sentence. • STT results wrong or getting input while no one has spoken (ghost input)
Programmed response incorrect	STT result was correct (full transcription of said words by user) but robot failed to answer in correct way, by either giving no, random (AnythingElse state) or wrong response because of being in wrong state.
Greets too early (no or far away user)	When there is no user present or is located far away, the robot still performs a greeting.
Greets too late (user is waiting for robot to start)	When there is a user in front of the robot in the engagement zone, but the robot does not detect the user and keeps being idle.
Not looking at person (beamformer not working)	When the robot is not locked on the user's head anymore during an interaction. Do not tick if the robot is in idle mode and user approaches but is not recognized.
Speak in keywords (instead of full phrases)	The user does not use full sentences as one would do to other users but speaks in keywords. Also tick if user switched to full sentences later in the interaction.
Speaking slow, with long pauses or low volume	The user speaks very slow, extra pronounced, with long pauses or low volume relatively from the general userbase.
Says extraneous words in dialog	User makes comments or says phrases not relevant for the interaction, possibly confusing Speech to Text or Watson assistant. Also tick if speech is not directed at robot but in general or to bystanders.
Must repeat themselves	Tick if a user must repeat words because the Speech to Text did not hear the user or had a too low confidence
Slow response (does not know what to say)	The user does not give an answer within 5 seconds of robot response.
Ask others what to do / look at others for confirmation	The user asks other humans what to do, how to approach the robot, what to say or looks for confirmation.
Leaves during interaction, engagement lost	User is engaged in interaction, meaning the robot is in engaged state and person is speaking to the robot, but leaves and stops the interaction before having given feedback.
Performs 'Switchback'; from using tablet back to using speech	If a user has started using the tablet as input, either because Speech to Text did not work correctly before, instructed by the robot or by own choice, the user switches back to using voice commands after.

B
Interaction dialog flow

Figure B.1 shows a visual overview of the dialog used with the robot build using the Watson Toolbox. The dialog structure is similar yet a bit bigger than the version built with the Pepper Toolbox. The dialog should be read from the top to bottom. A legenda is located on the left top. Dark orange text represent the state's name, red text entities gathered from the user. Green nodes are root nodes found in the Watson Assistant.

Figure B.2 and Figure B.3 show how this structure is translated into a Watson Assistant skill.

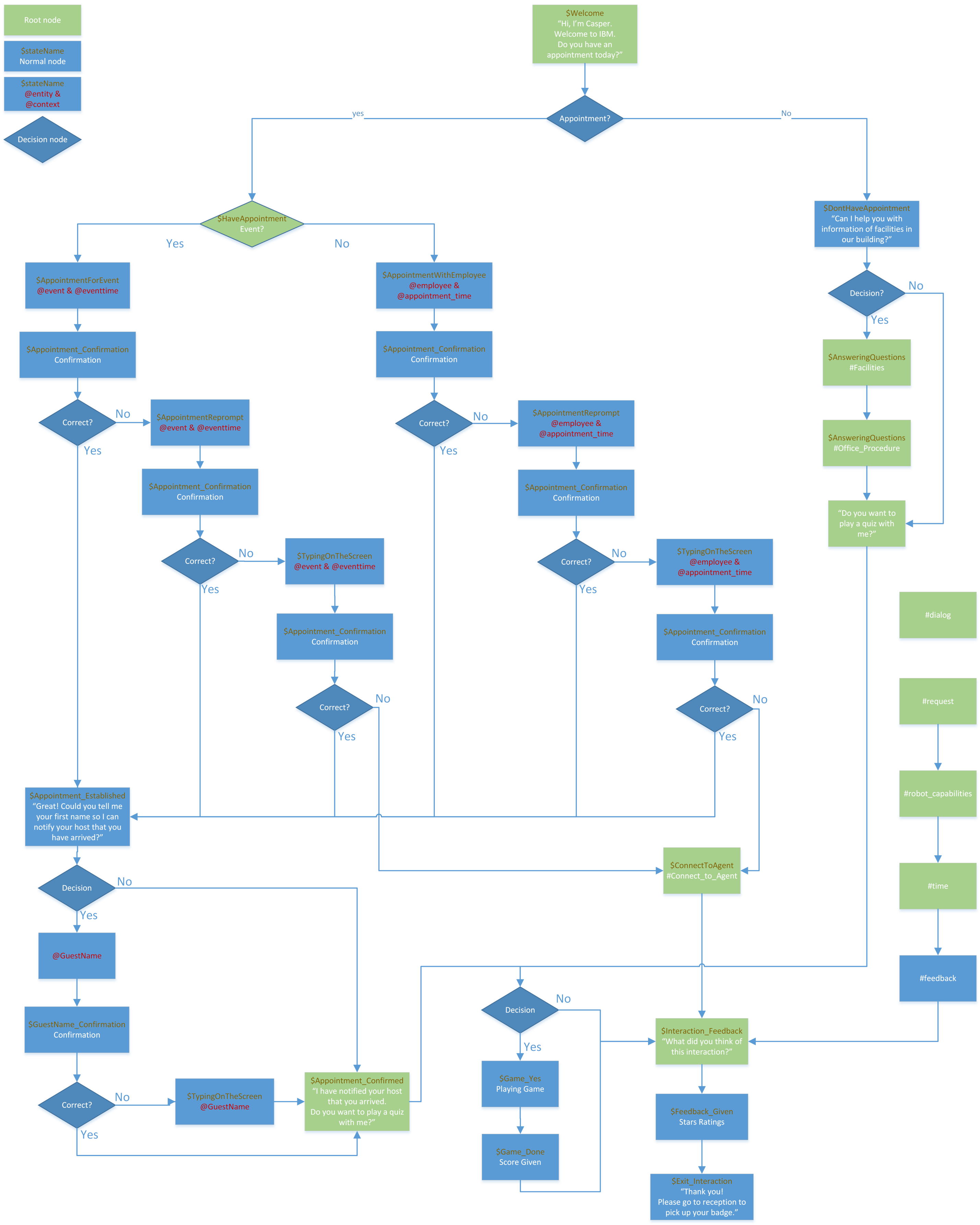
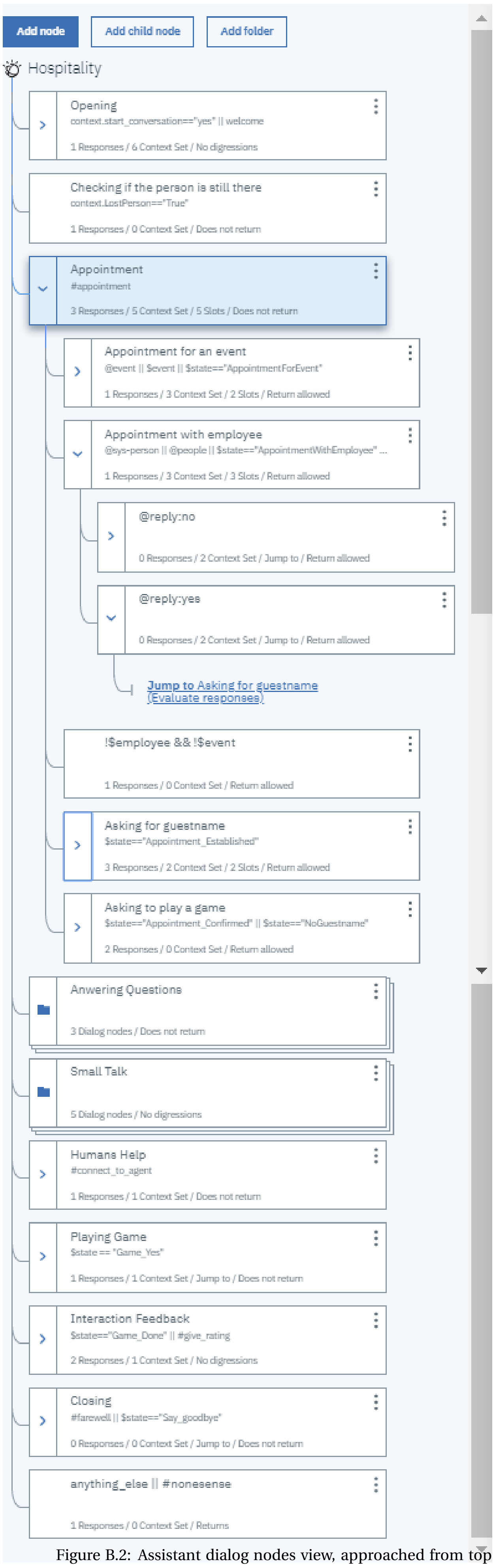

Root node
$stateName Normal node
$stateName @entity & @context
Decision node
$Welcome “Hi, I’m Casper. Welcome to IBM. Do you have an appointment today?”
Appointment?
yes
No
$HaveAppointment Event?
Yes
No
$DontHaveAppointment “Can I help you with information of facilities in our building?”
$AppointmentForEvent @event & @eventtime
$AppointmentWithEmployee @employee & @appointment_time
Decision?
No
Yes
$Appointment_Confirmation Confirmation
$Appointment_Confirmation Confirmation
$AnsweringQuestions #Facilities
Correct?
No
Yes
$AppointmentReprompt @event & @eventtime
Correct?
No
Yes
$AppointmentReprompt @employee & @appointment_time
$AnsweringQuestions #Office_Procedure
$Appointment_Confirmation Confirmation
$Appointment_Confirmation Confirmation
“Do you want to play a quiz with me?”
Correct?
No
Yes
$TypingOnTheScreen @event & @eventtime
Correct?
No
Yes
$TypingOnTheScreen @employee & @appointment_time
#dialog
$Appointment_Confirmation Confirmation
$Appointment_Confirmation Confirmation
Correct?
No
Yes
Correct?
No
Yes
#request
#robot_capabilities
$Appointment_Established “Great! Could you tell me your first name so I can notify your host that you have arrived?”
$ConnectToAgent #Connect_to_Agent
#time
Decision
No
Yes
@GuestName
#feedback
Decision
No
Yes
$GuestName_Confirmation Confirmation
$Interaction_Feedback “What did you think of this interaction?”
Correct?
No
Yes
$TypingOnTheScreen @GuestName
$Appointment_Confirmed “I have notified your host that you arrived. Do you want to play a quiz with me?”
$Game_Yes Playing Game
$Feedback_Given Stars Ratings
$Game_Done Score Given
$Exit_Interaction “Thank you! Please go to reception to pick up your badge.”

Figure B.1: Dialog structure including states, entities and decision points.



Hospitality

Opening
context.start_conversation=="yes" || welcome
1 Responses / 6 Context Set / No digressions

Checking if the person is still there
context.LostPerson=="True"
1 Responses / 0 Context Set / Does not return

Appointment
#appointment
3 Responses / 5 Context Set / 5 Slots / Does not return

Appointment for an event
@event || $event || $state=="AppointmentForEvent"
1 Responses / 3 Context Set / 2 Slots / Return allowed

Appointment with employee
@sys-person || @people || $state=="AppointmentWithEmployee" ...
1 Responses / 3 Context Set / 3 Slots / Return allowed

@reply:no
0 Responses / 2 Context Set / Jump to / Return allowed

@reply:yes
0 Responses / 2 Context Set / Jump to / Return allowed

Jump to Asking for guestname (Evaluate responses)

!$employee && !$event
1 Responses / 0 Context Set / Return allowed

Asking for guestname
$state=="Appointment_Established"
3 Responses / 2 Context Set / 2 Slots / Return allowed

Asking to play a game
$state=="Appointment_Confirmed" || $state=="NoGuestname"
2 Responses / 0 Context Set / Return allowed

Anwering Questions
3 Dialog nodes / Does not return

Small Talk
5 Dialog nodes / No digressions

Humans Help
#connect_to_agent
1 Responses / 1 Context Set / Does not return

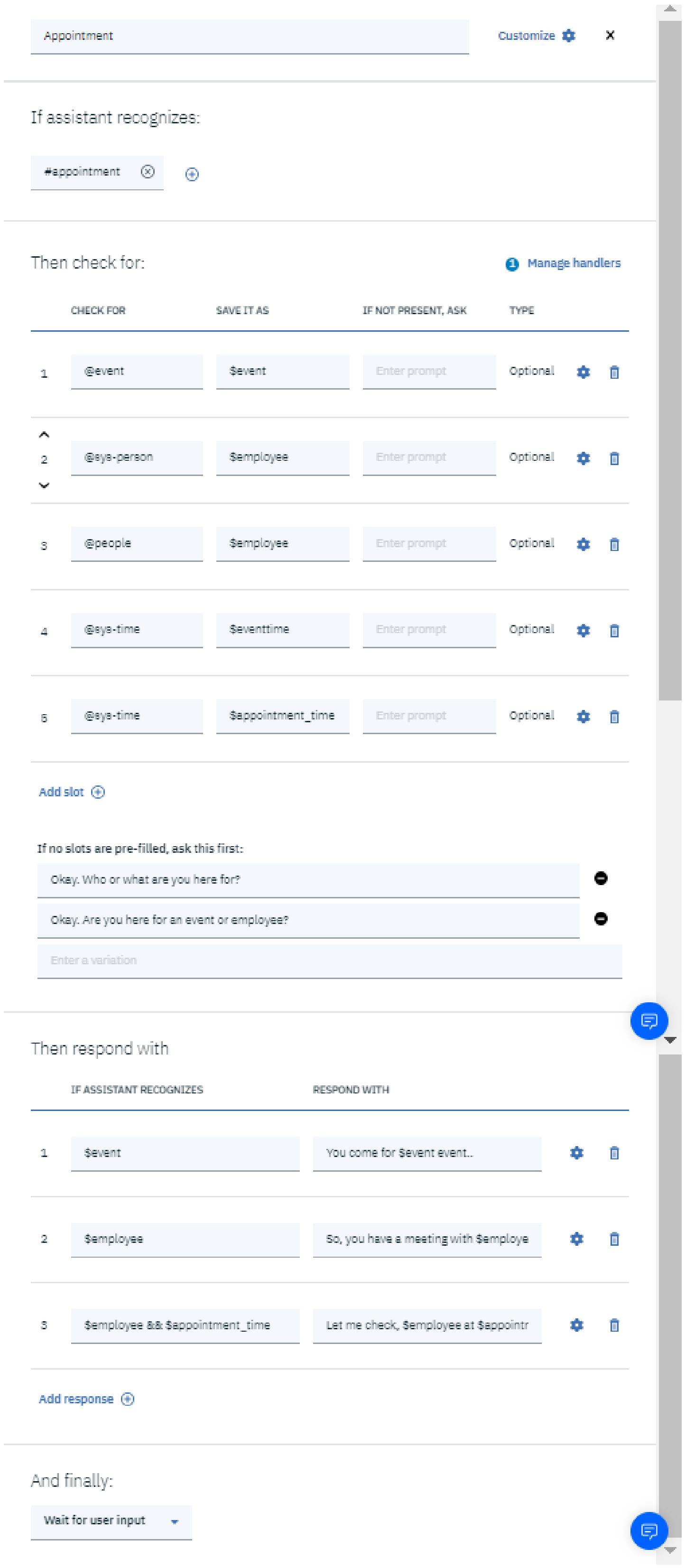
Playing Game
$state == "Game_Yes"
1 Responses / 1 Context Set / Jump to / Does not return

Interaction Feedback
$state=="Game_Done" || #give_rating
2 Responses / 1 Context Set / No digressions

Closing
#farewell || $state=="Say_goodbye"
0 Responses / 0 Context Set / Jump to / Does not return

anything_else || #nonesense
1 Responses / 0 Context Set / Returns

Figure B.2: Assistant dialog nodes view, approached from top to bottom. Nodes located on the left are root nodes



If assistant recognizes:

#appointment

Then check for:

	CHECK FOR	SAVE IT AS	IF NOT PRESENT, ASK	TYPE
1	@event	$event	Enter prompt	Optional
2	@sys-person	$employee	Enter prompt	Optional
3	@people	$employee	Enter prompt	Optional
4	@sys-time	$eventtime	Enter prompt	Optional
5	@sys-time	$appointment_time	Enter prompt	Optional

If no slots are pre-filled, ask this first:

Okay. Who or what are you here for?

Okay. Are you here for an event or employee?

Enter a variation

Then respond with

	IF ASSISTANT RECOGNIZES	RESPOND WITH
1	$event	You come for $event event..
2	$employee	So, you have a meeting with $employe
3	$employee && $appointment_time	Let me check, $employee at $appointr

And finally:

Wait for user input

Figure B.3: In the root node 'Appointment'. The node gets selected if the intent appointment has been determined. One can check for entities already told by the user so questions for that data can be skipped. If not, it will be asked for and a according response will be given.

C
Robot Platform Decision Matrix

The following five pages illustrate a filled-in example of using the Robot Platform Decision Matrix constructed in Chapter 9. It is based on the final product used in the second evaluation. Filled-in importance levels and the platform to use on the first two tabs have been based on the use case presented in Section 3.1. For the third tab on Software engineering actual line numbers for the two developed products were used, as can be found in Section 4.3. Since this represents the cloud product, all components use the cloud as the chosen option except for vision. This shows how mixed choices are handled in the matrix, which was also a part of the actual implementation in the used product. ’No’ was chosen for the framework since we had to build CASperSocket. The final tab on cost calculation has some fictional numbers for budget, the period of use, number of robots, developer costs, and robot and accessories purchase costs. The theoretically calculated operating costs from Section 8.6 have been filled in, which are the default values.

In the Total tab, the results of this exercise are shown. The blue marked column shows the results for the choices made before. With our configuration of architecture and by using a mix of the toolboxes for the components, we achieve higher total scores compared to using one single toolbox with a lower Total Cost of Ownership.

Approach to fill in the decision sheet:
1. Follow the tabs in given order and fill in the required data in the green colored cells. Each tab has seperate instructions available.
2. After filling all required data in all tabs, return to this tab which shows all calculated values using your input. Compare the numbers to make an informed decision.

Category	**full NAOqi**	**Chosen solution (Mixed)**	**full Cloud**	
Interaction score	113	216	191	
Quality and robustness	46	88	88	
Total score	**159**	**304**	**279**	
Software Engineering				
Development time (developer months)	16.84	7.33	7.96	
Development complexity (1-5)	2.2	1.8	1.8	
Costs				
One-Time Costs	154800	78750	83800	Including robot purchase, development, hardware
Operating costs	0	3276	7428	Costs for running cloud services over the total period of use
Maintenance	600	1200	1200	Maintenance costs for the total period of use
Total cost of ownership	**155400**	**83226**	**92428**	

Component	**Chosen toolkit**
Speech	`Cloud`
Interaction & logic	`Cloud`
Dialog	`Cloud`
Vision	`NAOqi`
Scenario	`Cloud`

INSTRUCTION: Look at the interaction indicators and determine per indicator how important (on a 5 point likert scale) this is for the social robot application.
Each indicator has additional instructions if needed.
Next, Choose which platform would be best suited for this indicator if needed. Direct advice on this choice is provided in the score column as some indicators might work better on the other platform. Change the platform according to your liking using this advice.
The final score per platform and for your choices is displayed at the bottom.

	Interaction indicators	**Importance (1-5)**	**Platform: NAOqi / Cloud**	**Score**	Naoqi score	Cloud score	**Instruction**
Speech	STT: Use of unrestricted/free speech input	4	Cloud	100%	12	36	Rate how important unrestricted (general) speech is for the robot application; increase your score if unrestricted language input is extensively needed in the application. Decrease the score if keyword based recognition is enough.
	STT: Ability to operate in noisy environments	4	Cloud	100%	8	28	Rate the importance of using the robot application in an environment with a lot of background noise, such as crowded events, festivals with music or other location with many people and devices making noise.
	STT: Low Word Error Rate	5	Cloud	100%	20	25	Rate how important accurate transcriptions or keyword matchings are. The KER of NAOqi is 32% vs Google <27%.
	STT/TTS Needs support for multi-language	2	Cloud	100%	4	14	Rate the importance for support of more than two languages with STT or TTS.
Interaction & logic	Duration of interactions are long and/or require many different robot functionalities	4	Cloud	100%	12	20	Rate how important it is for the robot to handle complex interactions having many states and/or processing various modalities and functionalities simultaniously.
	Needs support for multi-language	1	Not needed	Missing Platform			Rate the importance of support of more than two languages for both input and output in the full interaction including interfaces, more languages represent a higher importance.
Vision	Need (precise, often) recognition of objects	1	Not needed	Missing Platform			Rate how important object recognition is for your application; increase your score if computer vision is extensively used in your application.
	Needs face detection/classification	5	NAOqi	100%	45	20	Rate the importance of low-latency detection of faces in view of the robot.
	Needs face recognition / human metrics (age, emotion)	2	Cloud	100%	4	12	Rate the importance of accurate recognition of faces for small to large amounts of people. Increase the importance if age and emotion detection is also valueble for the interaction
Dialog	Dialog is large with many branching points requiring nlp and/or nlc. (Opposed to only yes/no closed answer space)	4	Cloud	100%	8	36	Rate how important it is to perform complex conversations compared to simple yes/no dialogues. If the conversation becomes more complex by the need to collect and reuse data, perform many (conditional) branches, support multiple users or query external services, increase the importance.

				NAOqi	**Chosen score**	**Cloud**
Interaction score				113	**216**	191

INSTRUCTION:
As with the previous tab, fill in an importance value per quality or robustness indicator followed by a choice of platform if needed.
The total score is displayed at the bottom.

	Quality and robustness indicators	**Importance (1-5)**	**NAOqi / Cloud**	**Score**	NAOqi score	Cloud score	**Instruction**
	Must run autonomously at all times	4	Cloud	100%	12	28	Rate the importance of the robot being able to run all day without human intervention. Rater higher if the robot needs to solve more problems and situations by itself, next to have a stable running system.
	Quick (near to realtime) processing times	2	Cloud	29%	14	4	Rate the importance of low-latency processing of all sensors, with the change of lower accuracy. The more one needs this, the higher the importance rating.
	High recognition rate in general	4	Cloud	100%	16	32	Rate the importance of high performance and accuracy, at the possible cost of increased latency.
	Remote/live updating required	2	Cloud	100%	2	16	Rate the importante for the option to perform updates and control without requiring direct access to the robot, and while it is in use.
	Can drive to different position	1	Not needed	Missing Platform			Rate the importance of mobility for the robot, and how well it can map its surroundings
	Trained Posture control (get a hand, catching a ball)	1	Not needed	Missing Platform			Rate the importance of accurate and quick joint control
	Language recognition	2	Cloud	100%	2	8	Rate the importance of the ability of the robot to detect in which language it is spoken to

	NAOqi	**Chosen score**	**Cloud**
Quality and robustness score	46	**88**	88

INSTRUCTION:
First, determine whether a base framework is available for the cloud by setting it to Yes or No. This reduces development time significantly and will therefore grey out a part of the sheet.
Then, determine your final choice per component based on the choises made in the Interaction tab. Estimate how many lines of code are needed to develop this part (for both systems if one wants to compare resuls), or leave the defaults if unkown.
Also enter the estimated lines of code for scenario development and other elements of code.
An estimated amount of developer months is calculated using the model described in Chapter 8 section 6 of the Thesis. Change these calculated values if a better estimation can be made by oneself.
Automatically a complexity score (1-5) is calculated based on the platform choice per component. If one feels the complexity differs, change the default values.

Base Framework Available	No

Instruction: If there a base framework available for the development of the application so that one only has develop a scenario, tick 'Yes' and skip the greyed out part of the table below. Otherwise select 'No'.

Component	**NAOqi / Cloud**	**Complexity NAOqi (1-5)**	**Lines of code estimation NAOqi**	**Complexity Cloud (1-5)**	**Lines of code estimation Cloud**	**Estimate chosen (dev months)**	**Instruction**	**Estimate full NAOqi (dev months)**	**Estimate full Cloud(dev months)**
Speech	Cloud	1	30	1	269	1.16	Look at the interaction tab component categories and determine the average choice made per component. Example:3/4 indicators of Speech state Cloud, the average choice is cloud. Fill in this choice in the NAOqi/cloud column. After this, estimate the lines of code needed to build this component, fill in 0 if this component is not used, or leave the default estimation.	0.15	1.16
Interaction & logic	Cloud	3	0	3	150	0.67		0	0.67
Dialog	Cloud	3	30	2	245	1.06		0.15	1.06
Vision	NAOqi	1	30	1	175	0.15		0.15	0.78
Scenario development & testing	Cloud	3	4660	2	1100	4.29		16.39	4.29
Other	Not needed								

Development time	**7.33**	If a better estimation can be made for amount of developer months, change the value here	**16.84**	**7.96**
Development complexity average (1-5)	**1.8**		**2.2**	**1.8**

INSTRUCTION:
Enter the required information in the green colored cells or keep the default values.
Some data is automatically imported from other tabs but can be changed if better estimate
At the bottom the total costs of ownership are displayed.

Cost indicators		**Instruction**
Available Budget	100000	Enter the available budget
Period of use (months)	12	Fill in the period the robot product is expected to be used in months
Number of robots per application	1	Enter the number of robots using the to be developed product
Developer cost per hour (eur)	50	Enter the cost for one developer hour.
Hours of development (months * 160h)	1173	This number represents the total amount of development hours, gathered from the Software engineering tab

Operational costs Cloud per component		
Operating costs Speech (month)	144	Enter the calculated monthly costs for the components. Leave the defaults if unknown.
Operating costs Server (month)	114	
Operating costs Dialog (month)	15	
Operating costs Vision (month)	346	

One time costs	**only NAOqi**	**Chosen solution (mixed)**	**only Cloud**	**Instructions**
Base framework	0	0	0	Enter the costs for the used base framework. Keep it at zero if NAOqi is used.
Robot(s) purchase	20000	20000	20000	Enter the cost for the robot(s) purchase
Accessories (Charging point, router, Mifi)	100	100	100	Enter accesories costs if existing infrastructure cannot be used. Think of a router, charging station for the robot, a mifi router for 4G Connectivity, etc.
Development (components * dev. costs.)	134700	58650	63700	This is the result of the calculated development time multiplied by developer costs
Subtotal One-Time Costs	**154800**	**78750**	**83800**	

Operating costs				
Operating costs Speech	0	1728	1728	The choices from the previous tabs are copied. The operational costs per component (if Cloud is chosen) for the total period of use are displayed
Operating costs Server	0	1368	1368	
Operating costs Dialog	0	180	180	
Operating costs Vision	0	not used	4152	
Subtotal Operating costs	0	3276	7428	These are the operational costs for the total period of use
Maintenance/updates	600	1200	1200	These are the maintenance costs for the total period of use
TCO Total cost of ownership	**155400**	**83226**	**92428**	
	Over Budget	Within budget	Within budget	

Bibliography

[1] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4960–4964. IEEE, 2016.

[2] Nikolaos Mavridis. A review of verbal and non-verbal human–robot interactive communication. *Robotics and Autonomous Systems*, 63:22–35, 2015.

[3] David Fischinger, Peter Einramhof, Konstantinos Papoutsakis, Walter Wohlkinger, Peter Mayer, Paul Panek, Stefan Hofmann, Tobias Koertner, Astrid Weiss, Antonis Argyros, et al. Hobbit, a care robot supporting independent living at home: First prototype and lessons learned. *Robotics and Autonomous Systems*, 75:60–78, 2016.

[4] Vladimir Murashov, Frank Hearl, and John Howard. Working safely with robot workers: Recommendations for the new workplace. *Journal of occupational and environmental hygiene*, 13(3):D61–D71, 2016.

[5] Corinna Underwood. Robots in retail - examples of real industry applications -, Oct 2017. URL https://www.techemergence.com/robots-in-retail-examples/.

[6] Jodi Forlizzi and Carl DiSalvo. Service robots in the domestic environment: a study of the roomba vacuum in the home. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 258–265. ACM, 2006.

[7] Brian Richard Duffy, CFB Rooney, Greg MP O'Hare, and RPS O'donoghue. *The social robot*. PhD thesis, University College Dublin, 2000.

[8] Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. Integration of cloud computing and internet of things: a survey. *Future generation computer systems*, 56:684–700, 2016.

[9] Jiafu Wan, Shenglong Tang, Hehua Yan, Di Li, Shiyong Wang, and Athanasios V Vasilakos. Cloud robotics: Current status and open issues. *IEEE Access*, 4:2797–2807, 2016.

[10] marketsandmarkets. Cloud robotics market by component (software and services), service model (iaas, paas, and saas), application, deployment model (public, private, and hybrid cloud), end-user (verticals and third-party users), and region - global forecast to 2022, Feb 2018. URL https://www.marketsandmarkets.com/Market-Reports/cloud-robotics-market-1035.html.

[11] Salvador Manuel Malagon-Soldara, Manuel Toledano-Ayala, Genaro Soto-Zarazua, Roberto Valentin Carrillo-Serrano, and Edgar Alejandro Rivas-Araiza. Mobile robot localization: A review of probabilistic map-based techniques. *IAES International Journal of Robotics and Automation*, 4(1):73, 2015.

[12] Rajesh Doriya, Paresh Sao, Vinit Payal, Vibhav Anand, and Pavan Chakraborty. A review on cloud robotics based frameworks to solve simultaneous localization and mapping (slam) problem. *arXiv preprint arXiv:1701.08444*, 2017.

[13] Vittorio Perera, Tiago Pereira, Jonathan Connell, and Manuela M. Veloso. Setting up pepper for autonomous navigation and personalized interaction with users. *CoRR*, abs/1704.04797, 2017. URL http://arxiv.org/abs/1704.04797.

[14] A Pandey, S Pandey, and DR Parhi. Mobile robot navigation and obstacle avoidance techniques: A review. *Int Rob Auto J*, 2(3):00022, 2017.

[15] Panagiota Tsarouchi, Sotiris Makris, and George Chryssolouris. Human–robot interaction review and challenges on task planning and programming. *International Journal of Computer Integrated Manufacturing*, 29(8):916–931, 2016.

[16] Erann Gat et al. On three-layer architectures. *Artificial intelligence and mobile robots*, 195:210, 1998.

[17] Mélanie Garcia, Lucile Béchade, Guillaume Dubuisson-Duplessis, Gabrielle Pittaro, and Laurence Devillers. Towards metrics of evaluation of pepper robot as a social companion for elderly people.

[18] Michiel de Jong, Kevin Zhang, Aaron M Roth, Travers Rhodes, Robin Schmucker, Chenghui Zhou, Sofia Ferreira, João Cartucho, and Manuela Veloso. Towards a robust interactive and learning social robot. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 883–891. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[19] Roberto Pinillos, Samuel Marcos, Raul Feliz, Eduardo Zalama, and Jaime Gómez-García-Bermejo. Long-term assessment of a service robot in a hotel environment. *Robotics and Autonomous Systems*, 79:40–57, 2016.

[20] Kheng Lee Koay, Kerstin Dautenhahn, SN Woods, and Michael L Walters. Empirical results from using a comfort level device in human-robot interaction studies. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 194–201. ACM, 2006.

[21] Edward Twitchell Hall. The hidden dimension. 1966.

[22] Jeroen Linssen and Mariët Theune. R3d3: The rolling receptionist robot with double dutch dialogue. In *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 189–190. ACM, 2017.

[23] Sahdev Zala, NailahMusaid, and Erin McKean. Robotic calculations and inference agent, Jul 2016. URL https://developer.ibm.com/code/patterns/robotic-calculations-and-inference-agent/.

[24] Labib Sadek Terrissa, Bouziane Radhia, and Jean-François Brethé. Towards a new approach of robot as a service (raas) in cloud computing paradigm. In *5th. International Symposium ISKO-Maghreb (Knowledge Organization in the perspective of Digital Humanities: Researches and Applications)*, 2015.

[25] Markus Waibel, Michael Beetz, Javier Civera, Raffaello d'Andrea, Jos Elfring, Dorian Galvez-Lopez, Kai Häussermann, Rob Janssen, JMM Montiel, Alexander Perzylo, et al. Roboearth-a world wide web for robots. *IEEE Robotics and Automation Magazine (RAM), Special Issue Towards a WWW for Robots*, 18(2): 69–82, 2011.

[26] Dominique Hunziker, Mohanarajah Gajamohan, Markus Waibel, and Raffaello D'Andrea. Rapyuta: The roboearth cloud engine. In *2013 IEEE International Conference on Robotics and Automation*, pages 438–444. IEEE, 2013.

[27] Guoqiang Hu, Wee Peng Tay, and Yonggang Wen. Cloud robotics: architecture, challenges and applications. *IEEE network*, 26(3):21–28, 2012.

[28] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing—the business perspective. *Decision support systems*, 51(1):176–189, 2011.

[29] Abdulaziz Aljabre. Cloud computing for increased business value. *International Journal of Business and social science*, 3(1), 2012.

[30] Xiong Xiao, Shengkui Zhao, Douglas L Jones, Eng Siong Chng, and Haizhou Li. On time-frequency mask estimation for mvdr beamforming with application in robust speech recognition. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3246–3250. IEEE, 2017.

[31] Takuya Higuchi, Nobutaka Ito, Takuya Yoshioka, and Tomohiro Nakatani. Robust mvdr beamforming using time-frequency masks for online/offline asr in noise. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5210–5214. IEEE, 2016.

[32] Wilson Rosa, Ray Madachy, Barry Boehm, and Brad Clark. Simple empirical software effort estimation model. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, page 43. ACM, 2014.

[33] Christoph Bartneck, Dana Kulić, Elizabeth Croft, and Susana Zoghbi. Measurement instruments for the anthropomorphism, animacy, likeability, perceived intelligence, and perceived safety of robots. *International journal of social robotics*, 1(1):71–81, 2009.

[34] M. I. Anisa Fardhani Prasetyaningtyas. Embodying a conversational agent into a robot: exploring the effect of different conversation recovery strategies on people's perception towards the robot's social attributes, 2018. URL `https://research.tue.nl/en/studentTheses/embodying-a-conversational-agent-into-a-robot`.

[35] Youngja Park, Siddharth Patwardhan, Karthik Visweswariah, and Stephen C Gates. An empirical analysis of word error rate and keyword error rate. In *Ninth Annual Conference of the International Speech Communication Association*, 2008.

[36] J. Sparreboom. Evaluating an asr pipeline for a social robot. TU Delft Master Thesis, 2019.

[37] Steve Renals, Thomas Hain, and Hervé Bourlard. Interpretation of multiparty meetings the ami and amida projects. In *2008 Hands-Free Speech Communication and Microphone Arrays*, pages 115–118. IEEE, 2008.

[38] Jon Barker, Ricard Marxer, Emmanuel Vincent, and Shinji Watanabe. The third 'chime'speech separation and recognition challenge: Dataset, task and baselines. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 504–511. IEEE, 2015.

[39] Kallirroi Georgila, Carla Gordon, Hyungtak Choi, Jill Boberg, Heesik Jeon, and David Traum. Toward low-cost automated evaluation metrics for internet of things dialogues. In *Proceedings Ninth International Workshop on Spoken Dialogue Systems Technology (ISWDS 2018), Singapore*, 2018.

[40] Helen Hastie. Metrics and evaluation of spoken dialogue systems. In *Data-Driven Methods for Adaptive Spoken Dialogue Systems*, pages 131–150. Springer, 2012.

[41] Marilyn Walker, Candace Kamm, and Diane Litman. Towards developing general models of usability with paradise. *Natural Language Engineering*, 6(3-4):363–377, 2000.

[42] Susan Robinson, Antonio Roque, and David R Traum. Dialogues in context: An objective user-oriented evaluation approach for virtual human dialogue. In *LREC*, 2010.

[43] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[44] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Number 07-49, October 2007.

[45] Barry W Boehm et al. *Software engineering economics*, volume 197. Prentice-hall Englewood Cliffs (NJ), 1981.

[46] Vu Nguyen, Barry Boehm, and Phongphan Danphitsanuphan. A controlled experiment in assessing and estimating software maintenance tasks. *Information and software technology*, 53(6):682–691, 2011.

[47] Irene Buchmann, Sebastian Frischbier, and Dieter Putz. Towards an estimation model for software maintenance costs. In *2011 15th European Conference on Software Maintenance and Reengineering*, pages 313–316. IEEE, 2011.

Glossary

beamformer Software capable of converting multi-channel audio to single channel, with the main functionalitiy of enhancing the audio signal (in our case speech) coming from a certain angle, and reducing signals coming from all other angles. This process enables the robot to understand the user's speech even in noisy surroundings, where otherwise the audio signal would have a too low quality for speech recognition.. 10, 40, 51

happy flow The interaction flow which a company (or other robot host) wants a user to take. This does not mean that other ways of going through the interaction is unwanted, but the happy flow describes the best way a user could be helped, either for the user itself or for the interests of the company. In the use case of this thesis, this means the flow: engage person => confirm the user has an appointment => identify the employee for the appointment => identify user => optionally play a game => give feedback on the interaction.. 43, 45, 55

interaction step A moment during an interaction where the system waits for user input, after which the system switches to another state based on that input. Multiple consecutive interaction steps form a dialog.. 34

Pepper Toolbox Set of tools used to program and control the Pepper robot (and other robots running NAOqi OS). The toolbox includes Choregraph (flow-based drag&drop programming), NAOqi (higher-level operating system) and a set of SDK's and API's for various programming languages.. 2, 4, 7, 30, 32, 35, 48, 59, 63, 64, 66, 67, 79

trade-offs Implementing a system component of the robot using the NAOqi toolbox or using the Watson toolbox has its pros and cons. These trade-offs usually involve costs, developer effort, performance and/or accuracy differences.. 4

Transcribing Transcribing is to convert a representation of language, typically speech but also sign language, etc., to another representation. In this thesis, the process of converting speech to text is typically meant.. 42

transcription A transcription is something that has been transcribed, in our case a representation of speech in the form of text.. 42

Watson Toolbox Collection of IBM Watson Cloud services which can be used to replace or add to built-in modules running on the Pepper robot. This toolbox includes a form of control logic, but this logic is not defined explicitly as multiple solutions are available for this. Example of this could be Project Intu or Node-RED.. 3, 4, 7, 30, 63, 64, 67, 79