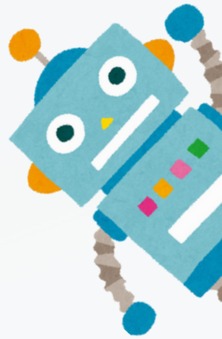
LT2319 | University of Gothenburg

“FEEL IT, SAY IT, SHOW IT”: EMOTIONALLY EXPRESSIVE FURHAT

Andrea Carrión

INTRODUCTION



Motivation

- Genuine interest in social robots.
- Learn how to build engaging, expressive interactions with Furhat.

Goal

- Exploring how to bridge the gap between LLM text and expressive non-verbal behavior on Furhat.
    - Original intention: explorative project
        - Research question: Can we use Furhat's physical and social cues to make the LLM's dialogue feel more natural and believable to the user?
    - BUT → "interface" project

HOW DO WE START?

- Mapping the LLM **response's emotional tone** to Furhat's **gestures** and **voice**.

HOW DO WE START?

- **PROMPT:**

Prompting an LLM (model: “llama3.1”) to provide...

1. An answer to the user’s input.
2. Choose the emotional tone that best matches its response.

```
const voiceStylesList = [
  "cheerful",
  "sad",
  "angry",
  "calm",
  "friendly",
  "empathetic",
  "serious",
  "excited",
  "depressed",
  "hopeful",
];
```

Return JSON object only:

```
{
  "response": "<natural language response text>",
  "voiceStyle": "<chosen emotional tone>"
}
```

HOW DO WE START?

- **GESTURES:**
  - Custom-made and built-in.
  - More than one gesture per emotion → list to link them accordingly.

```
const emotionGestures = {
  cheerful: [cheerfulGestureA, cheerfulGestureB],
  sad: [sadGestureA, sadGestureB],
  angry: [angryGestureA, angryGestureB],
  calm: [calmGestureA, calmGestureB],
  friendly: [friendlyGestureA, friendlyGestureB],
  empathetic: [empatheticGestureA, empatheticGestureB],
  serious: [seriousGestureA, seriousGestureB],
  excited: [excitedGestureA, excitedGestureB],
  depressed: [depressedGestureA, depressedGestureB],
  hopeful: [hopefulGestureA, hopefulGestureB],
};
```

Function: **selectGestureByVoiceStyle**()

- It receives the emotion and picks a random gesture from the corresponding array.

HOW DO WE START?

- **VOICE:**
  - Most challenging part!
  - Furhat uses TTS services from Microsoft Azure and Amazon Polly.

- **Amazon Polly Neural Voices** → "Neural voices can have different styles, currently we support Neutral, Conversational and News"

- **Microsoft Azure Neural Voices** → "Microsoft Azure voices are neural, have **special style tags** and a lot of options."
  - cheerful, sad, angry, calm, friendly, etc...

HOW DO WE START?

- **VOICE:**
  - Different approaches that were changing as I was bumping into technical difficulties:

Approach 1:

- Using Microsoft Azure Neural Voices → ***en-US-AriaNeural*** supported the most styles.

en-US-AriaNeural

`angry`, `chat`, `cheerful`, `customerservice`, `empathetic`, `excited`, `friendly`, `hopeful`, `narration-professional`, `newscast-casual`, `newscast-formal`, `sad`, `shouting`, `terrified`, `unfriendly`, `whispering`

HOW DO WE START?

- **VOICE:**
  - Different approaches that were changing as I was bumping into technical difficulties:

Approach 2:

- Using SSML (Speech Synthesis Markup Language) tags.
  - Embedding the LLM's response on an SSML tag.
  - Advantages: control over speaking style, rate and pitch.

Issues:

- It seems that Furhat's POST /furhat/say?text= endpoint does not interpret SSML, so it treats the SSML-tagged string literally as plain text.
  - SSML support only exists inside the Skill SDK?

HOW DO WE START?

- **VOICE:**
  - Different approaches that were changing as I was bumping into technical difficulties:

Approach 3:

- Since I can't use Azure's styles or SSML tags:
  - Listening to all (27) Azure's (en-US) female voices and manually choosing which one sounds more cheerful, serious, friendly, etc...

Issues:

- Not the best approach, simply a patch for voice style.

IMPLEMENTATION

GetUser
↓
AttendUser
↓
Greeting → invoke *LLMActor* (instructions) → assign llmResponse, llmVoiceStyle and messages (stores the full chat history)
↓
LLMSpeaks → 2 child states:
↓
Gesture → *fhLLMGesture* actor provides the corresponding gesture
Talking → *fhLLMSpeak* actor speaks the LLM reponse using the corresponding selected voice

IMPLEMENTATION

GetUser
↓
AttendUser
↓
Greeting → invoke *LLMActor* (instructions) → assign llmResponse, llmVoiceStyle and messages (stores the full chat history)
↓
LLMSpeaks → 2 child states: **Gesture** / **Talking**

EXAMPLE:
Selected gesture: [AsyncFunction: friendlyGestureB]
Response: Hello there! It's lovely to meet you!
Selected voice: EmmaNeural

IMPLEMENTATION

GetUser
↓
AttendUser
↓
Greeting → invoke *LLMActor* (instructions) → assign llmResponse, llmVoiceStyle and messages (stores the full chat history)
↓
LLMSpeaks → 2 child states: **Gesture** / **Talking**
↓
Listen → 2 child states:
↓
UniversalGestures → *fhUniversalGestures* actor provides a random "work-for-all" gesture (Blink / GazeAway / etc) to add expressiveness while waiting for the user's input.
Listening → *fhL* actor listens to the user's input.

IMPLEMENTATION

GetUser
↓
AttendUser
↓
Greeting → invoke *LLMActor* (instructions) → assign llmResponse, llmVoiceStyle and messages (stores the full chat history)
↓
LLMSpeaks → 2 child states: **Gesture** / **Talking**
↓
Listen → 2 child states: **UniversalGestures** / **Listening**
↓
TestLLM → **1**: guard "*userWantsToEnd*" [stopwords] → **Goodbye** (text, gesture, sound effect) → **End**
↘ **2**: invoke *LLMActor* → **LLMSpeaks**

IMPLEMENTATION

GetUser

↓

AttendUser

↓

Greeting → invoke *LLMActor* (instructions) → assign llmResponse, llmVoiceStyle and messages (stores the full chat history)

↓

LLMSpeaks → 2 child states: **Gesture** / **Talking**

↓

Listen → 2 child states: **UniversalGestures** / **Listening**

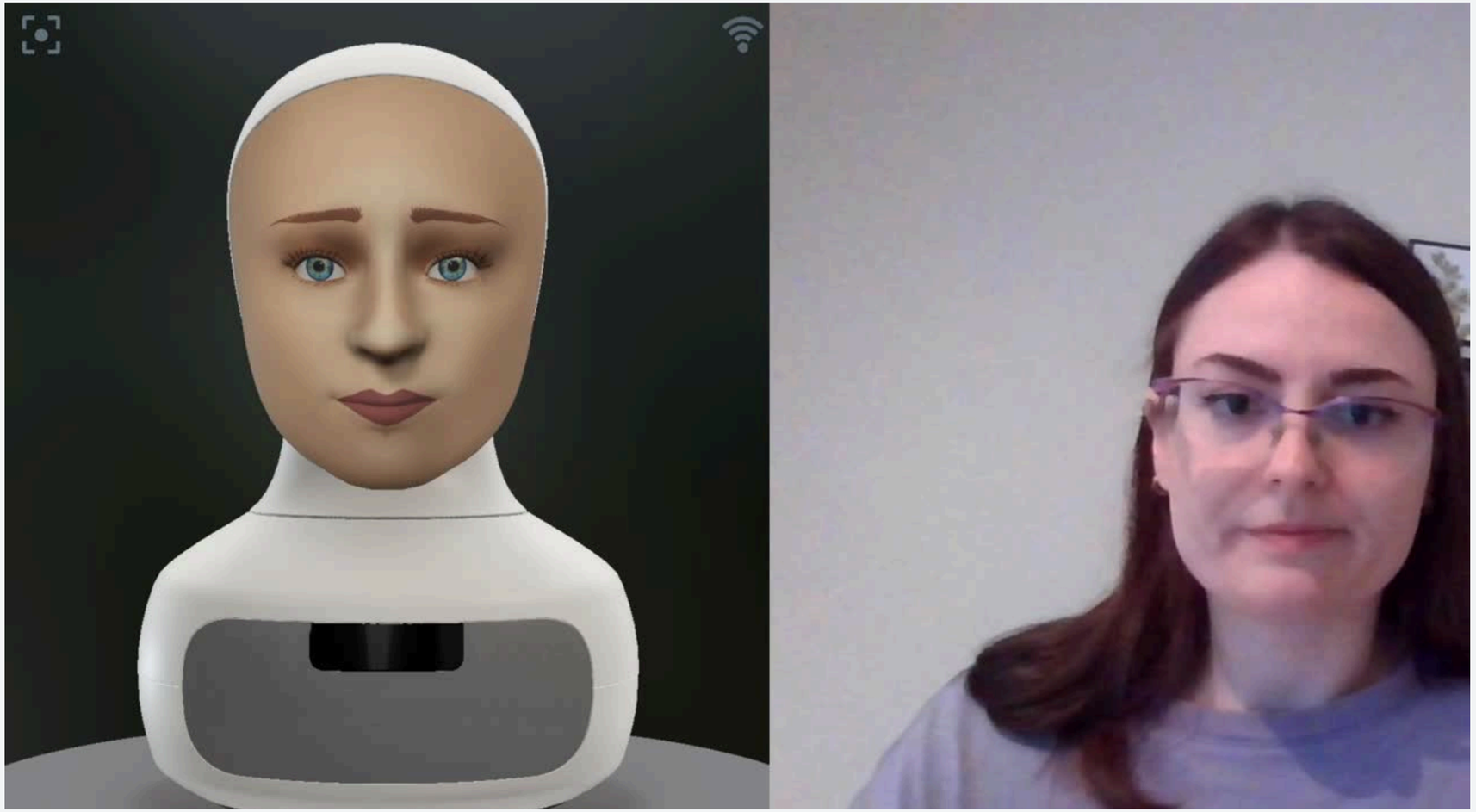
↓

TestLLM → **1**: guard "*userWantsToEnd*" [stopwords] → **Goodbye** (text, gesture, sound effect) → **End**

2: invoke *LLMActor* → **LLMSpeaks**

DEMO

Link to the demo (gold stardard)



CONCLUSION

- **The prompt with instructions for the LLM is key to this project...**
  - Pro:
    - Easy, fast and clean!

But...

CONCLUSION

- **The prompt with instructions for the LLM is key to this project...**
  - Cons:
    - It leans too friendly! → LLMs are socially optimized for politeness, so they gravitate toward “friendly” or “cheerful” tones unless explicitly told otherwise.
    - It seems that the prompt alone isn’t enough to elicit X emotions unless the user input itself frames the emotion explicitly.
      - Ex.: “Pretend you're an actor performing a high-tension movie scene. You’re really angry.”

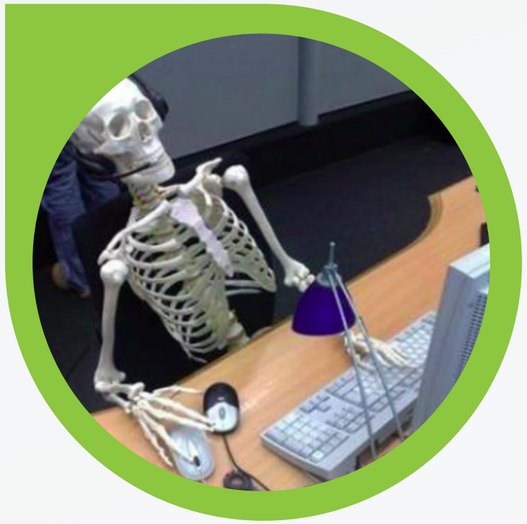
CONCLUSION

- **The prompt with instructions for the LLM is key to this project...**
  - Cons:
    - LLM's output format might not be consistent.

"**Alright, let's heat it up! We can start with the basics: the step-touch and the hip action.**"
{
"response": "**We can start with the basics: the step-touch and the hip action.**",
"voiceStyle": "friendly"
}

  - "Please remember to provide a JSON-styled format when giving a response!”

FUTURE WORK



- Manage to use voice styles:
  - Azure's own
  - SSML-styled tags
  - Creating a skill
- Improving prompting or exploring more reliable approaches.
- Improving gestures (more realistic?)
- Once a working implementation:
  - Next step → Transitioning from "interface" project to explorative project!

THANK YOU!