Furhat Conversational Platform

Contents

FurhatOS: A Groundbreaking Conversational System 3

Sensor fusion and situation modeling 6

Output processing and behavior decomposition 8

Modeling conversational context 10

Natural language understanding 14

Summary 15

Glossary 16

FurhatOS: A Groundbreaking Conversational System

This white paper aims to provide a basic technical overview of FurhatOS for the novice reader. It offers insight into how FurhatOS differs from traditional conversational systems, how the OS enables a broad range of face-to-face human-robot interactions, how these interactions are handled in real time and why this is essential. The paper also describes the different types of sensors, modules and actions which make this all possible.

To understand the principles behind FurhatOS, it is important to highlight the differences between typical spoken interactions with for example voice assistants, and the social face-to-face interaction that FurhatOS is designed for.

Traditionally, conversational systems have been designed to handle the exchange of speech in task-oriented dialog, such as ticket booking or weather information queries. In such voice-only applications, the physical space where the interaction takes place is typically neglected, and the visual channel is not used at all. Think *Alexa* or *Siri* - it doesn't matter where exactly the user is standing, what direction the user is facing, or whether the user is smiling or frowning; the system is audio only. Moreover, the system is assumed to interact with a single user. The system does not recognize the difference between one user speaking or several.

In contrast, FurhatOS is a conversational platform for face-to-face human-robot interaction. Suddenly, a much broader range of interaction scenarios is possible. With FurhatOS, several users may interact with the robot in one interaction. The visual channel (such as facial expressions) and the physical situation are taken into account, and many different types of interactions can be modelled.
FurhatOS is a modular and expandable system, consisting of a large set of predefined modules that handle visual input, speech input, situation modeling, behavior control, etc. This gives a very powerful platform for modeling social interaction.

The combination of all these techniques is no trivial matter. Face-to-face interaction involves a large number of real-time events that need to be orchestrated to handle phenomena such as overlaps, interruptions, coordination of head pose and gaze in turn-taking, etc. Therefore, FurhatOS has been designed to allow the developer to author the dialog flow in a way that is simple to understand for the novice, yet powerful enough to model more sophisticated behaviors.

Another key difference between traditional conversational systems and FurhatOS is that the traditional systems typically model the interaction on a turn-by-turn basis, essentially mapping questions from the user to responses from the system. The interaction feels flat or linear, even one-dimensional. In contrast, social human-robot interaction must be modelled in *real time*, since many things happen simultaneously and continuously in different communication channels (such as speech, gestures and gaze).

A very simple example is illustrated in Figure 1. A user approaches the robot; the robot looks up at the user and says, *"Hello there"* . The user replies *”Hi,”* but then leaves the interaction, after which the robot looks down again.

To model this interaction, FurhatOS distributes several different events that represent the sensations and actions that take place. For example, the vision component generates a `sense.user.enter` event when the user enters the interaction, which in turn should cause the robot to follow the user with the gaze (as issued with the `action.gaze event`). This also causes the robot to start greeting the user (the `action.speech` event triggers the text-to-speech, TTS, component to start speaking). When speech from the user is detected (`sense.speech.start`), this triggers the robot to smile (`action.gesture`). As this simple example illustrates, there is a continuous flow of events representing things that the robot perceives and actions that are being executed. In this white paper, we will describe how the different modules in FurhatOS generate and trigger these events, resulting in a complex behavior which is transforming - the way humans interact with robots.

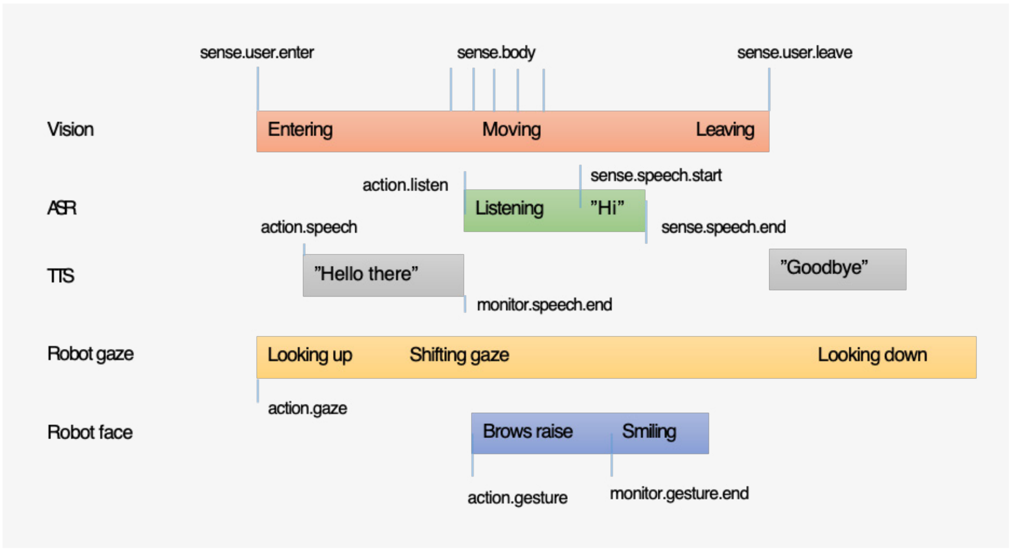

Figure 1: Different events occurring in different modules of the system, illustrated on a timeline.

Essentially, FurhatOS consists of a number of different modules, where each module may react to, and produce, new events. There are three basic types of events: `action`, `sense` and `monitor`.

`Action` represents things that the system should do, such as saying something, producing a gesture, or turning the head. `Sense` represents things that the system perceives (such as a user saying something or approaching Furhat). `Monitor` represents feedback sensations that modules are required to produce when performing an action. Such events are essential to real-time systems, since they allow other modules to know when actions have been performed or are about to be performed, which facilitates synchronization, sequencing, and interruptions of actions (for example if the utterance the system produces needs to be interrupted because the user is leaving the interaction in the middle of it).

Sensor fusion and situation modeling

An important sensor stream for face-to-face interaction is using the robot’s camera to track the location and rotation of the users' heads, as well as their facial expressions. Another important sensor stream is the speech from the users, as picked up by the robot’s microphone. FurhatOS supports the use of an array microphone (with beam forming and echo cancellation), or close-talking microphones, depending on the setting. FurhatOS also has several modules that integrate with cloud-based automatic speech recognizers (ASR) from providers such as Google and Microsoft.

To integrate the low-level events from these sensors, a situation model is being constructed. Figure 2 illustrates how this is being done in a game, where the users and the robot discuss the sorting of a set of cards on a table. These could potentially be physical cards detected with cameras, but in this example, we have used a touch table to display the cards and detect their location. The situation model (shown from the top in the figure) takes low-level events from the different sensors (camera, touch table, and microphones); creates a 3D representation of the situation; and then generates high-level events for the combined sensory data.

Also, if there are several sensors tracking the same users and objects, the situation model can merge these streams into one coherent model, and map sensory events to a common set of user IDs. That means speech recognition results from the microphones can be mapped to the right users based on their location, regardless of whether it is a microphone array or a close-up microphone. The system knows which user is talking when.

Another task of the situation model is to keep track of when users enter and leave the interaction space of the robot, which will trigger the robot to start and end the interaction. Finally, when (card game skill) cards are being introduced or moved on the table, the situation model translates the 3D coordinates, and computes which items are being moved the most (which means that they should be in focus).

Each user object in the situation model contains a record with variables that store information about the user. Thus, it is easy to store and retrieve personal long-term information (such as a user’s name), but also more temporary information, such as the itinerary during the booking of a trip. Furhat Robotics is currently working on integrating face recognition into the framework, which will make it possible to seamlessly retrieve long-term information about the user as soon as he/she appears in the situation model.

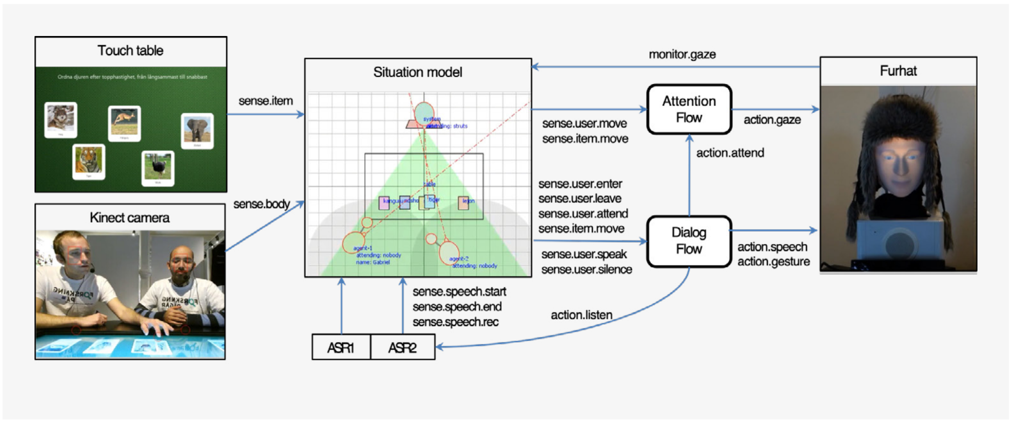

Figure 2: Overview of the different components and some of the events flowing in an example application.

Output processing and behavior decomposition

As can be seen in Figure 2, the higher-level sensory data is sent from the Situation Model to the Dialog Flow and The Attention Flow. Both these modules act as controllers in the system; that is, they take sensory events and map them to action events.

The task of the Dialog Flow is to manage the turn-taking and to generate responses from Furhat. The *Flow* mechanism will be described in more detail in the next section, but is basically a collection of states, where each state represents the current context. As will be explained, the states are defined in a hierarchical manner, which drastically reduces the number of states needed. This means complicated interactions can still be handled in a relatively simple manner. For example, the relatively complex card sorting game described above only has around 30 different dialog states.

The task of the Attention Flow is to tell Furhat which specific target to pay attention to at any moment. The Attention Flow receives continuous events about users and items moving from the Situation model, and sends continuous `action.gaze` events so that Furhat's attention to a certain target is maintained.

The Attention Flow can be in several different states (as instructed by the Dialog Flow): **Idle** (looking down, waiting for someone to interact), **AttendingItem** (looking at one of the cards), **AttendingUser** (looking at one of the users), or **AttendingAll** (shifting the gaze between the users).

The 3D position of the target is transformed into both neck and gaze movements of Furhat (taking Furhat's position in the Situation model into account). If a shift in gaze is small, only the eyes move; otherwise the eyes move first, after which the neck movement follows, but this can be configured.

FurhatOS supports several different speech synthesizers from major third-parties such as *Acapela*, *CereProc* and *Amazon*. The synthesized speech is then synchronized with the lip movements in the facial animation, using a lip animation model developed by Furhat Robotics. The face also supports a number of different facial expressions, which we call *gestures*, that can easily be extended by the developer.

Complex actions that the developer has control over, such as Say, Listen and Attend, are referred to as behaviors. When issuing such a behavior, several different parameters can be set to control, for example, how the attention should be shifted, and for how long the system should listen before giving up (if the user doesn't say anything). Furhat Robotics is building a library of more high-level behaviors that can make application development even more efficient.

An example of such a high-level behavior is shown in Figure 3. In this example, the behavior Greet relies on the behavior AskEach, etc. The behavior decomposition will then break this down to lower-level behaviors, attending all users one at a time, making sure they attend to the robot, ask them the question, listen for the answer, store the answers in the right user slots, ask necessary confirmation questions, etc.

Another benefit of such a library of reusable high-level behaviors is that this can reduce the need for a dialog designer with thorough knowledge to accomplish successful human-robot interaction - this detailed knowledge can instead be easily encoded through these high-level behaviors. It will also make it possible for developers to define their own generic behaviors and then share them with other developers. It's a system which is simple and scalable yet powerful, which can quickly lead to a whole new world of high-level behaviors and complex human-robot interactions.

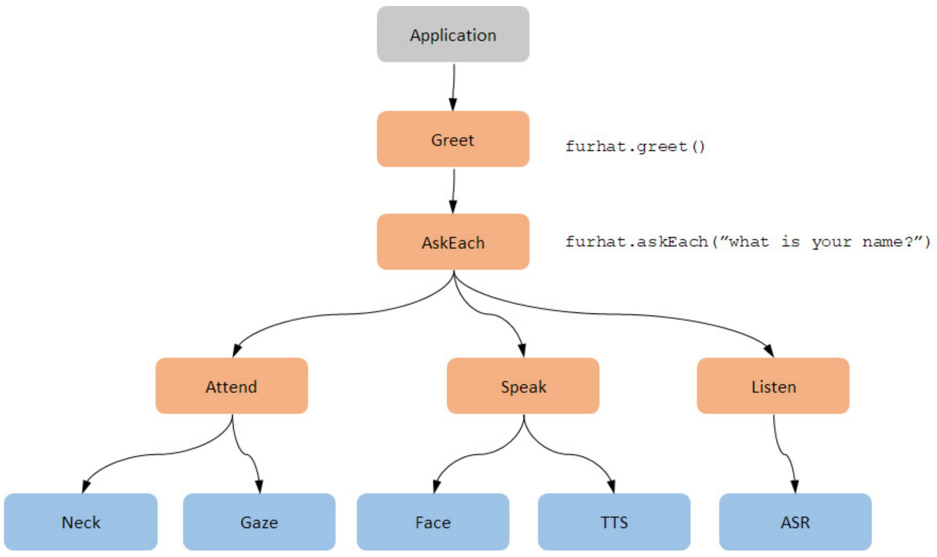

Figure 3: Example behavior decomposition

Modeling conversational context

A central problem of dialog management is that of selecting the next system action - in other words, mapping sensations to actions.

The most straightforward approach is a direct mapping of the last user utterance to a system response, which may work for very simple question-answering systems. However, since dialog is inherently context-dependent, the selection of action often needs to depend on the current *dialog* state.

A common model which takes the dialog state into account is finite state machines (FSM). FSMs are attractive in that they can be easily visualized and the flow is easy to understand. However, this only holds for simple models. FSM do not model mixed-initiative dialogs very well. In such dialogues, the user is not only expected to answer questions from the system, but can also ask other questions or shift topic at any point.

For example, if the user should be able to ask "*What is your name?*" at any point in the dialogue, a transition for this must be added to all states. If many such questions should be possible to handle, the number of states and transitions can easily explode.

Another model for representing the dialogue state is the so-called *information* state approach, where a record of variables is used to represent the context, such as the last user utterance, a stack of issues under discussion, etc. A set of rules are then applied that are conditioned on these variables and may cause other variables to change (causing other rules to apply), or result in actions that the system can take (similar to the concept of a *production system* in AI). This approach may scale better to more complex interaction patterns and domains than FSM, since the number of possible states the system can be in (in effect, all possible combinations of the variables) doesn't have to be itemized (unlike for FSMs). However, since these states are not explicitly defined, it may be hard to anticipate and get an overview of the system's behavior, as the number of variables and update rules grows. This also means that the state-space cannot be easily visualized, as is the case with (smaller) FSMs.

A powerful formalism for defining complex, reactive, event-driven systems - which is used in FurhatOS - is called *statecharts* (sometimes referred to as *Harel Statecharts* after its inventor David Harel). When applied to dialog control, this paradigm can be regarded as a middle ground between the two approaches outlined above. Just like FSM, the dialog state is represented as a set of predefined states with transitions that are triggered by events. However, Harel added a number of extensions that drastically reduce the number of states and transitions needed.

First, the states can be *hierarchically structured*; allowing the designer to define generic event handlers on one level, and more specific event handlers in the sub-states. Second, it is possible to add a *datamodel* (a set of global variables, much like the information state), which may affect further execution. Third, it is possible to add *guard conditions* to transitions, which are dependent on event parameters and the state of the datamodel. Fourth, transitions between states, or the entering or leaving of a state, can also be associated with *actions*, which can be used to raise internal or external events that either affect the further execution of the statechart, or give rise to actions in the other parts of the system (such as requesting the speech synthesizer to say something).

To simplify the authoring of the dialog behavior, Furhat Robotics has developed a DSL (Domain Specific Language) for authoring the dialog flow, this DSK can be regarded as a variant of Harel Statecharts.

To the basic statechart model, Furhat Robotics has added several new extensions. The most important extension is the possibility of recursion (or context-freeness). In addition to the regular *goto* transition, it is also possible to *call* another state. While *goto* corresponds to a traditional statechart transition, *call* makes the execution model put the current state and execution plan on a call stack before the transition takes place. The called state can then *return* to the calling state and execution plan without any explicit reference to a return point. The called state can also take parameters that affect the execution. This mechanism is used to define the *behaviors* outlined above.

For example, a set of states (i.e., a flow) can be defined that implements the *Ask* behavior. This flow is then *called* from the application flow whenever the robot asks the user something, and returns to the application when the behavior has been completed.

When designing a conversational application with FurhatOS, it is useful to first define a state hierarchy, where more specific states inherit more generic states.

A simple example of such a hierarchy is shown in Figure 4. In the example, we imagine a robot operating to take hamburger orders. On the top, we find more generic states. When no customer is engaging with the robot, it is in an Idle state. When a customer appears, it goes to the Order state. This state has in turn a number of sub-states for Greeting the customer, requesting the main order, what to drink, what to have on the side, and finally the closing.

There may also be even more specific states, such as talking about the flavor of the drink. In the more abstract states, more generic event handlers (that are the same for several states) may be placed, such as what to do if the user suddenly leaves the interaction, or what to say if the robot does not understand the user.

All events that are received are first checked against the most specific state the system is currently in (the lowest one in the hierarchy), and then against the more generic states (higher up) in the hierarchy. Thus, more specific states can override the event handlers in the more generic states.

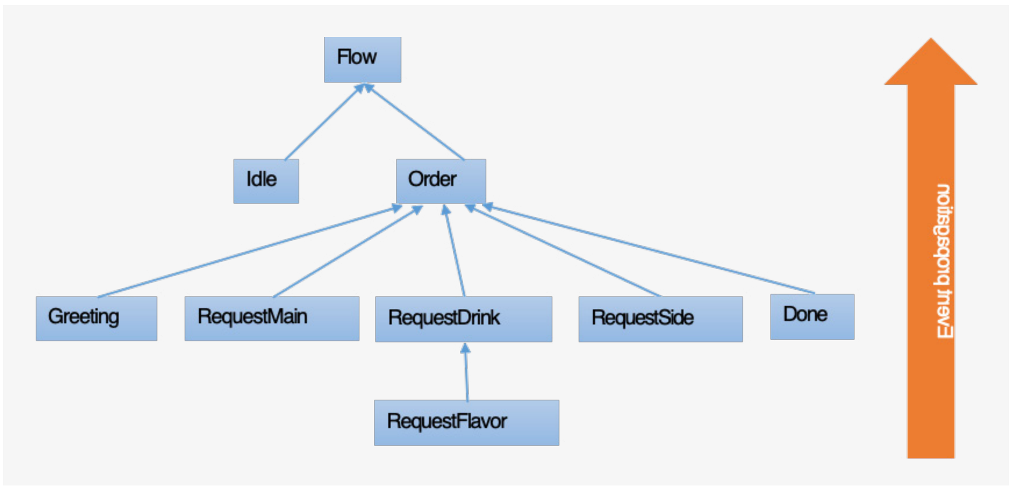

Figure 4: Modeling the hierarchical states

Figure 5 shows what this dialogue may look like. When the user walks up to the robot, it starts by asking what the user wants. When the user says that he wants a cheese burger, the system sees that it doesn't know what the user wants to drink, so it asks this (by going to the RequestDrink state). The user now *over-answers* this: he provides both the drink ("*a milkshake*") and the side order ("*some fries*"). However, this is fine, still the more generic state Order takes care of registering the side order. The system also detects that it does not know the flavour of the milkshake, so it goes to the RequestFlavor state. The user now asks, "*what options are there?*", which is interpreted in context of the RequestFlavor state (if it would have been in for example the RequestSide state, the answer would have been different). After hearing the options, the user changes his mind and instead orders a coke, which is handled by the higher-level RequestDrink state. When the system then asks for the size of the fries, the system can interpret that fragmentary answer "*large*" in the current context (RequestSide).

What this example shows is how language is inherently context-dependent, and that users do not always give exact answers to the question that was asked. The hierarchical context model in FurhatOS is designed to handle these things, creating a smooth, seamless user experience. The human-robot interaction is much closer to how a human-human interaction would look.

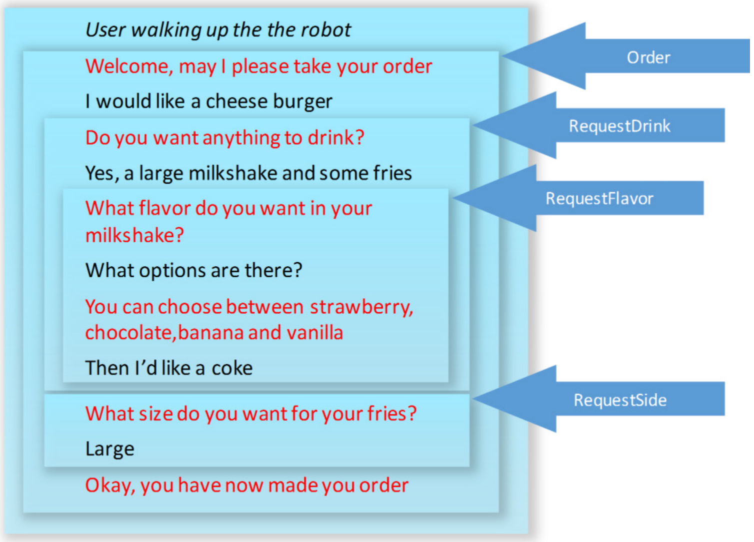
User walking up the the robot
Welcome, may I please take your order
I would like a cheese burger
Do you want anything to drink?
Yes, a large milkshake and some fries
What flavor do you want in your milkshake?
What options are there?
You can choose between strawberry, chocolate,banana and vanilla
Then I'd like a coke
What size do you want for your fries?
Large
Okay, you have now made you order
Order
RequestDrink
RequestFlavor
RequestSide

Figure 5: Example dialog with hierarchical states.

Natural language understanding

To process the meaning of what the user says, most systems today, including FurhatOS, are based on the notion of *intents & entities*, as illustrated in Figure 6. Intents represent the overall meaning of the utterance, such as Greeting or OrderPizza, whereas Entities are important concepts found in the text, such as Time ("*3 pm*") or Topping ("*pepper*" and "*ham*"). Entities can typically be user-defined for a specific application, but FurhatOS also provide a set of pre-built entities for generic things like Date, Time, PersonName, Number, etc. The overall intent is identified using machine learning, where the developer provides examples of the different Intents, eventually complemented with things that users say when interacting with the system.

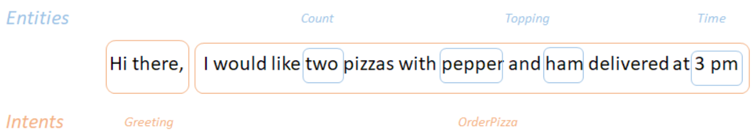
Entities
Count
Topping
Time
Hi there,
I would like two pizzas with pepper and ham delivered at 3 pm
Intents
Greeting
OrderPizza

Figure 6: Example of intent and entity detection.

Compared to other platforms, Furhat OS adds a number of novel advanced features that allow implementation of more natural & flexible conversations:

- Entities can be programmatically defined using a number of different tools & algorithms, including grammars, dictionaries, knowledge graphs and wildcards.

- Whereas most systems separate natural language understanding from dialog management & treat them as two separate steps), we tie it to the state hierarchy in dialog management (see Section 4). In this way, it becomes possible for the dialog author to define potential intents (things the user might say) on various levels in the state hierarchy. For example, more generic things (like "What did you say?") can be defined at the top & be active all the time, whereas more specific, context-dependent things (like "No, red") can be defined further down. At any point in the dialog, the current state hierarchy is examined & all potential intent triggers on different levels of the hierarchy are automatically collected & compiled into a context-sensitive intent classifier on-the-fly, without the developer having to think about it.

- Our natural language understanding module can handle utterances with multiple intents, as shown in Figure 6. This multiple-intent classification is automatically constructed from the different triggers for these different intents.

Summary

FurhatOS is a conversational platform for face-to-face human-robot interaction. Compared to conversational systems for simpler voice-only interactions - targeted towards chatbots, smartphones and smart speakers - FurhatOS has several unique properties:

- Instead of modelling the interaction on a turn-by-turn bases, FurhatOS models the interaction in real time, using various multi-modal events, allowing Furhat to do things like gazing and smiling at a newly arrived user in the middle of a sentence, and nodding while the user is speaking.

- Sensory input from cameras and microphones are combined into one situation model that can be used to understand where users are located, who is engaged in the interaction, who is speaking, and where their attention is directed.

- Simpler behaviors (involving both speech, gaze, and gestures) can be combined into more complex behaviors that can be reused across applications.

- The context of the interaction is modelled using hierarchical states, which allows Furhat to engage in mixed-initiative interactions and react to events with both generic and specific behaviors. This allows Furhat to engage in many different types of interactions where the dialog context is of importance, not just simple question-answering.

- The natural language understanding (NLU) is tightly coupled with the hierarchical states machine, allowing it to be highly context-dependent.

Glossary

API	Application Programming Interface
ASR	Automatic Speech Recognition
GUI	Graphical User Interface
I/O	Input Output
FSM	Finite State Machine
NLP	Natural Language Processing
NLU	Natural Language Understanding
OS	Operating System
SDK	Software Development Kit
TTS	Text To Speech

Contact hello@furhatrobotics.com

More info www.furhatrobotics.com

Follow

FurhatRobotics

@furhatrobotics

Furhat Robotics