Furhat Therapy Robot - Documentation

Fredrik Angström
Tim Carsting
Petra Faber
Jens Gummesson
Alexander Henne
Daniel Mastell
Jesper Mjörnman
Joel Tell

2020-12-16

Contents

Introduction

This documentation refers to an implementation of the Furhat robot for therapeutic use, designed as part of the course TDDI17 - Program Development Project at Linköping University during the autumn semester of 2020.

The goal of the project is to:

- Implement a framework to allow for using the Furhat robot in therapy sessions with one patient, using wizard-of-Oz-style direct control by a remote therapist.
- Implement a structured dialog flow, where the robot may communicate autonomously.

The project has been performed partly by working with the preexisting Furhat interface and system, and has been complemented by a custom user interface enabling finer control over relevant functions, as well as adding audio streaming, voice-to-text from the therapist, and more.

As this system is built using the Furhat robot system as a base, familiarity with the base system is recommended to complement the information in this document. At the moment of writing, the documentation for the base system is available here.

1 Setup

This chapter describes the process of setting up the Furhat therapy robot.

1.1 Starting the robot

Using the Furhat therapy skill requires a Furhat robot according to the specifications found here. Make sure that the robot matches the specifications, and perform any necessary updates. Detailed instructions for setting up the robot can be found here. Note that Furhat needs to be connected to the internet for the skills to function correctly.

After setting up the robot, the user will have access to the Furhat Studio interface. Please use this interface to confirm that the robot is properly connected.

1.2 Importing the skill

The main program of the Furhat therapy robot is a skill, which may be imported to the robot through the online interface, using Skills → Import Skill. By default, the skill may be found in the git repo at /main/Therapist/build/libs/Therapist-all.skill.

After importing the skill, the skill can be started through the same interface by pressing the play button. At this point, the skill should start and a skill running notification should be available in the interface.

Upon starting the skill, the robot will immediately start looking for users, asking the people in view if they want to speak to the robot. This behaviour confirms that the skill is running correctly.

At this point, no more setup is required to run the robot in the scripted dialog state. However, advanced and user-controlled functions are absent or limited. To setup these functions, proceed to the next section.

1.3 Setting up the user interface

The first step of setting up the user interface is making sure that all the requirements are available. There is a script located the UI directory named *install.sh* that should install all the requirements for Arch Linux, Manjaro, Ubuntu and Mac OS automatically. After installing the requirements, starting the UI is a matter of running the /UI/furhatui.py script. When running the script, the UI window should open.

1.4 Setting up the log viewer

Again, first check the requirements, and make sure that everything is installed, then run the file /ExternalScrpits/LogScripts/LogViewer/run_linux.sh. The program will then load all logs from the Furhat robot, which may be viewed and exported.

2 User interface

This chapter contains information on the Furhat therapy robot custom user interface. Before using the interface, please check the requirements.

2.1 Overview

The user interface consist of three parts: a camerafeed that shows the view of the therapy robot, a chat that shows the interaction of the session and also enables the user to control the robot through text and buttons for default phrases, gestures and changing states.

2.2 Default phrases

The user interface contains a section where default phrases can be loaded and used. The intent is to use these default phrases to quickly respond in generic ways to promote quicker and more fluid interactivity, and to allow the therapist time to formulate tailored responses.

To create a standard phrase for the user interface, write the desired phrase in *defaultphrases.txt* using the following syntax: *button name :phrase*

Important to note is that you need to write the phrase on the same line as your button.

Example: *(in defaultphrases.txt)* Hello :Hi, how are you doing today?

This example entry will result in a button labeled *Hello*. When pushing the button, Furhat will say "Hi, how are you doing today?".

The default phrases are loaded into the user interface at startup, meaning that they can not be added during operation of the user interface. Instead, add the desired phrases before starting the interface.

2.3 Gestures

A part of the interface contains buttons that control robot gestures. These can be used during the controlled dialog state to complement the audio provided by the therapist with gestures such as smiling, looking surprised, shaking the robot's head and more.

Gesture buttons can be added by adding the button in *gestures.txt* in the user interface folder. The format is as follows: *button name:gesture name*

Example: *(in gestures.txt)* Look surprised:Surprise

There are a number of gestures already built into Furhat, such as BigSmile, ExpressSad and Surprise. All of the built in gestures can be found in the Gestures class inside Furhat's API. Most should however already reside in the gestures.txt file.

If you want to make your own gesture you can add it to *customGestures.kt* in the skill code. Simply add a new property of type Gesture to the file and the skill will pick it up. An example custom gesture called LongSmile is provided. Then add a button to gestures.txt in the same way as for built in gestures. For example: *Long Smile:LongSmile*

2.4 Text log

The text log contains a log of previous speech by both users and Furhat. It can be used to track previous statements by both parties. The log is built by subscribing to corresponding events, and adding appropriate text on triggers.

2.5 Text input

The text input field can be used to make Furhat speak custom dialog. The text input field also shows real-time voice recognition results, when using that functionality. Sent messages are appended to the text log.

2.6 Voice

The user interface contains functionality for the user to tell Furhat what to say using voice-recognition and speech to text. More detailed information about speech-recognition can be found here.

2.7 Camera feed and audio

The camera feed is a video stream from Furhat's built-in camera. This will be displayed in the user interface in real-time. Simultaneously audio from Furhat's external microphone will be played on the user's speakers. More detailed information about the camera feed and the audio can be found here.

2.8 State change buttons

The user interface contains a section where state transition buttons can be loaded. These buttons allow for quickly transitioning to different parts of the dialog flow. The buttons in this area are loaded on startup from the file "states.txt". To add a new button, add a line in this file with the format *ButtonLabel :StateName*. Also, make sure that there exists a corresponding event trigger in the therapist skill's *changeState* partial state, located in *general.kt*. Note that the partial state must be included in states that should be available to transition from.

Example:

To transfer from *StateA* to *StateB*, there must be a corresponding line in *states.txt*, there must be an event trigger *onEvent("GoToStateBEvent")* in *changeState* in *general.kt* and *StateA* must include the partial state *changeState*.

2.9 Limitations

Most of the functionality of the Furhat robot is accessible through the user interface, but it has some limitations. Most importantly, the therapist skill can not be started through the interface. The therapist skill is set to autostart on boot, and we recommend that this setting is preserved when developing the system further. This setting can be toggled from the Furhat studio. Alternatively, the skill can be started from the Furhat studio.

Due to the user interface being dependant on the audio and video stream which is started in the skill initialization, the skill needs to be running before the user interface is started.

Furthermore, due to the Furhat robot's system implementation, any user disconnecting from the UI will cancel any active skill, as disconnecting from the internal broker which the UI uses to communicate with the robot explicitly cancels the currently running skill. As far as we understand, this is the Furhat developers' intended behaviour. For this reason, closing the user interface will abort any skill in progress, and users are recommended to keep the interface open until running the skill is no longer desired.

The text log in the user interface can only log what the patient is saying in the scripted dialog mode. The robot only sends events for detected speech when it is listening for a reply, which is not the case in the free dialog mode.

3 Attention and users

This chapter contains information regarding handling of users and robot attention.

This implementation of the Furhat system does not use the built in attendance policies. This is because the intent is always to attend a single user for a therapy session, and other users should not be attended. Neither should a new user automatically be attended simply because the previous user disappeared, since it is especially important to ensure that the actual user is consistent in this context.

To facilitate this behaviour, two new variables are added to the skill, present in *user.kt*, as well as one extension to the user object itself.

3.1 targetUser & hasTargetUser

The two new extensions to the skill, *targetUser* and *hasTargetUser* are used to keep track of the attended user. When a user is attended, the user ID is saved to the *targetUser* variable, and the *hasTargetUser* boolean is set to true. As a result, the robot will stop attending new users, and will also be able to check if the user leaving is the currently attended user, in the *onUserLeave* trigger.

The intended usage is to always set *hasTargetUser* to false when the currently attended user leaves, to provide a fallback enabling reattendance when the user returns.

These variables are not extensions of the user class, since Furhat can not currently guarantee that a user that has been lost will regain the same user ID on returning.

3.2 User.disregard

The new variable *disregard* is an extension of the user class. The variable is used to keep track of users who have declined to engage Furhat in conversation. This behaviour means that Furhat will not repeatedly address users who have already declined interaction.

3.3 Establishing attendance

In the current implementation, Furhat is initialized without a user to attend. On startup, Furhat asks all present users for attendance. On affirmative reply, the responding user is set as *targetUser* and *hasTargetUser* is set to true. On a negative reply, the user is disregarded. If no present user answers affirmatively, Furhat waits for new *onUserEnter* events, and asks the same question to the new users.

Furhat asks these questions in a state *findTargetUser* which takes a user, as well as the question and response strings as parameters, to make it easy to edit what Furhat says, in case the question needs to be formulated differently depending on the calling state.

4 Streaming

This chapter contains information regarding the streaming from the robot's microphones and camera.

4.1 Listening to the audio stream

When the skill first starts, the microphone streamer queries the system for available microphones. The last microphone in the list of available microphones with support for one of the supported audio formats (8-48KHz, 16bit, mono, little/big-endian) is chosen, which is not necessarily the microphone selected in the microphone settings of the robot's built in control panel. If an external microphone is connected, it should prioritize that.

Please note that the audio streaming functionality does not support the built-in speaker, but requires an external speaker. The audio stream supports a wide variety of formats. If support for another format is required, the format specification may be added to *microphone.kt*.

The microphone streamer then starts a TCP socket server binding on port 8887 where it broadcasts PCM audio data from the selected microphone to all clients. When a new client connects to this microphone streaming server the server first sends it a WAV header detailing the format of the audio data, so that the client can read the audio stream as a simple WAV-format stream.

The client implementation uses Python and connects to the robot on port 8887 where the streaming server resides. It then reads the data from the server, interpreting the WAV header and continually forwarding the audio data to the speakers of the computer that the Python script is running on until the script is terminated.

4.2 Video stream

The video stream capturing takes advantage of the camera feed already built into the Furhat robot. The user interface subscribes to the video stream to a ZeroMQ socket and receives images as a raw bytestream that is then translated to a displayable image with the help of OpenCV.

5 Appearance selection

This chapter contains information about the face and voice selection of Furhat.

5.1 Face selection

Face selection is performed by using the built in function *furhat.setTexture()* which takes the name of the desired face texture as argument. No further functionality has been added to the base Furhat framework, in this respect.

5.2 Voice selection

Currently, the voice selection uses the built-in *furhat.voice()* function to select a voice according to gender and nationality. It currently uses an English voice for the male options, as a male Swedish voice is not available. This should be extended to choose a Swedish male voice as soon as it is available.

When setting the voice, the *furhat.setInputLanguage()* function is also used to ensure that Furhat always expects a Swedish speaker, even if using a foreign voice.

5.3 Appearance selection

The appearance selection is started in the state *AppearanceStateGender*. In this state, the user is asked by furhat to choose between a male and female appearance. Choosing the appearance also switches the voice to an appropriate option. After user confirmation, this choice is locked in and the answer is passed to the next state.

The next state called is *AppearanceStateSpecifics*. In this state, the user is then given an option between two different faces, with corresponding voices, according to their choice in the previous state.

5.4 Helper functions

To determine user preference, Furhat listens for user answers using custom intents found in the file *nlu.kt*. The relevant intents for this application are called *MaleIntent*, *FemaleIntent*, *Nr1Intent* and *Nr2Intent*. The first two of these intents are used to determine whether the user prefers a male or female therapist, and the second two are used to determine the specific appearance in *AppearanceStateSpecifics*. These intents can be freely extended with text strings to improve the voice recognition.

6 Logs

This chapter contains all information about logs, how to start a logging session, as well as reading and exporting them. (Opens port 8888 for the client connection.)

Exporting is possible by either exporting in an existing skill or creating a skill that calls for the log to export and after exits.

6.1 Logging dialog and states

The logger uses the built-in Furhat *dialogLogger* which logs the interactions between the user and robot. These logs are saved in (.json) format in */home/furnix/logs* in the local storage of the robot. Handling this is done by a class named *Logger* which is found in the file *logger.kt*. The *Logger* handles starting or ending logging sessions and sending the files to a client, if instructed to.

The recommended usage is declaring a global variable, since only one *Logger* object is allowed at a time. Starting and ending the logging session can be done anywhere in any skill using the declared variable. Setting the *Logger*'s *autoExport* to false will disable the export at the end of session. The logs can then be exported manually by calling any export function that is local-based. Note that the robot must be online and the client must be connected via the *FurhatLogConnection.py* script (Fetching the logs on external client).

6.2 Functions and declarations

autoExport	If true auto export most recent log on stopLogging
cloudTokenApi	The token used in order to upload logging sessions to the cloud.
sessionName	Name of the logging session based on the default AWS format: ("YYYY-MM-dd%YYHH-mm-ss").
logger	Logger object taking use of Furhat DialogLogger for logging sessions.
servSocket	Socket used in order to connect a client to the Robot. Default port is 8888.
startLogging(token: String? = null	Starts a logging session. Token is the Furhat cloud API key. If the token is null no log will be uploaded to the furhat cloud.
getLogger(): DialogLogger	Get the active DialogLogger object.
getCloudtoken(): String?	Get the current logging session ID.
getDate(): String	Get the current date: Year-Month-Day.
exportActiveLog()	Ends the logging sessions and exports the most recent log from the Robot's memory.
clearLogsFromFurhatMem(date: String? = null)	Clear the logs from the Robot's memory. Date is the specific logs to be deleted in format "YYYY-MM-dd". If date is null all files will be removed.
export(arg: String? = null, clear: Boolean = false)	Export the logs from the Robot's memory to the client. Arg refers to what type of export. If clear is true the exported file will be deleted. Arguments: • "all" - exports all logs. • "YYYY-MM-dd" - removes logs with the specified date/day. • null - exports the latest log.
exportDateLogs(date: String, clear: Boolean = false)	Export all logs from the logs folder with specified date ("YYYY-MM-DD"). If clear is set all logs will be removed after being exported.
exportAllLocalFiles(clear: Boolean = false)	Export all logs from the logs folder. If clear is set all logs will be removed after being exported.
findNewLogName(path String) : File?	Finds the most recently modified log file. Returns the File if found, else null.
stopLogging()	Stop the current logging session. Exports all logs to /user/Furhat-logs/.

6.3 Fetching the logs on external client

In order to fetch the logs that are local on the robot, the Python script *FurhatLogConnection.py*, with Furhat's IP as an argument, is needed to run. The script connects to the Furhat IP through port 8888. First, the timestamp is received and decoded. Depending on what format the timestamp is in it will then receive data in different ways (either cloud from http or from files sent by the robot). The received data after the timestamp is then written to a *dialog.json*-file in the folder of the received timestamp. Note that the timestamp may be off by +/- 3 seconds, as the timestamps created by the robot are somewhat unreliable.

6.4 FurhatLogConnection.py

This script connects to the robot via port 8888 and receives data via network. Depending on what the received filename is, it will format the name to be "YYYY-MM-dd HH-mm-ss". The received name will decide if the script will fetch data from the cloud or directly from the robot. The fetched files are then placed in */home/FurhatLogs*.

6.5 LogViewer

The *LogViewer* presents a UI for reading and exporting logs that have been fetched from the robot. The program searches through the *FurhatLogs* folder for the logs. This folder will be created when using the *FurhatLogConnection* to receive logs from the robot.

7 Modes of operation

The Furhat therapy robot has two different modes of operation. The robot may act autonomously according to a predefined script, this is intended as the normal operating procedure. The robot may also enter a therapist-controlled mode, where the robot simply tracks the user and a remote therapist controls how the robot responds.

7.1 The scripted dialog mode

In this mode, the robot's interaction with the patient follows a predefined script. This is the normal operating procedure for the robot, and as such it is automatically entered when starting the therapist skill. This mode is defined through many different robot flow states, following the conventions described in the official Furhat documentation. Modifying this flow is a matter of adding or modifying existing states, or editing which states flow into each other depending on patient interaction.

7.2 The therapist-controlled dialog mode

Through the user interface, the robot may be changed to the therapist-controlled dialog mode. In this state, normal operation is suspended. This mode suspends the normal dialog flow in an empty state, and has no predefined interaction aside from the normal patient tracking functionality. Instead, this mode can be used to allow the therapist to interact freely with a patient. This is done through the user interface, where the therapist can use text to speech and predefined phrases to prompt Furhat to repeat to the patient what the therapist is saying.

7.3 Transitioning between modes

To transition between modes, Furhat must be prompted to do so through a trigger. The *ChangeModeEvent* trigger is sent by the UI and caught by the robot. Upon catching it the *ChangeModeEvent* runs. When the event is run in scripted dialog mode then a call to therapist-controlled dialog is made and when it's run in the therapist-controlled dialog it simply terminates and returns to scripted dialog.

Note that for Furhat to be able to enter the therapist-controlled mode, the partial state *goToControlledDialog* must be included in the departure state. This partial state contains the *onEvent* handler responsible for moving to the controlled dialog state.

8 Speech recognition

This chapter contains information about the *Speech Recognition* used to enable the therapist to speak through the Furhat robot. The requirements for using the speech recognition can be found here.

The recognizer supports both asynchronous and synchronous recognition. Where the synchronous refers to recording a file first and the sending it to Google for recognition and the asynchronous is a streaming recognition.

The Furhat robot's implementation uses the streaming, or asynchronous, recognition which means that each utterance is sent to the Google Cloud API which handles the recognition in real-time when you speak. This approach is reliant on good internet connection, good microphone and quiet surroundings. If any of these are missing or lacking the process may suffer from delay and/or bad recognition results. The returned result from the Google Cloud API is then sent as a text for the Furhat robot to speak.

9 TCP Interface

This chapter describes the process of communicating with the Furhat therapy robot. The communication between the custom user interface and the Furhat robot is implemented with Furhat's built in event system over TCP.

9.1 Event System

The built in event system uses JSON-serialized objects, both to tell the user interface what has occurred on the robot and also to receive action events to perform. The event types have names that distinguish them from one another and individual event objects have a unique ID and timestamp. The different event types contain different fields, depending on information needed in each case.

9.2 Implementation

This chapter contains implementation details regarding the implementation of the TCP interface.

9.2.1 furhatinterface.py

The *furhatinterface.py* python script contains both the interface and the TCP client that communicates with the robot. The class *FurhatTCPConnection* handles the underlying communication. The class *FurhatInterface* uses the *FurhatTCPConnection* and contains the functions to subscribe to, and send, events. This separation allows the developer to freely extend interface functionality, without having to modify the communication functionality.

9.2.2 furhatevent.py

The *FurhatEvent* class, located in the file *furhatevents.py*, is an abstract class that the other defined events inherit from. The file contains contains a number of events used in the current implementation, and can be extended to include other types of events supported by Furhat.

9.2.3 Subscribing to an event

The subscribe function has two parameters: the first parameter is the name of the event, the second parameter is a callback. The callback is triggered when the interface receives an event from the robot. The callback function must allow for one parameter, that parameter is a dictionary of the event.

```
1 def callback_say(event)
2     print(event.text)
3
4 furhat = FurhatInterface("UIClient","192.168.43.131")
5 furhat.subscribe("furhatos.event.actions.ActionSpeech", callback_say)
```

Code Example 1: Subscribing to a speech event

9.2.4 Sending an event

The events that are currently used all have a function defined in the *FurhatInterface* class.

```
1 furhat = FurhatInterface("UIClient","192.168.43.131")
2 furhat.say("Welcome")
```

Code Example 2: Sending a speech event

10 Requirements

This section details the requirements for using the Furhat therapy robot. The Furhat therapy robot system implements the Furhat robotics system, with additions in the form of tailored external controls and improved streaming and storing of video, audio and logs. The system has been developed for use on Linux systems, and has been tested on MacOS. As such, only Linux and Mac are officially supported, and the developers make no guarantees regarding functionality on other platforms.

10.1 The Robot

The Furhat therapy skill has been developed using a Furhat robot marked for research use with FurhatOS version 1.21.0. At the time of writing, Furhat robots marked for commercial use do not allow for exporting of the video stream, which would limit the use of the user interface as well as limit the possibility of saving actionable information about therapy sessions.

10.2 User interface

The user infterface for the Furhat therapy robot implements multiple features, including:

- Audio feed
- Video feed
- Text-to-speech
- Standard phrases
- Voice-to-text-to-speech
- Dialog flow control

As each of these functions have different requirements, to use the full functionality of the UI, the services below are required:

- python3
- pip3
- imutils
- numpy
- opencv-python==4.1.2.30
- pyzmq
- pyaudio
- pyside2
- ffmpeg-python
- google-cloud-speech
- A Google Cloud API key
- Wave

10.2.1 Therapist-side Speech Recognition

For the recognition to function three Python packages are needed: *PyAudio, Wave, google-cloud-speech* and a *Google Cloud API key*. The Google Cloud API key should be saved to `_key/GAPI.json`. It can be found in the speech section of the Furhat's control panel. The required Python packages can be installed by running the `install.sh` script located in the `ExternalScripts/SpeechRecognition/` folder.

10.3 The log viewer

In addition to the user interface, the Furhat therapy robot comes packaged with a log viewer, which may be used to view logs of previous sessions. The log viewer can be used independently of the UI, and has the following requirements:

- python3
- pip3
- tkinter
- fpdf

Tkinter is needed for the user interface of the log viewer, while fpdf is needed to utilize the log export function of the log viewer.