University of Stavanger

Faculty of Science and Technology

BACHELOR'S THESIS

Study program/ Specialization: Datateknologi	Spring semester, 20 22 Open / Restricted access
Writer: Rune Henriksen, Sindre E. Mosand	Rune Henriksen, Sindre E. Mosand (Writer's signature)
Faculty supervisor: Krisztian Balog, Ivica Kostric External supervisor(s):	
Thesis title: A Conversational Framework for Multiple Modalities	
Credits (ECTS):	
Key words: Social Robot, Conversational AI	Pages: 52 + enclosure: ………… Stavanger, 15.05.22 Date/year

University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

A Conversational Framework for Multiple Modalities

Bachelor's Thesis in Computer Science
by

Rune Henriksen
and
Sindre Ekrheim Mosand

Internal Supervisors
Krisztian Balog
Ivica Kostric

May 15, 2022

Abstract

This paper is concerned with creating conversations around the ECIR conference with multiple modalities. We use a Furhat robot from Furhat robotics to have a physical presence and handle voice. Rasa addresses the AI component of the system. First, we develop a framework for managing messaging between Rasa AI with the Furhat frontend and a displayed frontend. Second, we create a protocol that describes how to format messages for this connection. The results of the thesis discuss the conversations from the ECIR conference. We evaluate the log from the conference and do both statistical analysis and a discussion of the qualitative results. The thesis also describes the general principles of conversational AI in the background chapter.

Acknowledgements

We would like to thank our supervisors for their fantastic enthusiasm and help with writing this thesis. Also special thanks to Tølløv Aresvik for styling our custom frontend.

Contents

Abstract iii

Acknowledgements iv

1 Introduction 3
1.1 The DAGFiNN project . . . 3
1.2 Objectives . . . 5
1.3 Approach and Contributions . . . 5
1.4 Outline . . . 6

2 Background 7
2.1 Conversational AI . . . 7
2.1.1 Types of Dialogue Systems . . . 8
2.1.2 Reasoning behind choice of dialogue system . . . 9
2.1.3 Main Components of Task-oriented Dialogue Systems . . . 10
2.2 Rasa . . . 13
2.3 Social robots . . . 15
2.3.1 Furhat robot . . . 16
2.4 Multiple modalities . . . 18

3 Approach 21
3.1 Architecture . . . 22
3.1.1 Rasa servers . . . 23
3.1.2 Details on choosing API . . . 23
3.1.3 User interfaces . . . 24
3.1.4 Custom connector . . . 26
3.1.5 A well behaving Furhat . . . 28
3.2 Message protocol . . . 29
3.2.1 Emojis . . . 29
3.2.2 Facial expressions . . . 29
3.2.3 Maps . . . 31

4 Results 33
4.1 Quantitative Results . . . 33
4.1.1 Filtering results . . . 33

4.1.2 Useful replies 34
4.1.3 Length of conversations 35
4.1.4 Response to intents 35
4.2 Qualitative results 36
4.3 Methodology of in-house testing 38
4.4 Furhat responses in testing phase 38

5 Conclusions 41
5.1 Analysis of work done 41
5.2 Analysis of the conference results 42
5.3 Future Directions 42

A Instructions to Run System 45

B Message protocol 49

Bibliography 53

Division of Labour

	Rune	Sindre
Introduction	50%	50%
Background	25%	75%
Approach	50%	50%
Results	75%	25%
Conclusion	75%	25%
Web-chat	25%	75%
Chat-log	25 %	75%
Kotlin-kode	75%	25%
Custom-connector	75%	25%
Carousel	25%	75%
Message protocol	100%	
How-to-run-documentation	100%	

Chapter 1

Introduction

Conversational AI is gaining in both the number of application use-cases and mind-share. It is now commonplace for companies to use chatbots, especially in customer service. The prospect of further reducing salary costs can give these companies an incentive to use conversational AI to improve and scale their solutions.

The use cases are not limited to customer service but also include sales assistance, therapy, conversational partner for the elderly, or anyone wanting to chat with someone.

Advances in hardware in later years have opened up considerable opportunities for AI research. Researchers now have access to massive data sets and increased computational power compared to before. This has made more practical applications in this field of technology possible.

This thesis explores conversational AI with a physical speaking robot in Furhat robot 1.1 from Furhat Robotics. The robot is powered by Rasa AI capabilities in the project DAGFiNN where we make a conversational assistant.

1.1 The DAGFiNN project

The DAGFiNN project has an actual physical robot 1.1, with speech recognition capabilities as well as the ability to speak and make facial expressions, mimicking an actual human. We refer to it as the Furhat robot or simply the Furhat.

Figure 1.1: Furhat robot with a screen

The physical presence of the Furhat robot is significant. It feels more natural to talk to a robot with human features like eyes and facial expressions.

The DAGFiNN project will focus on Stavanger's European Conference on Information Retrieval (ECIR) 2022. The main dialogue for the conversational assistant will evolve around this conference. One task for the Furhat is to help find a place to eat in Stavanger. Another job is to answer questions regarding the conference.

The DAGFiNN project consists of several teams. Each team has specific tasks that all contribute to the more significant project. The other groups that are part of the greater DAGFiNN-project are as follows:

- The Conference Team : Creating dialogues with information from the conference.
- The Point of Interest team : Create dialogue with information about the Stavanger area and places to eat.
- Personalization team : Work out how to personalize the robot interaction, e.g let the robot recognise users etc.
- Framework team : Develop framework and message protocol

The Framework team focuses on two conversational modalities, text/image, and voice. The DAGFiNN project discusses using two user interfaces; webchat on a web page and voice communication with Furhat with a screen to display chat transcript and images.

All teams in the DAGFiNN project contribute to the same codebase:
github.com/iai-group/DAGFiNN

1.2 Objectives

The framework's main objective is to enable users to interact with DAGFiNN conversational system on multiple modalities. The modalities are the text and images displayed on a screen and voice through the Furhat robot. This can be viewed as a two-step process. First, implement user interfaces, connect with a backend, and create a message protocol that supports it. The interface is a webchat on a webpage, and, Furhat with a screen next to it. The latter is the primary user interface used in this project.

- Research and present theoretical background information.
- Develop a framework for multiple modalities.
- Develop a message protocol for the framework.
- Evaluate the results from the conference.

1.3 Approach and Contributions

There are challenges in handling conversations on the different modalities. For example, sending messages in real-time has to be addressed and presented in a package that suits the modality.

It is essential to handle various messages, like images, emojis, text, and buttons for interactivity.

This can be summarized as the contributions of this team, listed below.

- Socket.IO integration
- Message protocol
- Webchat
- Furhat-screen
- Instructions manual

1.4 Outline

Chapter 2 covers the background material for understanding conversational AI and the technologies involved in this project, namely Furhat and Rasa AI. This background info will make it easier to understand what the team describes in Chapter 3.

Chapter 3 discusses how the team approached solving the stated problems and reaching the paper's goals.

Chapter 4 shows statistics from the conference and project-participants evaluation of the system's performance. This Chapter also shows results from testing and verifying the robot's capabilities.

Chapter 5 concludes the thesis with the team's thoughts on the progress, evaluation, and what can be future work.

Chapter 2

Background

This Chapter will provide the theory behind conversational AI. This term casually denotes a broad range of systems capable of understanding natural language and generating some response that mimics human dialogue [1]. However, this understanding has its limits.

First, we discuss the bird's eye view of Conversational AI. Then we look at the inner workings of this AI by looking at the types of dialogue systems. Furthermore, information about Rasa and the Furhat robot. In the end, we look at previous work in this field with multiple modalities.

2.1 Conversational AI

In real life, we notice that robots are entering our lives. We interact with them on diverse platforms, and we communicate with them by text and voice-based systems. This is something that previously we only observed in popular cultures, like in the movies.

In Figure 2.1 there is a dialogue sequence from fiction. This scene plays out a dialogue between a human and a robot. The robot only follows its protocol in this situation. In the real world, we create robots with certain domains of understanding and protocols to follow. They can only function within a given limit.

B1 BATTLE DROID: Halt.

QUI-GON JINN: I'm ambassador to the supreme chancellor. I'm taking these people to Coruscant.

B1 BATTLE DROID: Where are you taking them?

QUI-GON JINN: To Coruscant.

B1 BATTLE DROID: Coruscant? Uh, that doesn't compute. Uh, wait. Uh, you're under arrest.

Figure 2.1: Dialogue between robot and human from Star wars episode I the phantom menace (1999)

We delve into the theory behind the dialogue systems powering conversational AI.

2.1.1 Types of Dialogue Systems

There are different types of dialogue systems in the scientific field of conversational AI. Dialogue systems or conversational agents are programs that communicate with human users in natural language [2].

Conversational Dialogue systems discuss two-way or three-way categorization. In two-way, it falls under two classes: Task-oriented dialogue systems and Chatbots [2].

In three-way categorization, discussion falls under three classes: Task-oriented, Social chat, and Interactive QA [3, 4]. See Table 2.2 for the characteristics of the three types.

Task-oriented is a dialogue system that follows a clearly defined structure that is explicitly related to completing particular tasks in a closed domain [1].

In the 'Task-Oriented' chat, a conversation should be brief and efficient. Simple requests and short answers fall into this type in a single-turn, one-question one-answer fashion.

Social chat aims to carry an extended conversation that mimics human-to-human conversation. The objective is to be as human-like as possible and be able to talk on various topics extensively. It is developed for unstructured and open conversations[1]. Social chat is mainly conversations for the pleasure of having someone to engage in conversation. This works as a substitute for communication with another human.

The most difficult challenge in conversational AI is approaching Social-chat. Since humans can talk about whatever falls in their minds, this can be hard to follow for a conversational system. In many cases, this reflects how human-to-human conversations work; sometimes, they can be hard to follow.

Interactive QA aims to provide concise and direct answers to user queries. The conversation follows a question-answering pattern. This type is evaluated by the correctness of the system's solutions [1].

A real-life example would be ordering food from a waiter in a restaurant. The ordering itself is considered Task-oriented, but getting there often means many questions about the dishes. So a pattern would be a customer asking about dishes on a menu and a waiter answering. This pattern will be in a QA multi-turn fashion.

Task-oriented	**Social chat**	**Interactive QA**
Aim to assist users to solve **a specific task** (as efficiently as possible)	Aim to **carry on an extended conversation** ("chit-chat") with the goal of mimicking human-human interactions	Aim to **provide concise, direct answers** to user queries
Dialogues follow a **clearly designed structure** (flow) that is developed for a particular task in a closed domain	Developed for **unstructured, open domain** conversations	Dialogues are **unstructured**, but commonly follow a question-answer pattern; mostly **open domain** (dictated by the underlying data)
Well-defined measure of performance that is explicitly related to **task completion**	Objective is to be **human-like**, i.e., able to talk about different topics (breadth and depth) in an engaging and coherent manner	Evaluated with respect to the **correctness of answers** (on the turn level)

Figure 2.2: Categorization of conversational AI systems taken from [1], based on [3, 4].

With the types of dialogue systems listed, we can better understand which dialogue system is the best for our use-case.

2.1.2 Reasoning behind choice of dialogue system

Behind dialogue systems, there are some algorithms that enable communication with humans in natural language [2]. Moreover, the algorithms used for understanding natural language and generating natural responses are advanced.

Social chat can handle some simple cases that we have predicted based on what we think people attending the conference will ask it.

Though there are specific differences between Task-oriented and Interactive QA, they have the same principal component.

In this thesis, we use Task-oriented Dialogue System since this is most conducive to our goals of asking simple questions with clear-cut answers. Next, we explain the components by which this Dialogue system is built.

2.1.3 Main Components of Task-oriented Dialogue Systems

The main components of Task-oriented Dialogue Systems are Natural Language Understanding (NLU), Dialogue Management (DM), and Natural Language Generation (NLG). Figure 2.3 shows the three main components with sub-components. The names of the sub-components can vary in different articles, frameworks, and platforms; however, they have the same meaning.

Natural Language Understanding - NLU

The task of the NLU is to classify user input. This classification is performed by AI using neural networks, which is an advanced approach. By using AI, this field has achieved high performance in conversational dialogue systems [4].

NLU has sub-components that perform intent determination and domain detection. NLU is executing this classification on utterances with the current user in mind. The NLU is a pre-processing step for later components in dialogue systems and has a significant impact on the system's overall quality [4]. Further, we need to explain the following terms.

Intents, Domain, Entities and Slots

Intent describes what the dialogue system understands, e.g., the system understands that the user intends to 'greet' if they say 'Hi.'

The conversational assistant operates in a domain with content it should know about and be able to handle. There is often a domain for intents that are out of scope. A trained NLU can recognize all or at least most intents that do not fit. Custom out-of-scope replies commonly handle these utterances.

The entity is a keyword or a structured piece of information from an utterance. This information extracted from the user input is the system's information to achieve the user goals [5].

Slots are considered a memory of a conversation which gives the ability to have a back-and-forth conversation with meaning. This information could, for instance, be about a user's home city or as well as information gathered from a database.

In many situations, the present utterance alone can be ambiguous or lack the information it needs. When making slot-tagging predictions, slot filling helps the system learn which part of contextual information it should attend [4].

The task of slot filling is to classify the associated sentences with the correct set of slots, domain, and intent. This can be done, for example, by training a sequence model to map from input words representing slot fillers, domain, and intent [2].

Entity, Slot, and Intent are all a part of the conversational agent's same domain [5].

Dialogue Management - DM

The dialogue manager is the component of task-oriented dialogue that is the decision-making module in the system. This is also referred to as the dialogue policy. The task of the DM is to choose system actions at each step to guide the dialogue to successful task completion [6].

The DM maps the dialogue state to a system action. It receives a new observation by the NLU and emits a system dialogue to the NLG [6]. It operates at a semantic representation level between the NLU and NLG.

Natural Language Generation - NLG

NLG is about generating a response. This task is often modeled in two stages: Content planning (what to say) and Sentence realization (how to say it). The chosen dialogue to continue the following sentence is generated with some attributes. These are the slots and values that will be used as part of an answer, or a confirming strategy [2].

There are several approaches to language generation. Template- or rule-based ones are the most commonly used in practice. A set of templates or rules is used to select a proper candidate amongst several valid answers to generate sentences [4]. The solution generated is a response already prepared. These answers can be dynamic or static based on templates and rules. This means that a domain expert designs the answers that the dialogue system replays. Decisions by NLU and DM mechanisms establish the response.

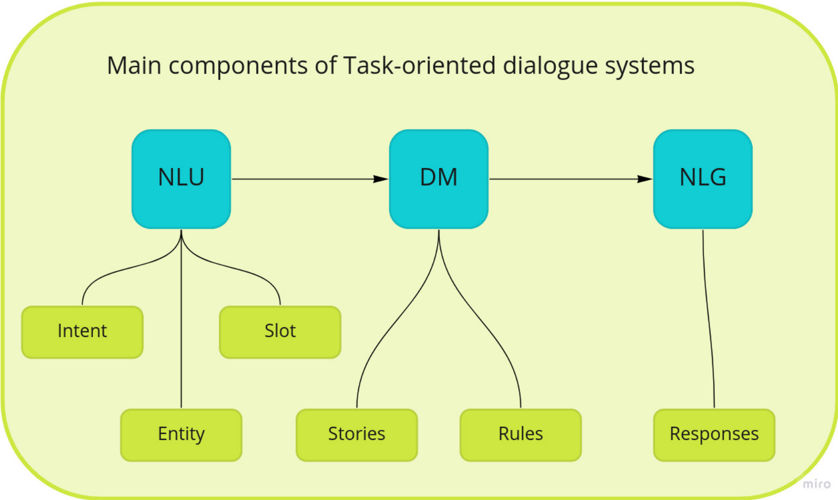

Figure 2.3: Main Components of Task-oriented Dialogue Systems

The arrows in Figure 2.3 describe the direction of how the dialogue system will handle user input. NLU is the starting point when initiated by a user, DM makes decisions on the input, and finally, NLG gives a response back to the user.

The task-oriented dialogue system we use may also fall under the conversational information access category.

Conversational Information Access-systems (CIA)

CIA aims at task-oriented dialogue to support multiple user goals, including search, recommendation, and exploratory information gathering [1].

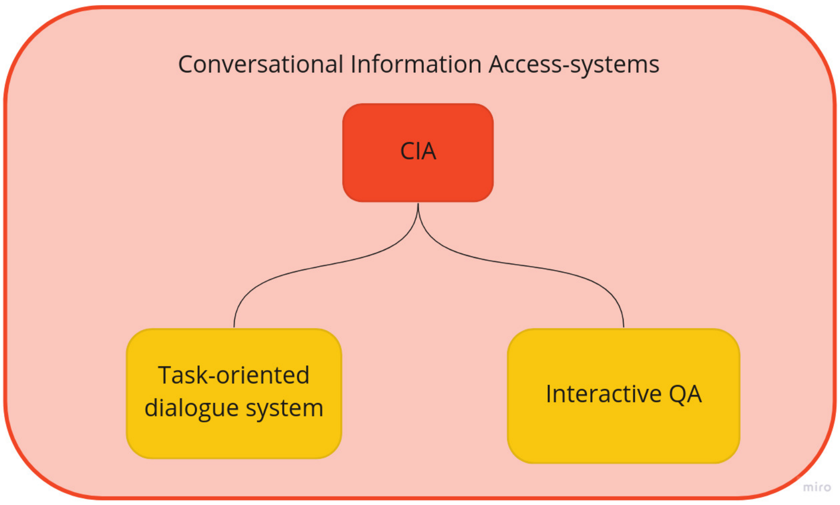

Figure 2.4: Components of Conversational Information Access-systems.

The CIA system is about completing specific tasks and providing concise and direct answers. This is relevant because this project uses a CIA system. We lean towards both sub-component in CIA. In some cases, we have one question and one direct answer, and in other cases, we have a QA pattern.

The tool we use to facilitate the dialogue system is Rasa. So, first, we discuss the details about Rasa.

2.2 Rasa

Rasa is an open-source machine learning framework for automated text and voice-based conversations. Understand messages, hold conversations, and connect to messaging channels and APIs [5].

The NLU component is a part of Rasa Open Source that performs intent classification, entity extraction, and response retrieval [5]. The NLU classifies user input by intent. For example, Figure 2.5 shows examples of how a user might greet the system.

```
- intent: greet
  examples: |
  - hey
  - hello
  - hi
```

Figure 2.5: Example of how Rasa uses Intents to recognize user input

Rules and Stories (DM)

Rules and Stories are handled by the Dialogue Management component in the Rasa frameworks.

Rules and stories have a similar form in Rasa. Rules are meant to handle small and specific tasks, and stories can manage more generalized conversational paths [5]. For example, look to Figure 2.6, where 'rule' can be switched out with 'story' to work as a story. In the same model, the intent is by NLU classified as a 'greeting.'

The next step is to answer the user's intent correctly. That is with a Rasa action. Then, the DM tells which action to use for the recognized intent. With a rule, this is more straightforward, where the intent has a given action.

Stories can take several paths depending on how the NLU classifies the users uttering. The NLU decides which way to follow based on intent recognition. One example could be deciding whether to pursue a 'happy' path or a 'sad' path. Different intents determine what course to take. The NLU should be trained with examples to recognize this.

The intent could also lead to a custom action that can run any code, including API calls, database queries, etc. [5]. Custom actions in stories must be added to the action section in domain.yml. The action should respond with a list of events and responses [5].

```
rules:
- rule: Greet when greeted
    steps:
    - intent: greet
    - action: utter_greet
```

Figure 2.6: Example of Rule

Responses and Events (NLG)

Figure 2.7 shows one response. It is possible to have many examples as text for the response. Rasa picks one randomly every time.

There are several types of events in Rasa. First, a slot is an event. It is automatically tracked by Rasa if filled by an entity of the same name. If not, it must be tracked by a custom action [5].

Rasa automatically tracks some events (e.g., user messages). However, other events can only be tracked if they are returned by a custom action [5].

```
responses:
    utter_greet:
        text: "Hey! How can I help you today?"
```

Figure 2.7: Example of response

Rasa training and models

Rasa NLU training stores structured information about user messages [5].

The Rasa server is responsible for retrieving data from the trained model. Then, the NLU picks up the user intent, and the NLG generates a response based on the decisions by DM.

With the use of a pre-trained language model, the Rasa NLU can classify one or more intents from user messages [7].

In the pre-trained language model, each word is represented as word embedding. That is a vector representation of words, which means that a word is converted to a vector, and similar vectors should represent similar words [7].

Various model architectures compute vector representations of words. This is called word representation in vector space [8].

New training models make it possible to compute very accurate high dimensional word vectors from an extensive set of data with millions of words [8].

Now that we have a firm understanding of the system's backend, we will look at the project's primary user interface—the social robot Furhat.

2.3 Social robots

A social robot is a physical robot distinguishing it from chat-widgets and avatars on websites and screens. The physical aspect is the crux here as it is how humans interact with other humans, at least when we want to connect. Interactions in the physical space lead to deeper and more meaningful experiences. Expressing emotions and seeing emotions expressed on a face is what we humans have evolved to do [9].

> Social robots are the next major user interface, that are typically designed based on the oldest user interface we as humans know - the face [9].

A social robot has a low barrier of entry. As a result, almost everyone can talk even if they are incapable of other types of communication due to limiting factors. This makes the social robot helpful in many cases where other options are not available or can reach a larger audience.

> We are "designed" to interact with people that share the same space as us – which is why we find it easier to trust or build an emotional connection with people when we see them in person, rather than speaking over the phone or Skype [10].

Furhat robotics mentions 7 potential applications of a social robot. These are Employee training, Unbiased recruitment, Medical screening, Helping children with autism, Travel information, Receptionist, Teaching Assistant [11].

Pepper Robot

An example of a social robot is the Pepper robot. This robot is especially great for engaging your visitors, hosting, and entertaining them. [12]

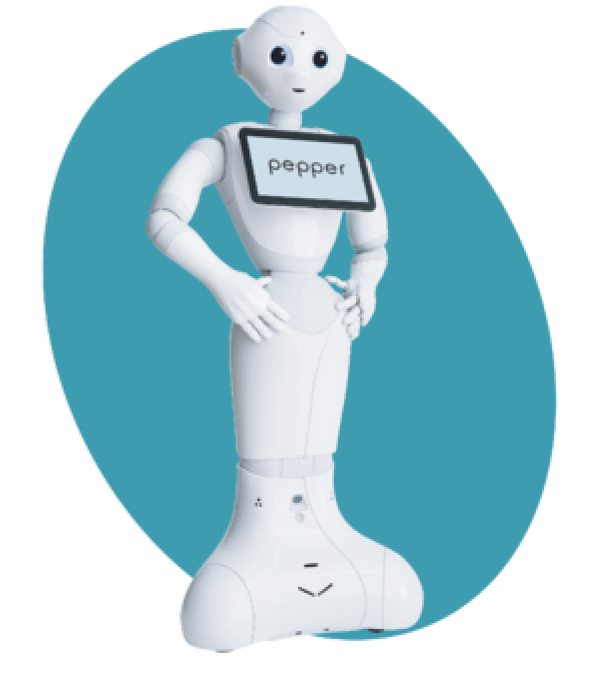

Figure 2.8: Pepper robot waiting to receive customers

As seen in Figure 2.8 the robot comes with a screen, arms, and legs. Pepper is also an option to move around and be a tour guide.

2.3.1 Furhat robot

One such social robot is the Furhat robot, as seen in Figure 2.9. The Furhat robot is meant to be a blank slate. This means that your particular use case must be developed. But it can be done since the framework isn't locked to one specific subject or mode [9].

Figure 2.9: Furhat robot from Furhat Robotics

To get a better overview of the size of the Furhat's we present a table with key dimensions.

Physical Dimensions			
(HxWxD)	410mm	270 mm	240 mm
Eye Height:	300mm		
Weight:	3.5 kg		

The robot comes with a back-projected face. This means that the face is lit up from behind to create facial expressions and gives the robot the ability to change characteristics such as skin color, make-up, being male or female, and placement of eyebrows.

The Furhat robot only consists of a head with no other limbs. The main focus for the Furhat team is to enhance facial expression and voice further as that is more important for an enjoyable conversation without the added complexity from limbs and a robot body [13].

The Furhat also comes with a development SDK, including a virtual version of the Furhat. A virtual version of the Furhat can be seen in Figure 3.9. The SDK allows developers to create working robot systems without constantly testing them on the physical robot. This way, a whole development team can easily share one physical robot.

In the DAGFiNN project, meta-communication such as facial expression is considered part of natural language. The Furhat robot is capable of expressing various emotions that are built-in. These can be customized, which gives the ability to enhance the communication to more than just spoken and written language. It provides a face on which users focus their attention. Talking to a robotic face that mimics human expressions feels more familiar than comparison to some object like Amazon's Alexa.

2.4 Multiple modalities

A screen could easily be available when interacting with a voice system and even a physical robot that speaks. The situations where a user of such systems has different options available are referred to as multiple modalities. For example, a screen could be available in addition to voice.

Even with excellent voice communication, images can enhance the conversation. An example could be a map that is difficult to describe with words. Also, the map may not mean much without an oral explanation. Both modalities enhance each other. [14]

A screen can help a voiced conversation by letting the user know what the system picked up by displaying a transcript of the conversation. This can also help the user get a better overview of options presented by the conversational system, such as the listing shown in Figure 2.10.

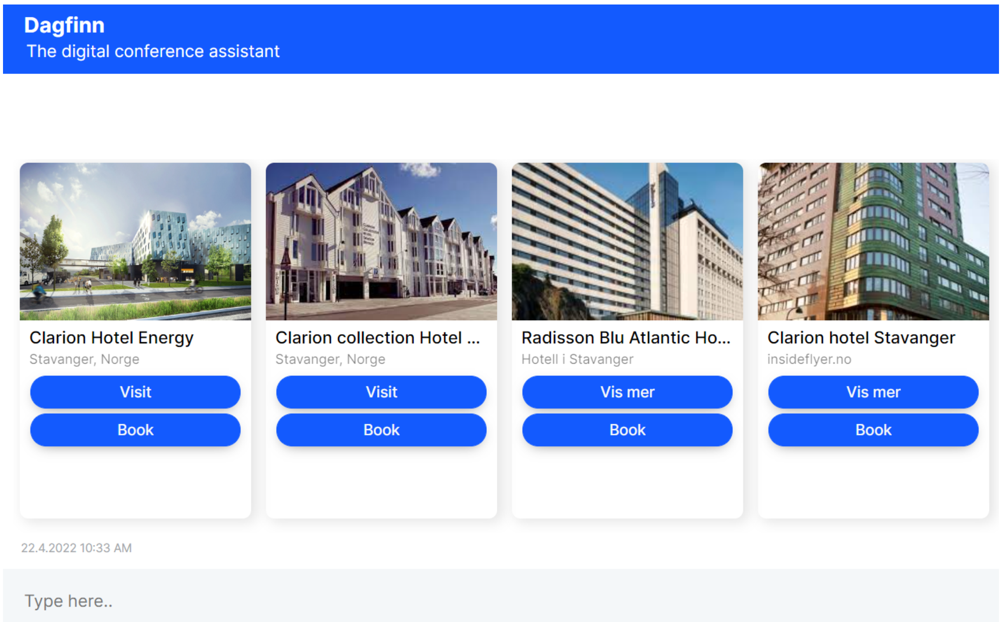

Figure 2.10: Listing four options for the user to see and reference

Part of the point of having a robot to talk to is to have a natural conversation and thus use natural language when speaking. In such a scenario, buttons might limit natural language. However, it can help the conversational system clearly understand what the user means. This is a trade-off between correct information transferred and natural language.

When combining modalities, such as text and voice, there are things to consider. The following sections discuss a few of those points.

Audio and text

Jannach et. al. [14] also suggest that simply translating a text from a visual to a speech setting is not desirable. This makes it essential for dialogue writers to test their conversations in a spoken environment, should their dialogues be used in these scenarios.

Text is written to be read, often different from text written to be spoken. So there is a need to generate text-to-speech from the ground up with a voice in mind. Examples would be abbreviations, lists, or other constructs meant for text and reading.

Screen size

As screens can vary in size, the dialogue creators need to know what information can be displayed and comprehended easily by the user. Too many options or too much information for each option on a small screen isn't ideal.

Screen size and images

A large screen can often accommodate pictures, graphs, and maps to enhance responses without disturbing the text. On the other hand, smaller screens would require links or too small images halting the natural flow of the conversation and leading to the conclusion that smaller screens should have limited extra images.

Screen size and text length

Long texts are appropriate for larger screens, but for smaller screens like a mobile screen, the text should be shorter not to overwhelm. Jannach et. al [14] also suggest users prefer more extended, more informative summaries in a text setting and that this is not observed for audio summaries.

Audio and text length

An audio response needs to be short to avoid monologues. A natural and enjoyable conversation is a back-and-forth endeavor. It is also easier to "remember" long texts as one can reference them as you go. This is not the case with audio. So if a user asks for restaurant options, a voice reply should limit the options to less than three options, often enquiring the user to narrow down the choice [14].

Chapter 3

Approach

Overview

- Initial requirements
- Architecture
- Message protocol

This Chapter is concerned with the description of how we developed the back-end system with a connection with two front-ends, and a message protocol. First, we look at the initial requirements of the project. Then we discuss step-by-step each component of the system. Finally, it ends with a description of the message protocol elements.

Appendix A describes how to run the system. While Appendix B is the full message protocol.

Initial requirements

The following are the initial requirements set by the thesis description.

This thesis is concerned with developing the back-end dialogue management system as well as front-ends for two modalities: text (as a web widget and/or Google Chrome plugin) and audio (Furhat robot). The back-end will need to be connected to an open-domain question answering system that has been developed internally by our research group.

While working on this thesis, this has been the guideline but not a step-by-step recipe.

3.1 Architecture

The architecture of the project, shown in Figure 3.1 illustrates the connection of the components of the system with Socket.IO.

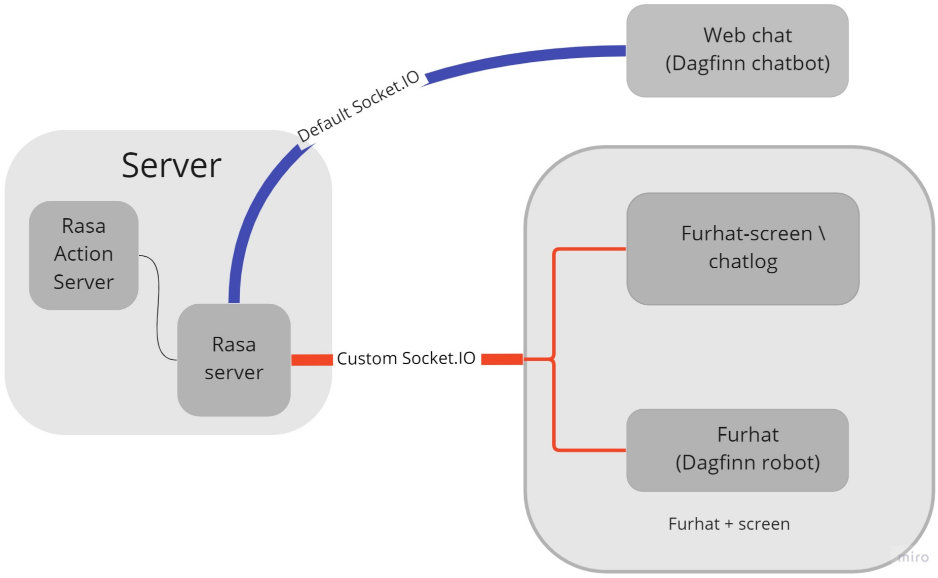

Figure 3.1: Architecture of Server and connection to the modalities.

Components

- Rasa server
- Rasa Action server
- Default Socket.IO connection
- Custom Socket.IO connection
- Webchat
- Furhat-screen
- Furhat

Figure 3.1 shows the user interfaces, Furhat + screen, and the Rasa web chat. These are all connected to the Rasa server and use Socket.IO as API. Figure 3.5 shows the custom connector and Socket.IO connector.

The rationale behind this architecture is that Rasa is the core of this system. And all information flows to and from Rasa.

3.1.1 Rasa servers

Two Rasa servers run in the DAGFiNN project. A Rasa server with a trained model and an action server that serves custom actions.

A Rasa action server returns responses and events when it is finished running a custom action. This action could be anything from finding a restaurant or asking about the weather dynamically.

To pass messages to and from Rasa, we need a proper API.

3.1.2 Details on choosing API

Before deciding to go for Socket.IO, we looked into other APIs and created a Table 3.1to show the pros and cons. Out of the options in the table, it is Socket.IO that has broadcast and is event-driven.

	Pro	Con
Socket.IO	Has broadcast Supports full duplex over a TCP channel. Supports real-time data. Event-driven technology. Used by Rasa.	Might be more difficult to implement. Is not meant to use as background service for mobile platforms.
WebSocket	Supports full duplex over a TCP channel Supports real-time data transfer Event-driven technology	Does not support broadcast.
RESTful API	Make use of standards, such as HTTP, URI and JSON Based on HTTP protocol. Not tied to any specific client-side technology.	NOT supporting real-time data

Table 3.1: Overview of Socket.IO, RESTful API and Websocket

Socket.IO, WebSocket and RESTful API

The communication between Rasa and the interfaces is considered event-driven. In this case, an event is an uttering by a user or response from Rasa. These should go through an open channel without waiting to be pulled.

WebSocket and Socket.IO share the common technology of having interactive communication between a client and a server [15].

Socket.IO is a library that enables real-time, bidirectional, and event-based communication. One of the main advantages is sending messages to a server and receiving an event-driven response without pulling the server for a reply [16].

Socket.IO provides additional features like automatic re-connection and broadcasting to all clients or a subset of clients [15]. The broadcasting feature is essential for communication between multiple modalities.

RESTful API was one option that we discarded in favor of Socket.IO. The advantages of Socket.IO and the conversational nature of the communication between clients and servers were decisive.

3.1.3 User interfaces

This project has two user interfaces. One is a webchat where all conversation is in text and pictures. In addition, this webchat has interactivity, where the user can press buttons if applicable. The other is the Furhat robot with a screen next to it. The screen displays text and pictures, if needed, to enhance anything the Furhat says.

Webchat

For this project, we choose the Rasa webchat. This is because we want a webchat that is free of charge as open-source code and is well documented. Rasa webchat ticks all of these boxes. Figure 3.2 shows an example of Rasa webchat.

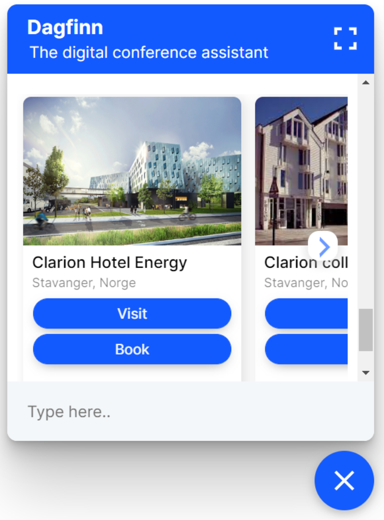

Figure 3.2: DAGFiNN webchat with carousel

One feature of Rasa webchat is the chat carousel. This feature can display one or several cards with information, images, and buttons shown in Figure 3.2.

Chat carousel

An example of how to format a chat carousel is shown in 3.3.

```
message = {
        "type": "template",
        "payload": {
            "template_type": "generic",
            "elements": [
                {
                    "title": "Clarion Hotel Energy",
                    "subtitle": "Stavanger, Norge",
                    "image_url": "https://www.anImageGoesHere",
                    "buttons": [
                        {
                            "title": "Visit",
                            "url": "https://www.anExampleHotel.com",
                            "type": "web_url",
                        },
                        {
                            "title": "like",
                            "payload": "liked suggestion",
                            "type": "postback",
                        },
                    ],
                },
            },
        }
```

Figure 3.3: Example code for chat carousel

This is implemented as part of a Rasa action. What is possible here is to have multiple options that form a carousel. The 'Elements' is an array that with the information given by the NLG, which could be images, buttons, lists, or URL links.

Chat software for Furhat screen

The Furhat-chat is a component that is connected with Socket.IO to the Rasa server shown in Figure 3.1. Combined with the Furhat robot, this is the main user interface.

The purpose of the Furhat-chat is to show the interaction log and display images and other information such as maps and QR codes. Therefore, we had to decide between creating a custom web page or using the Rasa webchat already working.

Since Rasa webchat uses React and we have no experience with that, we chose a custom solution. This makes it easier to do the customization that we want to do. However, we drew inspiration from the Rasa webchat source code in making our chat software. This solution needed styling. Thanks to Tølløv Aresvik in the POI team, who did that work.

Figure 3.4 shows an example conversation where the conversational assistant also gives information with a map and QR code.

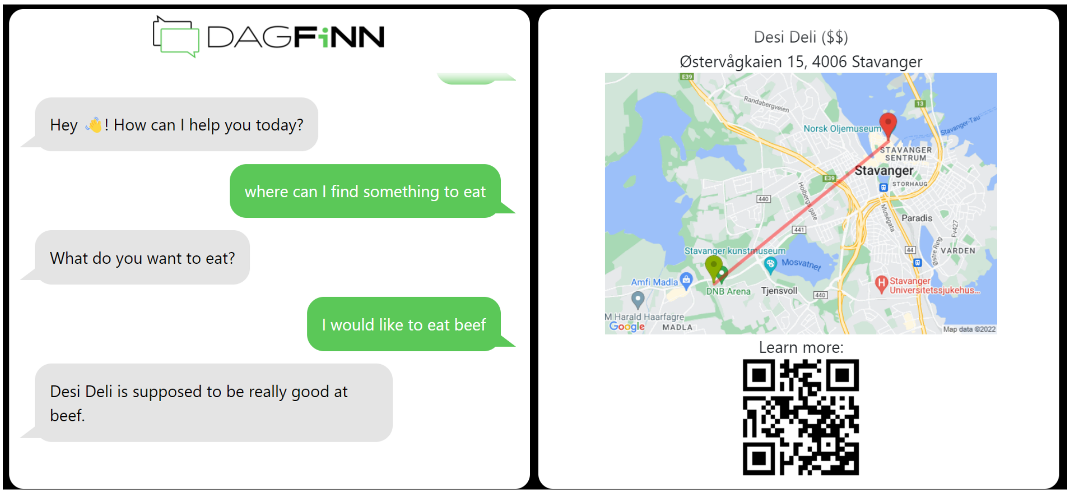

Figure 3.4: Custom furhat-chat for the furhat-screen

Consistency between the webchat and the Furhat-chat was a goal for us. So, any response given by Rasa would be displayed in the same way on both interfaces. A custom channel can be specified if the messages need to be different on either interface. A custom channel is defined by creating a custom connector.

3.1.4 Custom connector

The custom connector has two purposes in this project; one is to create a custom channel, and the other is to get messages broadcast to the Furhat robot and Furhat-chat. We did that by customizing the Socket.IO connector provided by Rasa. This modified version of the default Socket.IO connector is what makes it possible to broadcast the responses from the Rasa server.

The Rasa webchat uses Socket.IO, which is a channel in the initial Rasa setup, see line 7 in Figure 3.5.

```
addons.customconnector.Custom_socketIOInput:
    user_message_evt: user_uttered
    bot_message_evt: bot_uttered
    user_recognition_evt: user_recognition
    session_persistence: false

socketio:
    user_message_evt: user_uttered
    bot_uttered_evt: bot_uttered
    session_persistence: false
```

Figure 3.5: Custom connector and Socket.IO connector in the credentials.yml file. Both are a Socket.IO implementation

With a custom connector, we can direct messages to specific channels.

Using the custom channel

We can send different messages to different user interfaces, which gives flexibility. For instance, we can send one message to only the Furhat robot and another to the webchat.

The custom channel is only used where the NLG (Natural Language Generation) has set the response sent by the custom connector. For example, figure 3.6 shows a greeting message with the default Socket.IO channel and a greeting using the Furhat custom channel.

```
responses:
    utter_greet:
        - text: Hi! I am a greeting message using Default Socket.IO.
        - text: Hi! I am the Custom Socket.IO channel.
        channel: furhat
```

Figure 3.6: NLG using default Socket.IO connection and a custom channel.

Events need to be handled on the Furhat, but we need to look at how the Furhat interface works.

3.1.5 A well behaving Furhat

The Furhat needs to know when a user is talking to it. This is mainly done via a camera situated in front of detecting users. If no user is detected, the Furhat is idle and is not listening or moving. However, when a user enters its field of vision, it will go active and start a cycle of listening and speaking. This cycle of idle or active listening or speaking is referred to as 'Flow.' When the Furhat detects a user's speech, it will send an event to the Rasa server.

Handling events

The listening event triggers the Furhat to listen for 5 seconds. Then, if it picks up sound, it sends an event to Rasa for Rasa to generate a reply. Figure 3.7 shows a snippet of the code producing this event.

```
onResponse{
    socketConnection.emitToRasa(it.text)
}
```

Figure 3.7: Sending an event to Rasa

The Furhat also needs to handle the events received from Rasa. This is code written in Kotlin. Every time an event reaches the Furhat 3.8, it checks whether or not it has text. Then it will trigger a speak and listen to cycle, e.g., the Furhat first speaks, then it listens. The Furhat can also emote, and the system will also check for and trigger emotes events. See Facial Expressions 3.2.2

```
if (text != ""){
    EventSystem.send(Response_event(text))
    }
    ...
onEvent<Response_event>{
    furhat.say(it.utter)
    furhat.listen()
}
```

Figure 3.8: Eventsystem in Kotlin running on Furhat

We can better describe the message protocol with all components in working order.

3.2 Message protocol

The message protocol describes how to format messages for the system. Much of this message protocol is based on how Rasa defines its messaging, but further considerations are needed. The message protocol in its entirety is listed in Appendix B. Here we will look at some of the components of the message protocol in detail.

3.2.1 Emojis

Rasa can handle emojis. With the Furhat custom channel, we can send events with and without emojis. On the other hand, we want to have emojis in the Rasa webchat because it is natural to write and receive messages with emojis.

On the Furhat robot, we want to avoid emojis, as they come as text. So, for example, the Furhat will speak a waving hand emoji as "waving hand."

This issue was handled with Regex (Regular expressions) on the Furhat side by only allowing alphanumerical characters, effectively removing any emoji from the text.

3.2.2 Facial expressions

Facial expressions are an essential part of a conversation. So since we have a physical robot present, it can and should show emotions and head movement. We can see in 3.9 that the Furhat can emote.

Figure 3.9: Virtual Furhat expressing himself

Sending custom data as shown in Figure 3.10 was the choice of action.

```
utter_goodbye:
  - text: "Bye"
    custom:
      data:
        expression:
          expression_type: "anger"
```

Figure 3.10

There are 5 different facial expressions available to use in this system. This is shown in 3.11

```
"anger" to Gestures.ExpressAnger,
"shakehead" to Gestures.Shake,
"nod" to Gestures.Nod,
"wink" to Gestures.Wink,
"surprise" to Gestures.Surprise
```

Figure 3.11

The message format for expressions is structured to facilitate customization of the facial expressions if desired further down the road.

```
expression:
  expression_type: "anger"
  expression_length: 10
  // ... other parameters here
```

Adding more parameters such as expression length is an option. However, all extra parameters would have to be handled on the Furhat as it only takes expression types.

3.2.3 Maps

Lastly, we configured a way to serve maps with Google maps dynamically. This was not ready for the conference. And currently not part of the code repository. Google map is only handled on the Furhat-screen, e.g not webchat. It needs minimum latitude and longitude, but zoom and map type can also be set. The map automatically sets a marker at the location of the coordinates. Figure 3.12 shows a code snippet for how this works in Rasa.

```
utter_show_map:
  - text: "Look at this awesome map"
    custom:
      data:
        googlemap:
          latitude: 52.2456
          longtitude: 65.546
          zoom: 17
          maptype: 'roadmap'
```

Figure 3.12: Formatting response to trigger a dynamically served map

Chapter 4

Results

Since this project concerns the ECIR conference, we want to look at the results from the Furhat's test at the venue. The results are divided into two sections. Quantitative, where we look at statistics, and qualitative, where project participants discuss how the Furhat worked in the conference setting, presented as qualitative results.

We also did some in-house testing to verify the system's behaviour.

4.1 Quantitative Results

Results from the ECIR conference give us some numbers and figures. Our main interest here is to find data on things that worked. There were a few issues with the system at the conference that affected the results. These are mainly left out.

4.1.1 Filtering results

We filter out log results that to us didn't make sense. These are conversations that were empty according to the logs, or conversations that consisted almost entirely of *"Hello"* and *"Hi, How can I help you today?"* In addition to this very short conversations and very long conversations are dropped. The numbers for this is seen in 4.1 The filtering process was done using a Python-script.

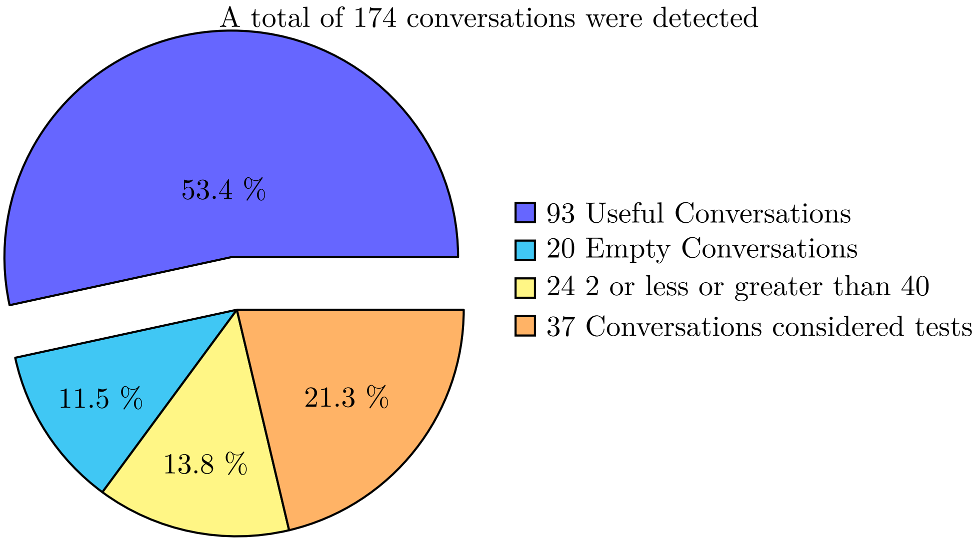

Figure 4.1: Approximately 53% of the conversations started with the Furhat are considered useful conversations.

Some conversations were started that had no content. This may be due to logging errors. These issues are not a problem for the user experience, but should be looked into to clean up logs. Might be indication of a bug somewhere else.

4.1.2 Useful replies

In the 458 replies that took place in the 93 useful conversations, 58 replies from the Furhat were out of scope replies and 11 replies were NLU fallback replies.

The system replied a total 458 replies.

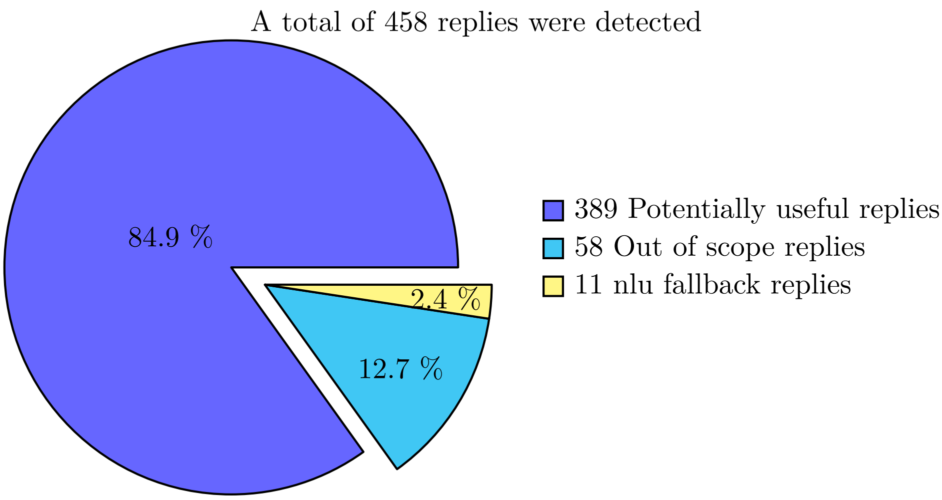

Figure 4.2: Approximately 85% of the replies from the Furhat were replies the system thought were good.

To really know if the reply is really good or not, the team would have to manually go through the 93 conversations to check if the reply really is a correct reply. As reported in the qualitative results, many users try to chit-chat with the Furhat, leading to many *out of scope* replies.

4.1.3 Length of conversations

The length of a conversation is defined as an utterance. Meaning one question with one reply is equal to a conversation length of 2. We see in Figure 4.3 that of conversations with length 10, there were 5. Or 15 conversations were of length 4.

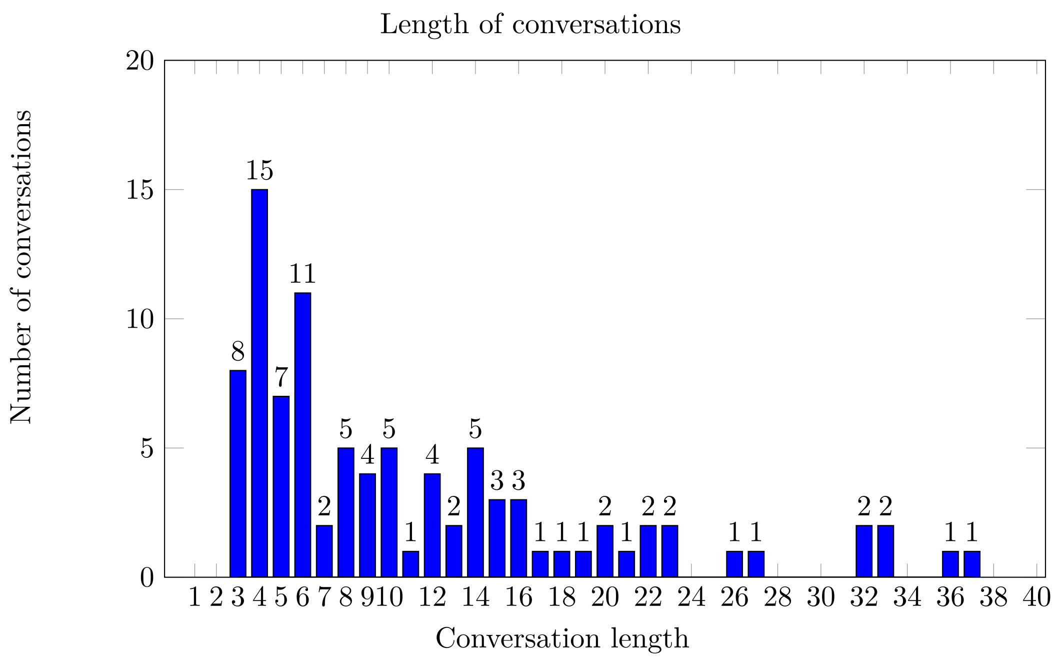

Figure 4.3: Length of conversations

Most conversation are short and lower than 10 utterances.

4.1.4 Response to intents

The Furhat can reply with 42 different utterances. There are also variations within an utterance, such as recommendations for different restaurant is considered the same utterance here. Figure 4.4 shows the top 17 utterances triggered. The 25 other possible utterances were triggered less than 4 times each and are not shown.

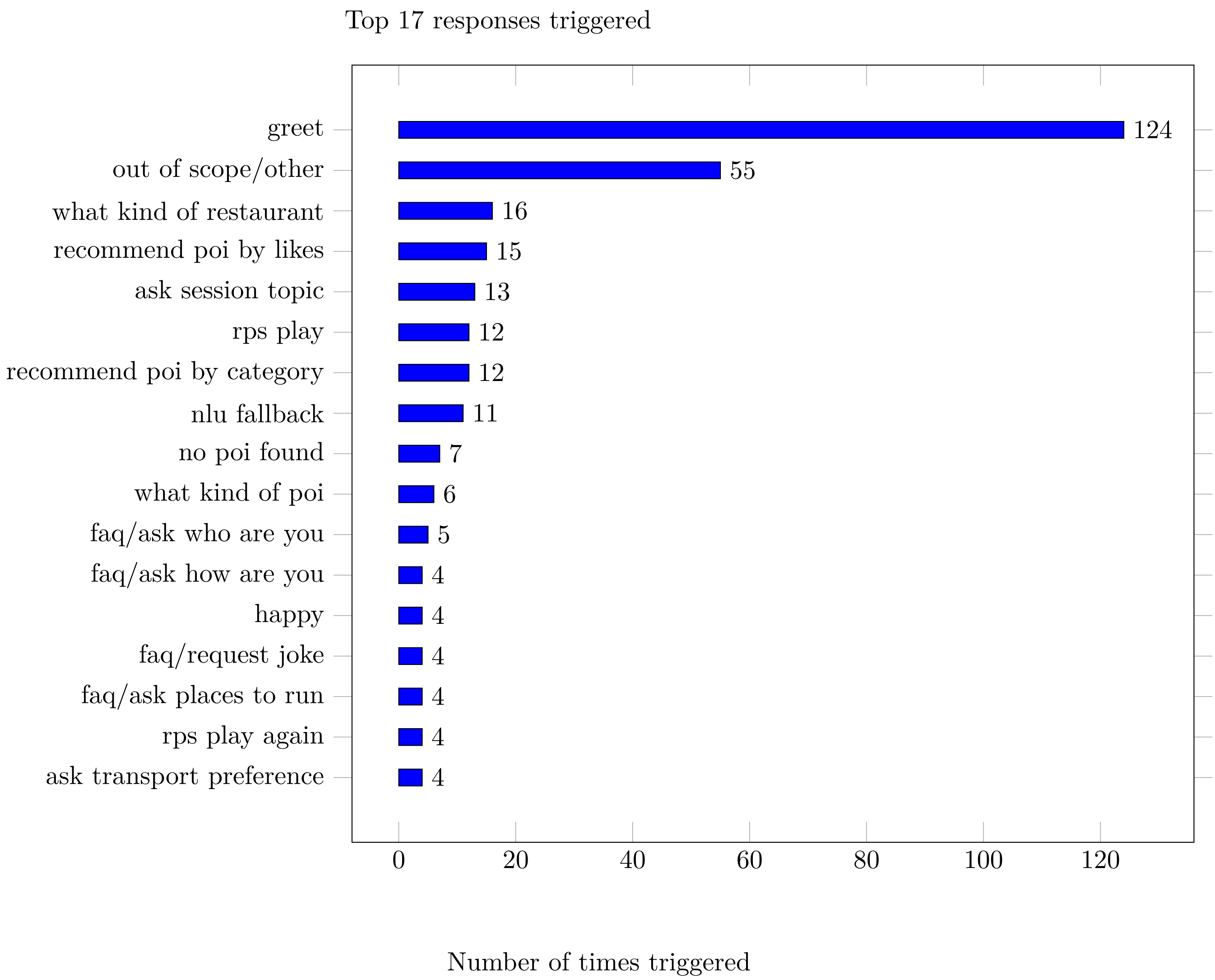

Figure 4.4: Showing the top 17, out of 42, responses. The remaining responses were triggered less than 4 times each

With the numbers and graphs in mind, we look at the project participants' thoughts about how the Furhat worked in the conference setting. These thoughts are casual and anecdotal.

4.2 Qualitative results

When we did the response testing, we had all the questions prepared and were expecting the Furhat to be able to answer all or at least most questions. This is different from a user who has no idea what to expect from the robot or knows how it works.

Since we've worked with the robot, we know how to speak with it. However, that's not necessarily conversational. This is different from conference users.

The overall sentiment is that the dialogue needs more chit-chat to be more conversational. In addition, it should guide the conversation toward subjects the Furhat robot can answer and what the robot should answer in the context of where it is located. In our case, that would be things related to the town, Stavanger, and the conference.

There seems to have been not enough content and information about the conference. It was also suggested that the conference material didn't have enough rich responses, responses with images, etc. Though this is reported to be a communication error, it was implemented; only the conference team knew how to invoke these responses.

At the conference, users were reported to be happy and satisfied with the interaction with the robot. Almost exclusively, these were users that were guided on how to speak with the robot. On the other hand, users that didn't get instructions were largely unhappy with the conversations.

Many users start the conversation by trying to get to know the Furhat. But the robot doesn't have extensive domain knowledge in this field. This falls under the category of needing more chit-chat. Answering essential questions about who you are and where you came from is the pinnacle of chit-chat.

The Furhat manages conversations by seeing if a user is nearby and in contact with the Furhat. Users report weird behavior where the Furhat will move its head toward a new user entering its field of view. Which it shouldn't. This means the Furhat did not detect the current user. This issue is regarding the 'Flow' in the Kotlin code on the Furhat robot.

Among the things suggested to be added is information about the time, weather, weather forecast, and fjords. Some of these topics were added at the conference.

The robot was easy to set up. No issues with that.

Users had to talk a bit too loud for the comfort of the robot. This might feel unnatural to the user. The reasons could be people's accents and noise levels in the conference area. This might result in people articulating their words stiltedly and reverting to "baby-talk." Meaning using very few words and only keywords. This works counter-productive to NLU, which works better with longer sentences and more terms. Users also reported having to repeat utterances.

Meta feedback was that conference participants didn't have time to talk to the robot. Other users seemed more enthusiastic about the robot, especially kids. Few people seemed interested in the robot.

The Furhat interrupted users. This is part of the cycle of speech. This is due to its five-second time to wait to hear a user uttering. It is possible to handle this on the *onNoResponse* event in the Kotlin code. There, an option is to make the robot do something else when it is not hearing anything within the given time period.

The Furhat is often viewed as creepy. The sudden head movement, is having a negative effect.

4.3 Methodology of in-house testing

We tested Furhat's capabilities by going through questions provided by the POI- and Conference-team. We experiment with physical Furhat with a screen next to it. When testing, we look for proper responses to the specific questions provided.

4.4 Furhat responses in testing phase

The testing results will show the expected result and actual result, as well as what the user intends to say and what the robot picks up. These are the test conversations considered unique enough to be presented here.

```
Intended Question  |  What is there to see in Stavanger?
Furhat interpreted |  what is there to see in calendar
Furhat Reply       |  Do you have anything specific in mind?
Expected Reply     |  (Talk about what to do in Stavanger)
```

We know that "Do you have anything specific in mind?" is part of food stories and thus not an appropriate answer here. A better response would suggest alternatives since a user might be after things that Stavanger is famous for, like historical locations or local attractions.

```
Intended Question  | recommend me some bars?
Furhat interpreted | recommend me some bars
Furhat Reply       | Pjolter and Punsj is an excellent bar.
```

Figure 4.5

This result is as expected, with a map and QR code where a user can find more info about this bar.

```
Intended Question  | Who are the keynote speakers of the conference?
Furhat interpreted | who are the keynote speakers of the conference
Furhat Reply       |
    These are the titles of the keynotes: Keynote 1 Accountable and
  Robust Automatic Fact Checking, Keynote 2 Empirical evaluation of
  predictive models: A matter of scales and means, keynote 3 Towards
  Language Technology for a Truly Multilingual World
```

Information about the conference, listing three items, but no text or listing on the Furhat-screen. This should be presented as a list to read the given information better.

```
Intended Question  | What will tomorrow\'s keynote be about?
Furhat interpreted | keynote be about
Furhat Reply       | Perhaps you could repeat that?
Expected Reply     | (Give information about the conference)
```

Sometimes the Furhat isn't listening, and the user starts to speak before it has entered the listening mode. The result here shows this case. The Furhat can only understand an utterance when it is attending to the user.

```
Intended Question  | Are there any sessions going on right now?
Furhat interpreted | are there any sessions going on right now
Furhat Reply       | What topics are you interested in?
Furhat interpreted | I like a AI
Furhat Reply       | I do not understand what you are saying.
Expected Reply     | (Ask to narrow down topic, or give user info about
   an AI session)
```

This result here is that the NLU, in the end, does not understand the user utterance and is then classified as an NLU fallback. Again, this is typical of not having enough examples in training.

```
Intended Question  | When is the lunch break?
Furhat interpreted | when is the lunch break
Furhat Reply       | I can't help you with that, I'm sorry.
```

The response given by the NLG is from out-of-scope utterances. Therefore, the word "break" is classified by the NLU to that domain.

We have some concluding thoughts on the conference results, and in-house testing presented.

Chapter 5

Conclusions

We divide our conclusion into three sections. First, an analysis of the work done. Second, an analysis of the conference results, and third, concluding with future directions.

5.1 Analysis of work done

Our main objectives in this thesis have been to develop a connection between components and develop a message protocol.

With Socket.IO, we achieved a contact between the components. Though there were other options, for instance, RESTful API, we chose Socket.IO because of the event-driven communication and built-in features such as broadcast.

Customizing the Socket.IO channel, we enabled the possibility for messages to be specified for particular usage in NLG (natural language generation). With this channel, one can emit messages with or without certain features, like messages containing emojis.

The message protocol describes how to handle text, images, and lists accordingly with the Furhat screen and the Furhat robot. In addition, we established functionality for buttons and maps for the conference but discarded them due to limited time closing in on the deadline. Ultimately, we are not far from deploying interactivity features for future usage.

5.2 Analysis of the conference results

174 conversations were logged from the conference. After filtering undesirable data, we had 93 conversations that were deemed valid. The number of filtered conversations can be reduced if the system works better.

There could also be an improvement in the flow of the Furhat robot. Its status changes from listening to idle if a user briefly turns away from the robot. Doing that ends the conversation, and that leads to starting from scratch. It should recognize the same person it was talking to just a few seconds ago.

We could've planned information retrieval better. Letting the logging to the database be more specific about logging the amount f. ex uttered intents, length of conversations, timestamps, etc.

One of the significant issues discovered at the conference was the lack of a chit-chat domain. Instead, people talked about random things, but the Furhat could only speak about prepared things. e.g., things to do in Stavanger and conference material. In particular, a domain about the robot itself was in need.

5.3 Future Directions

People often say the robot looks almost human, but not entirely, leaving them uneasy. Improving facial linguistics can remedy this. For instance, one could make the robot look away from the user and let its eyes rest elsewhere from time to time.

We can not store information about any user without consent. This could be dealt with by having the robot in inactive mode until a user presses a consent button on the Furhat screen, which then activates it. However, this has limitations because of GDPR (general data protection regulation).

The message protocol could be expanded upon and explained in further detail. We developed a solution for handling facial expressions through Rasa.

Furhat Robotics has available several gestures and emotes ready for use. This should be looked at and made available for the story-writers.

A map solution should look into a different provider than Google Maps since Google Maps is not necessarily free.

With interactivity in mind, it's easy to create stories where the robot can ask users simple yes and no questions and have them press a button as an answer, even if that could be answered directly by the robot.

Final words

In the end, we are satisfied with the result of the ECIR conference. The contribution to the DAGFiNN project on our part has worked. We have, during the whole process shared information on discoveries we have made on the way, especially regarding the Furhat robot, which is new to the DAGFiNN project.

Appendix A

Instructions to Run System

How to run Webchat, furhat - virtual and physical - with screen

Step 1 - Setting the IP-address of the rasa-server

Step 1 for everything is to enter the correct IP for the RASA server. Normally when testing, you will use 'localhost' as your address. If you are running over a different network, go to config-file and change IP

You change IP at the top of the file in question. You change the IP in

> dagfinn\ui\furhatScreen.index.js

and in

>dagfinn\ui\furhat-skills\rasa\src\main\kotlin\furhatos\app\rasa\flow.interaction.kt

```
const IP = "10.192.201.55"     // or  "localhost"
const PORT = "5005"
```

Remember to recompile after changing IP in 'interaction.kt'

Step 2 - Starting RASA server(s)

In every case run rasa servers first. Make sure you are standing in the base project folder. And that you are in the virtual environment (dagfinn) > (dagfinn) dagfinn

Before starting the bot and after every change we need to retrain the bot. This can be done with the command:

```
rasa train
```

To run the chatbot you will need two terminals. The first one is for the *actions* server. You can start it by typing:

```
rasa run actions
```

In the second terminal you can start the Rasa server with all defined channels:

```
rasa run --cors "*"
```

Step 3 - Running virtual furhat + furhatscreen

In order to run the virtual furhat you need to have downloaded the furhat virtual launcher. Go to the following link, you need to LOGIN you access the site. > https://furhat.io/downloads > Choose the virtual launcher of your choice: Mac, linux, windows.

Start the launcher and start the SDK. Then press start skill. The skill will be located in ‘ui\furhatscreen‘ The name of the skillfile is: > rasa-all.skill

Now to virtual furhat should be running and connected to rasa.

Step 3 (alternative) - Running physical furhat + furhatscreen

Start the robot by pressing the on button, small button, not the big wheel button. In order to connect to the furhat, your laptop needs to be on the same network.

Guest network

By default the furhat is on the guest-network on uis. Go to uis.no and get access to the guest network

> https://www.uis.no/en/student/wifi-network

Make sure you are actually connected to the guest-network. Make sure that the config file points to the correct IP address. Are you running rasa on the laptop, the IP you enter in the config file is your laptop’s IP. Usually something like ‘10.168.200.5‘

Wifi Hotspot

You can connect the furhat you your laptops wifi hotspot. In order to connect to your hotspot you will need to connect a keyboard to the furhat in order to type the password to hotspot wifi.

Step 4 - Running rasa webchat frontend

Open a new terminal. Change directory to

> dagfinn\ui\webchat

or

> dagfinn\ui\furhatscreen

run the command `python -m http.server <port>` You can also omit portnumber and the server will default to port 8000 `python -m http.server` or `python -m http.server 9000` in order to run the server on a different port. this can be necessary if you try to run multiple web servers.

Making changes to the skill | kotlin code | compiling

go to folder

>ui\furhat-skills\rasa\

run

```
gradlew shadowJar
```

the furhat skill will be under >ui\furhat-skills\rasa\build\libs

and is named 'rasa-all.skill'

If you make any changes to the kotlin code, you have to recompile with gradlew shadowJar.

Troubleshooting

Always Press Stop SDK, to make sure the SDK stops gracefully. View Console at the bottom of the screen seen in Figure A.1 to view the output from the console.

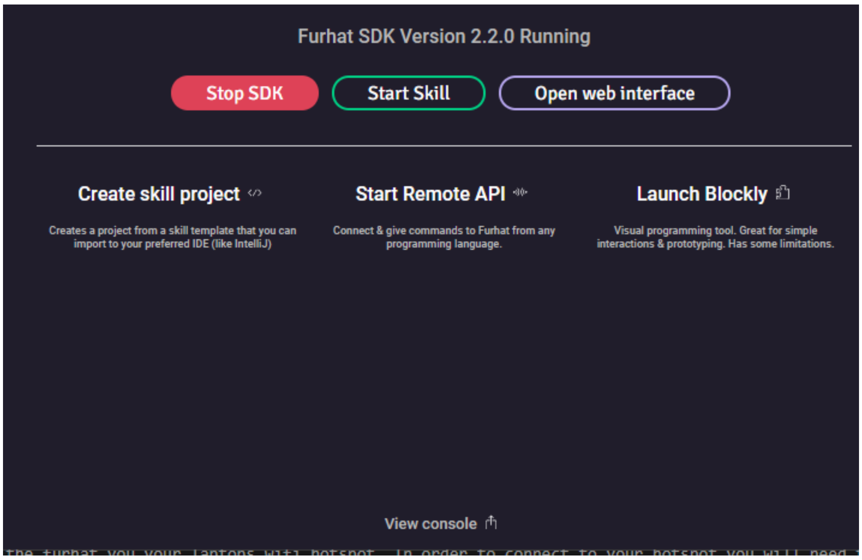

Figure A.1: Virtual Furhat launcher

The Furhat can't find the Rasa server, or is otherwise unable to reach the network, seen in Figure A.2

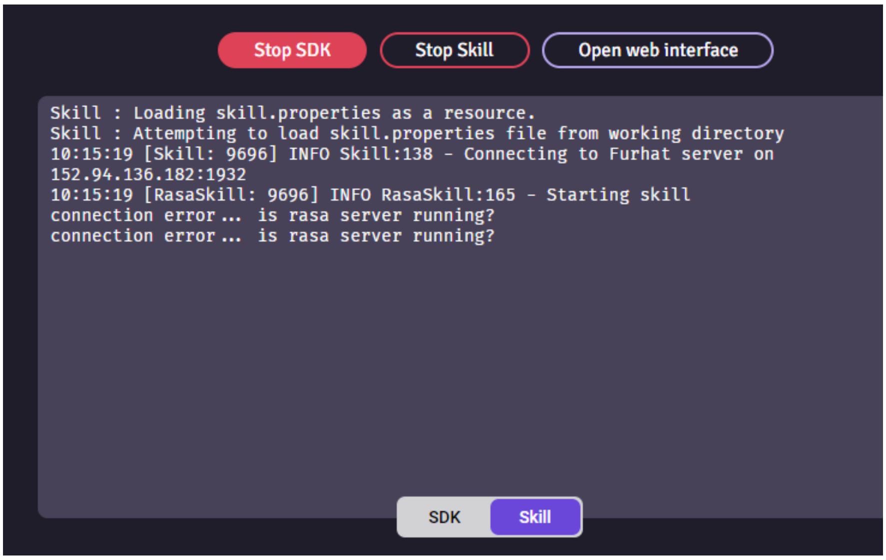

Figure A.2: Error showing no connection to the Rasa server

Appendix B

Message protocol

> Focus on making it work in webchat

Webchat

In general

```
responses:
  utter_greet:
  - text: "Hey, {name}. How are you?"
```

{name}can be set programatically or can be taken from a slot.

```
dispatcher.utter_message(
    template="utter_greet",
    name="Sara"
)
```

Conditionals

To specify responses to only be sent to the Furhat, and not webchat you can apply a conditional:

‘channel: Furhat‘

```
- text: Link to [Uis Homepage](https://uis.no)
- text: Visit UIS no
  channel: Furhat
```

If no ‘channel: Furhat‘ is present, responses will still be sent to Furhat to be uttered. So normally you don’t use this. Only use this if you need to distinguish between webchat and Furhat / spoken.

Links

```
text: Link to [Uis Homepage](https://uis.no)
```

Issue: Since this is text, Furhat will try to speak out the link.

To solve this you can use conditional statements:

```
- text: Link to [Uis Homepage](https://uis.no)
- text: Visit UIS no
  channel: Furhat
```

This means the Visit UIS no will be spoken by the Furhat, and appear at the Furhatscreen, while the link will appear in the webchat.

Emojis

Most Emojis work everywhere that can display text. Check out this website to copy-paste some emojis.

* https://emojipedia.org/

Images

```
utter_cheer_up:
 - text: "Here is something to cheer you up:"
   image: "https://i.imgur.com/nGF1K8f.jpg"
```

Carousel

A carousel is a custom action that needs to be modified to your specific needs. There is a action_quiz carousel basic template.

So you will need to copy the template, register it as a new action, and fill in the response as you need. This example returns the name ‘action_quiz‘. Rename your new file, your class, and return name. The name returned, is the name you register in ‘domain.yml‘

```
class ActionQuiz(Action):
    def name(self) -> Text:
        return "action_quiz"
```

You should now be able to call this action, to serve a list. That works both for webchat and the Furhatscreen.

The carousel will behave differently on Furhatscreen and the webchat. No interactivity on the Furhatscreen, so no buttons.

Furhatscreen

`![Frontend_quiz](assets/frontend_quiz.png)`

Webchat `![Webchat_quiz](assets/webchat_quiz.png)`

Buttons

```
utter_greet:
  - text: "Hey! How are you?"
    buttons:
    - title: "Super happy"
      payload: "mood_great"
    - title: "super sad"
      payload: "mood_unhappy"
```

Both carousel and buttons examples use intents as payloads. See https://rasa.com/docs/rasa/responses/# for info on other payloads such as /inform

As with the carousel, no buttons are shown on the Furhatscreen.

Maps

If maps are needed it should simply be an image. Any markers would be put on the map before saving the image.

Furhat

Emojis

If you use emojis in responses, the Furhat should only allow alphanumerical values, e.g scrub any emojis from the text, and not try to pronounce emoji-names.

Facial expressions

Make the Furhat show emotions. Currently the Furhat will raise eyebrows and smile as a default response.

Adding a custom data to the response with the following syntax will make the Furhat show facial expressions.

```
utter_goodbye:
  - text: "Bye"
    custom:
      data:
        expression:
          expression_type: "anger"
```

These 5 facial expressions are currently supported.

- "anger" to Gestures.ExpressAnger,
- "shakehead" to Gestures.Shake,
- "nod" to Gestures.Nod,
- "wink" to Gestures.Wink,
- "surprise" to Gestures.Surprise

Bibliography

[1] Krisztian Balog. Conversational ai from an information retrieval perspective: Remaining challenges and a case for user simulation. In *2nd International Conference on Design of Experimental Search & Information REtrieval Systems*, DESIRES 2021, pages 1–11, 2021.

[2] James. H. Martin Jurafsky Daniel. *Speech and Language Processing*, volume 3. Stanford University, 2021.

[3] Jan Deriu, Alvaro Rodrigo, Arantxa Otegi, Guillermo Echegoyen, Sophie Rosset, Eneko Agirre, and Mark Cieliebak. Survey on evaluation methods for dialogue systems. *Artificial Intelligence Review*, 54(1):755–810, 2021.

[4] Jianfeng Gao, Michel Galley, Lihong Li, et al. Neural approaches to conversational ai. *Foundations and trends® in information retrieval*, 13(2-3):127–298, 2019.

[5] Rasa Technologies Inc. Rasa documentation, 2022. URL https://rasa.com/docs/rasa/. Accessed: 2022-01-25.

[6] Pararth Shah, Dilek Hakkani-Tur, and Larry Heck. Interactive reinforcement learning for task-oriented dialogue management. 2016.

[7] Tobias Wochinger. Rasa nlu in depth: Part 1 – intent classification, Feb 2019. URL https://rasa.com/blog/rasa-nlu-in-depth-part-1-intent-classification/v.

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[9] Susanna Dillenbeck Chris Wood. What are social robots? an introduction to the furhat robot, 2022. URL https://furhatrobotics.com/blog/what-are-social-robots/. Accessed: 2022-02-06.

[10] Furhatrobotics why-do-we-need-robots-arent-chatbots-and-avatars-enough, 2022. URL https://furhatrobotics.com/blog/

why-do-we-need-robots-arent-chatbots-and-avatars-enough/. Accessed: 2022-02-07.

[11] Furhatrobotics 7 best usecases, 2022. URL https://furhatrobotics.com/blog/the-7-best-use-cases-for-social-robots/. Accessed: 2022-02-07.

[12] Pepper robot promotional, 2022. URL https://www.robotlab.com/Pepper-Robot. Accessed: 2022-05-14.

[13] Furhatrobotics faq, 2022. URL https://furhatrobotics.com/resource-hub/#FAQ. Accessed: 2022-02-06.

[14] Dietmar Jannach, Ahtsham Manzoor, Wanling Cai, and Li Chen. A survey on conversational recommender systems. *CoRR*, abs/2004.00646, 2020. URL https://arxiv.org/abs/2004.00646.

[15] Automattic. Socket.io documentation, 2022. URL https://socket.io/docs/v3/how-it-works/. Accessed: 2022-01-27.

[16] MDN Web Docks. Websocket documentation, 2022. URL https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. Accessed: 2022-01-27.