Building applications with the Furhat Robot

Contents

Introduction

The Furhat Robot is a state-of the art conversational robot and versatile software platform, which can be used by anyone interested in exploring social robotic applications, whether the application domain is teaching language, performing interviews, telling stories to children, or helping passengers at the airport. The Furhat Robot is currently used to research human communication and perception, speech technology, conversational agents, but particularly in research into Human-Robot Interactions. In this white paper, we will provide an overview of the different ways in which the Furhat Robot can be programmed and how applications (or “skills”) can be developed. If you want to read more about the details, we refer you to our documentation at docs.furhat.io and our examples on GitHub.

Figure 1 shows an overview of the Furhat Platform. As can be seen, it includes an SDK with a suite of tools and API's to create interactions with the Robot. When you develop applications, you can either try them out on the physical Furhat Robot, or with a Virtual Robot that comes with the SDK.

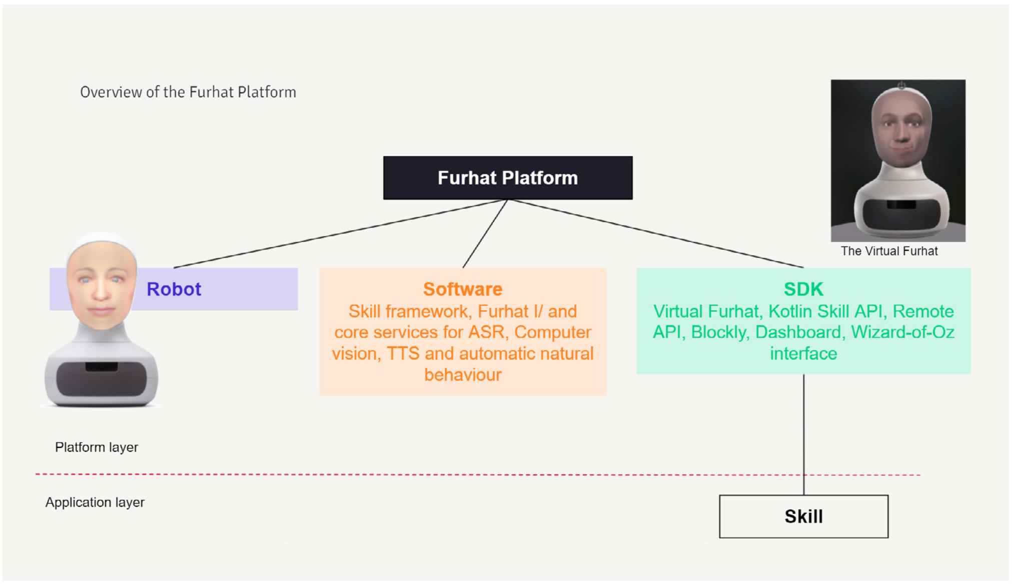

Overview of the Furhat Platform
Furhat Platform
The Virtual Furhat
Robot
Software
Skill framework, Furhat I/ and core services for ASR, Computer vision, TTS and automatic natural behaviour
SDK
Virtual Furhat, Kotlin Skill API, Remote API, Blockly, Dashboard, Wizard-of-Oz interface
Platform layer
Application layer
Skill

Figure 1: Overview of the Furhat Platform.

Since the potential use cases of the Furhat Robot are so many, we believe that the best way to accommodate developers is to provide different ways of programming the robot. Currently, we provide three ways of doing this, as illustrated in Figure 2:

- The **Remote API**, by which you can access the core Robot I/O functionality, using any programming language.
- The **Kotlin Skill API**, by which you have access to the **Skill Framework** and all functionality of the platform, in order to build rich interactions.
- **Blockly**, which is a graphical programming tool with which you can also access the Skill Framework, albeit limited to simpler interaction.

In the next sections, we will describe these three approaches in more detail.

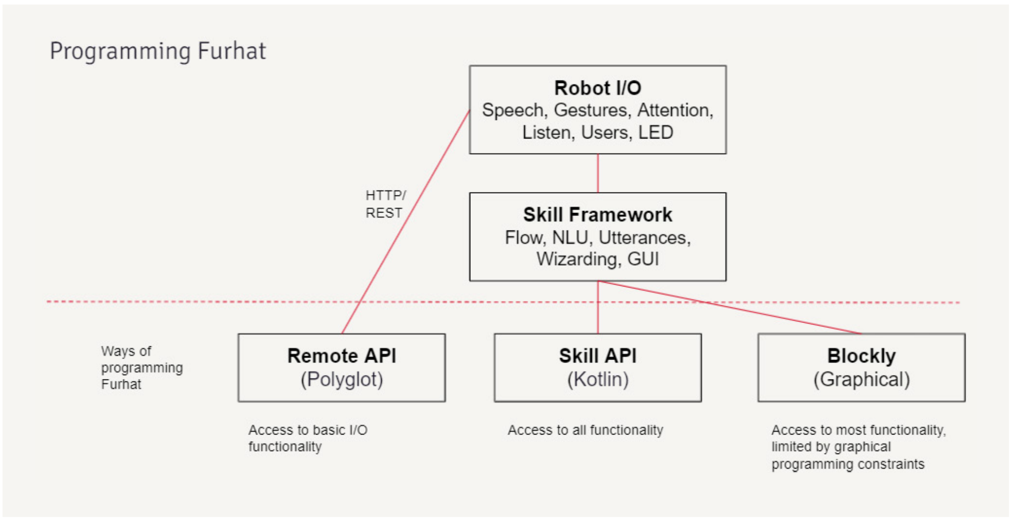

Programming Furhat
Robot I/O
Speech, Gestures, Attention, Listen, Users, LED
HTTP/ REST
Skill Framework
Flow, NLU, Utterances, Wizarding, GUI
Ways of programming Furhat
Remote API
(Polyglot)
Skill API
(Kotlin)
Blockly
(Graphical)
Access to basic I/O functionality
Access to all functionality
Access to most functionality, limited by graphical programming constraints

Figure 2: The three different ways of programming the Furhat Robot.

Robot I/O through the Remote API

The Remote API is intended for users of the Furhat Robot that are only interested in the core input/output functionality of the robot, such as speech synthesis, face tracking and speech recognition, and want to access these functionalities from the programming language of their choice. For example, let's say you already have a dialogue framework written in Python, and you want to connect it to Furhat, so that you can track users in front of Furhat, recognize speech, and perform speech synthesis with synchronized lip movements. In that case, the best approach is likely to use the Remote API. As the name suggests, this will assume that your code runs on a separate machine connected over a network to Furhat, and that your program sends remote instructions to the robot. During development, you can also connect it to the virtual Furhat running on your computer.

Remote API is polyglot, including support for over 50 different programming languages. The Remote API implemented it as a REST API over HTTP, using the tool Swagger to generate the different client libraries. This means that you can generate an API for your language of choice, be it C#, Python, Scala or Bash.

Since Python is such a popular programming language (especially in the research community), we have made it even easier to use the Remote API from Python, by putting an already compiled client library on PyPI (called "furhat-remote-api"). The following examples show how easily the library can be used from Python, after it has been installed:

```
from furhat_remote_api import FurhatRemoteAPI

# Create an instance of the FurhatRemoteAPI class, providing the address of the
robot or the SDK running the virtual robot
furhat = FurhatRemoteAPI("localhost")

# Set the voice of the robot
furhat.set_voice(name='Matthew')

# Say "Hi there!"
furhat.say(text="Hi there!")

# Listen to user speech and return ASR result
result = furhat.listen()
```

The core I/O functionality that the Remote API supports includes:

- **Speech synthesis:** We use the Amazon Polly TTS and Acapela TTS, with support for 210 voices and 43 languages. You can send a text to synthesize to Furhat, and synchronized lip movements will be added automatically. By using SSML tags, you can modify speaking rate, stress, etc. You can also send a pointer to an audio file to play. If it contains speech, lip movements will also be added.

- **Speech recognition:** Furhat can listen to speech and return a text with the recognized speech using its built-in microphone. We use cloud-based services from Google Speech and Microsoft Azure, with support for 40+ languages.

- **Facial gestures:** Furhat has a library of 20 pre-defined gestures (such as “smile” and “brow raise”) that you can activate. If you want, you can also define your own gestures using a set of 52 low level parameters.

- **Gaze:** You can control where Furhat should be looking by providing coordinates in space. The neck and eyes will automatically move in a natural way.

- **Face tracking:** The users in front of Furhat are tracked with the built-in camera. You can get access to the location of these users, as well as their head pose and facial expressions.

- **LED ring:** You can control the color of the LED ring under Furhat.

The Skill Framework

While the basic I/O functionality might be suitable if you already have some kind of external dialogue framework or if you only want to control the robot for a specific experiment, it does not have any built-in support for building complete social robotic applications, what we refer to as “Skills”. To support this, we provide the Skill Framework and the Skill API. As illustrated in Figure 2, the Skill Framework builds on top of the basic I/O functionality described above and offers more complex functionality and behaviors. This includes **Natural Language Understanding (NLU)**, **Dialog management**, **Multimodal utterances** (mixing synchronized speech, gestures and other behaviors), support for **Wizard-of-Oz** (i.e., semi-autonomous control), **Logging interactions**, and **Graphical User Interfaces (GUI)** (to complement the conversational interaction). You access the Skill framework through the powerful **Kotlin Skill API** or using the more simplistic graphical programming interface called **Blockly**.

Kotlin Skill API

You access the Skill Framework through the Skill API, which is implemented in the Kotlin programming language. Kotlin is a modern language, using the Java Runtime Environment, which means that you can use all existing Java libraries directly from Kotlin. One of the strengths of Kotlin is that it is statically typed, which means that you will get very good code completion when developing in an IDE, to explore all the different methods provided, get documentation, and verify that you use the API correctly. We recommend using the IntelliJ IDE when developing your skill, which has native support for Kotlin. You can run and debug your skill directly from the IDE, either towards a Furhat Robot, or Virtual Furhat running on your computer. When you want to deploy your skill, you can package it as a single skill-file and upload it to your robot, as well as distributing it to others if you want. This way, the robot can run your skill completely stand-alone without any additional computer connected.

Note that you can also use the Skill API even if you are only interested in the basic Robot I/O functionality. For example, if you have built an application in Google DialogFlow and want to integrate it with the Furhat Robot, it is very easy to connect your skill to a Google DialogFlow agent, and make the Furhat Robot ask questions and read out the responses.

Dialog framework

Central to the Skill Framework is the Flow, which is a programming framework for managing the interaction. Basically, you can say that the Flow defines how the robot should react to various events (such as sensory input), depending on which state it is in. In dialog systems, this is often referred to as *dialog management*, but since a human-robot interaction is multimodal, it does not only handle the verbal input/ output, but also users entering and leaving, people shifting attention and making facial gestures, etc.

The Flow is based on the concept of hierarchical state machines. The idea is that the interaction transitions through states, and in each state you can define how the robot should react to events, through triggers. There is a large set of built-in triggers for various types of events, such as the user saying something, a user entering or leaving the interaction, the user shifting attention, or if a certain timeout has passed. Through Kotlin's support for extension functions, you can also define your own triggers, and your own complex behaviors.

The states are defined in a hierarchical fashion, which means that you can define general behaviors (such as what should happen if a new user enters the interaction) in a parent state, and then more specific behaviors in the leaf states.

One of the strengths with Kotlin is its support for DSL:s (Domain Specific Languages), which allows you to write well structured code in a declarative way. The Flow is an example of this, and it allows you to easily define Furhat's behavior on a high level of abstraction. Here is a very simple example of how two states within a Flow can be defined:

```
val OrderFruitState = state {

   onEntry {
       furhat.ask("Would you like to buy some fruits?")
   }

   onResponse<BuyFruit> {
       // Access the fruits from the intent
       val fruits = it.intent.fruits
       // Store the ordered fruits with the user
       users.current.fruits += fruits
       // Confirm the order
       furhat.say {
           +"Okay, ${fruits.text}"
           +Gestures.Smile
           +"what a lovely choice!"
       }
   }

   onResponse<No> {
       furhat.say("Ok, maybe some other time then!")
       goto(EndState)
   }

   onResponse {
       furhat.say("Sorry, I didn’t understand")
       reentry()
   }

   onNoResponse {
       reentry()
   }

}

val EndState = state {

   onEntry {
       furhat.say("Goodbye!")
       furhat.attendNobody()
   }

}
```

Natural language understanding (NLU)

As the example above shows, the special onResponse trigger is also integrated with the NLU engine. For example, the trigger *onResponse<No>* will be triggered if the user says something that can be interpreted as the “intent” *No*, such as “nope” or “don’t think so”, regardless of which of the supported languages was used. The developer can use any of the built-in intents, or define new ones that are specific to the domain. In the example above, the *BuyFruit* intent would be an example of that. The intents also have “entities” in them, which would be the fruit that is being ordered in this example. The platform also comes with a set of built-in entities, such as time and date expressions. If you have used other modern NLU frameworks, such as DialogFlow, LUIS or RASA, you will find the basic principles to be familiar. However, here the NLU is tightly integrated with the flow. This means that the potential intents that can be recognized depend on which state the dialog is currently in, and will be automatically classified. Since the intents and entities are defined programmatically in Kotlin, you can also create integrations with your own database or other backends for creating these dynamically.

Multimodal utterances

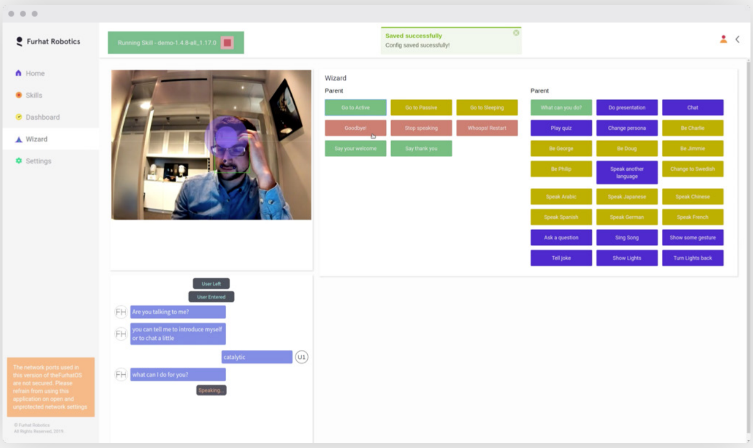
In many cases, you want to add facial expressions and other behaviour with the Robot’s speech. In the example above, a smile was inserted in the middle of the Robot’s response after the user triggered the intent *BuyFruit*. You can mix in any behaviors, including gaze shifts, or arbitrary code to be executed to form multimodal utterances.

Wizard-of-Oz

When developing human-robot interactions, or dialog systems in general, it is often hard to foresee all the things users might want to say to the robot, or how they will behave in unexpected situations. A common approach is to use a so-called Wizard-of-Oz setup, where a hidden person (the “Wizard”) controls the robot initially. The interactions can then be logged and analysed, to guide the development of the system.

Another situation where Wizard-of-Oz is useful is if you are a human-robot interaction researcher and you want to do controlled experiments on how the robot’s behaviour affects the user. Let’s say you want to investigate whether a robot that smiles more often also makes the user smile more often. You might want to set up a simple interaction where this can be tested, such as a robot interviewer which asks the user a couple of questions. This can then be done without building a fully autonomous system.

The Skill Framework has very powerful built-in support for Wizard-of-Oz. You can simply add *onButton* triggers in your flow with associated robot behaviors. These buttons will then appear in the dashboard of the web console where you can monitor and manage the Furhat Robot, as shown in Figure 3. This means that you can have different buttons appearing, depending on the current state of the dialog. And you can mix autonomous behavior with controlled behavior. As you can see in the figure, the buttons can also be organized with colors and grouped. The camera feed from the robot is shown to the left. If you click in the camera view, you can make Furhat attend a specific location or a user (and then automatically follow that user as she moves around).



Saved successfully
Config saved sucessfully!

FH: Are you talking to me?
FH: you can tell me to introduce myself or to chat a little
U1: catalytic
FH: what can I do for you?

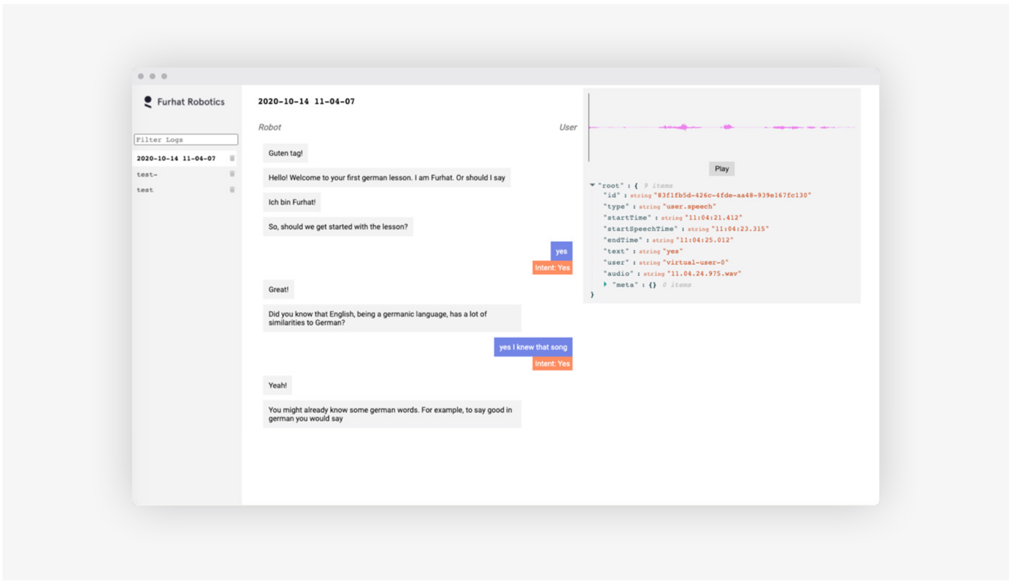
The network ports used in this version of theFurhatOS are not secured. Please refrain from using this application on open and unprotected network settings

© Furhat Robotics
All Rights Reserved, 2019.

Figure 3: The dashboard with the Wizard buttons.

Logging interactions

Regardless of whether you are doing HRI experiments, developing your skill using Wizard-of-Oz, or wanting to fine-tune your skill, it is very useful to be able to log the interactions. The SDK offers a Log Viewer tool, where you can see logs from your interactions, as shown in Figure 4. There you can see detailed timestamps of events, read the transcriptions of what has been said, and also listen to the user's speech. Note that your interactions are not automatically logged, but it is very easy to start and stop logging from anywhere within your skill.



2020-10-14 11-04-07

Guten tag!

Hello! Welcome to your first german lesson. I am Furhat. Or should I say

Ich bin Furhat!

So, should we get started with the lesson?

yes

Intent: Yes

Great!

Did you know that English, being a germanic language, has a lot of similarities to German?

yes I knew that song

Intent: Yes

Yeah!

You might already know some german words. For example, to say good in german you would say

```
▼ "root" : { 9 items
    "id" : string "83f1fb5d-426c-4fde-aa48-939e167fc130"
    "type" : string "user.speech"
    "startTime" : string "11:04:21.412"
    "startSpeechTime" : string "11:04:23.315"
    "endTime" : string "11:04:25.012"
    "text" : string "yes"
    "user" : string "virtual-user-0"
    "audio" : string "11.04.24.975.wav"
  ▶ "meta" : {} 0 items
}
```

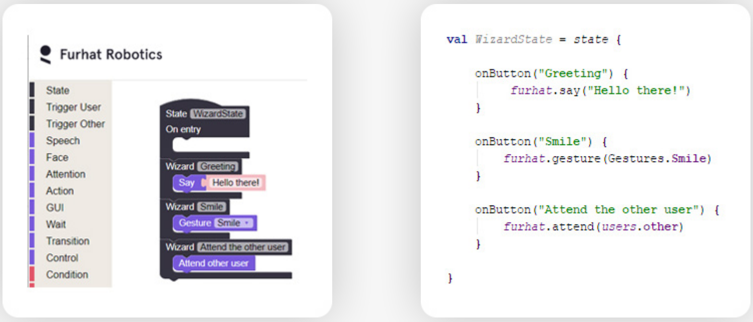
Figure 4: The Log Viewer tool

Graphical programming using Blockly

While the Skill Framework is very powerful, it naturally requires a certain level of coding experience from the developer, and some time to learn the Skill API. We therefore also offer a graphical programming environment called Blockly. Using this, you can build simple interaction flows, with states and triggers, according to the principles outlined above. However, instead of writing code, you drag and connect blocks from a toolbox onto a canvas, as shown in Figure 5. You can immediately run your flow (on either the physical or the virtual robot) and watch how it is being executed, without any need to first compile anything.

Using Blockly is a very good (and fun!) way to familiarize yourself with the basic principles of skill development and the functionality of the Furhat Robot. It is also very good if you want to build simple Wizard-of-Oz interactions (as shown in Figure 5), if you want to prototype a skill, or to make stage performances. It can also be used in education.

However, it should be noted that there is a limit as to how complex interactions you can build with this tool, and you will not be able to integrate your interactions with external components, such as databases or cloud services.



```
val WizardState = state {

    onButton("Greeting") {
         furhat.say("Hello there!")
    }

    onButton("Smile") {
        furhat.gesture(Gestures.Smile)
    }

    onButton("Attend the other user") {
        furhat.attend(users.other)
    }

}
```

Figure 5: Blockly programming vs. programming the flow in Kotlin.

Summary

The Furhat Robot is a state-of-the art social and conversational robot and includes a versatile platform to create natural interactions, and the potential applications of the Furhat Robot are many. Users of the robot have very different backgrounds, therefore, we offer three different ways of programming the Furhat Robot, to accommodate these different needs. The strengths and limitations of these approaches are summarized in the table below:

	Strengths	Limitations
Remote API	• Support for 50+ programming languages. • Easy to get started and integrate with existing software.	• Limited to basic robot I/O. • Requires another dialog system running on a separate system.
Skill Framework & Skill API	• Support for all functionality in the Furhat platform, including tools such as dialog flow, NLU, Wizard-of-Oz and Logging. • Can be used to build complete interactions (Skills). • Skills can be packaged and run on the robot.	• You have to learn Kotlin. • If other components or frameworks you want to use are not Kotlin or Java-based, it will require a more advanced integration.
Blockly	• Very easy to get started and learn. • Quick iterations while developing and running (no need to compile).	• Can only be used for simpler applications.

Contact hello@furhatrobotics.com

More info www.furhatrobotics.com

Follow

FurhatRobotics

@furhatrobotics

Furhat Robotics