# FARMI: A FrAmework for Recording Multi-Modal Interactions

**Patrik Jonell**[1], **Mattias Bystedt**[1], **Per Fallgren**[1]
**Dimosthenis Kontogiorgos**[1], **José Lopes**[1], **Zofia Malisz**[1]
**Samuel Mascarenhas** [2], **Catharine Oertel**[1], **Eran Raveh**[3], **Todd Shore**[1]

[1] KTH Speech, Music and Hearing, Stockholm, Sweden
[2] GAIPS INESC-ID, Lisbon, Portugal
[3] Multimodal Computing and Interaction, Saarland University, Germany
pjjonell@kth.se

**Abstract**

In this paper we present (1) a processing architecture used to collect multi-modal sensor data, both for corpora collection and real-time processing, (2) an open-source implementation thereof and (3) a use-case where we deploy the architecture in a multi-party deception game, featuring six human players and one robot. The architecture is agnostic to the choice of hardware (e.g. microphones, cameras, etc.) and programming languages, although our implementation is mostly written in Python. In our use-case, different methods of capturing verbal and non-verbal cues from the participants were used. These were processed in real-time and used to inform the robot about the participants' deceptive behaviour. The framework is of particular interest for researchers who are interested in the collection of multi-party, richly recorded corpora and the design of conversational systems. Moreover for researchers who are interested in human-robot interaction the available modules offer the possibility to easily create both autonomous and wizard-of-Oz interactions.

**Keywords:** multisensory processing, human-robot interaction, multimodal interaction

## 1. Introduction

Recording group interactions is an essential step towards a better understanding of the mechanisms of multi-party interaction both between humans and between humans and robots. In such interactions, humans communicate via a number of different channels. Expressions in any modality, as well as the interactions between modalities and participants, need to be reliably captured to enable a proper analysis of how humans communicate in group settings. Capturing multi-modal and multi-party interaction is still a tremendous challenge in terms of hardware and software development. Recordings of this kind generate very large amounts of data. Ensuring synchronisation between sensor, audio and video streams is crucial for reliable analysis.

When it comes to recording multi-modal datasets or implementing multi-modal dialogue systems, it is difficult to find a framework that has been widely adopted by the community. Given that there are a lot of constraints imposed by the hardware, often the only option left is to develop a framework around that same hardware. In addition, such custom frameworks frequently lack detailed documentation. There were previous attempts to develop frameworks that support multi-modal dialogue systems, for instance (Bohus and Horvitz, 2009) and IrisTK (Skantze and Al Moubayed, 2012). IrisTK enables the recording of multi-modal data collections, however, data stream synchronisation is not fully solved. Mint.tools (Kousidis et al., 2013) tackle synchronisation problems and perform recordings in an annotation friendly manner, however this framework does not close the loop into a dialogue system and is not, to our knowledge, publicly available.

An artificial agent needs to have additional methods of perceiving human behaviour in order to be context aware in human-robot interaction (Turk, 2014). Apart from natural language understanding, several shallow input modalities such as eye gaze or body posture can be used to detect different patterns of human behaviour and reason towards generating appropriate agent behaviour and feedback. Open source frameworks for real-time recognition of social signals from many sensors exist (Wagner et al., 2013) also featuring machine learning and pattern recognition tools. Such modules can be included in dialogue system pipelines. (Thompson and Bohus, 2013; Bohus et al., 2017) provide a platform for analysis and development of multimodal interactive systems that is extensible and deployable for many situated intelligence tasks such as speech recognition leveraging multimodal signals, human-robot interaction supported with face recognition and so on. The platform is restricted however to sensors provided by one software company.

## 2. Goals

In this paper, we first describe the proposed architecture. By architecture we mean an abstract conceptualisation of how the components of the system interplay. Next, we present a framework that implements the aforementioned architecture.

Our goal is to tackle the problems encountered in previous attempts and present an open-source framework specifically developed to handle multi-party, multi-modal data recordings. The framework is modular, lending flexibility to different future applications. It has been designed to be modality agnostic, meaning that several modalities can be added or removed depending on the sensor data available and the perception models to be used. Both code and documentation are available for download [1].

[1] https://github.com/kth-social-robotics/multisensoryprocessing

As a test case for the framework, we present an application scenario involving the Furhat robot-head (Al Moubayed et al., 2012) as a participant in a role-playing game "Werewolf". In the scenario, we include several input modalities of verbal and non-verbal human cues that can lead to better understanding of the context and situational awareness of the conversation: speech, gaze and body posture and movement. In the case study, as we describe in Section 5., we start by collecting low level sensor data that we further aggregate in near-real time to higher level decision making models. We use such models in order to reason effectively how to generate appropriate agent behaviour. Note that in our implementation, we do not support e.g. incremental processing, such as the approach presented in (Kennington et al., 2017), but rather, after aggregating gathered data, we make a decision on what the current state or intent of the participant is.

## 3. Architecture

The framework implements a modular message-oriented architecture for sending, receiving and handling of sensory data from multiple sources in a distributed manner. Beyond the sensory hardware and software itself, the system is composed of fours layers: Sensor API modules, sensor data preprocessors, message processors and end consumers. Each layer is composed of individual modules which send and/ or receive information from other modules asynchronously via a message broker. Despite being published and consumed in an asynchronous manner, each message includes information from a central time server that enables messages published by different physical systems to be synced to within a negligible margin of error.

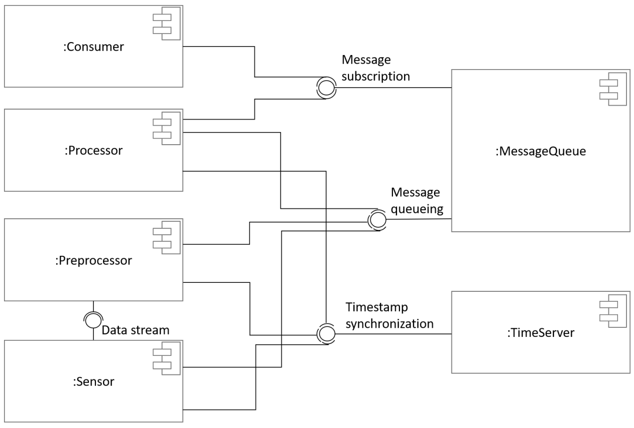

Figure 1: Interfaces between application layers.

**Sensor** modules interface with the sensory control software (e.g. for sound cards or gaze-tracking equipment), sending sensory data to any module which is listening on its relevant socket. Upon initialisation, each sensor module broadcasts information using the message broker which is required to connect to it and read socket data.

**Pre-processors** convert data streams from sensor modules into individual messages, which are submitted to the message queue for publishing to other modules. For instance, the speech recognition result in the case of the microphone sensors.

**Processors** process messages from other modules which they have been registered to receive via the message queue, e.g. performing dialogue act detection recognition on the data received from the speech recogniser or predicting visual attention from gaze data.

**Consumers** use data handled by the framework for domain-specific purposes, e.g. dialogue planning, language generation or action planning for robots or virtual agents.

The individual modules in each layer are only grouped conceptually; A given module defines the module(s) it should be listening to through the message queue, and thus is dependent only on the message queue itself and the time server to synchronise timestamps for messages as well as raw data streams (see Figure 1).

### 3.1. Data Synchronisation

Data streams and messages are synchronised by offsetting the timestamps using a delta function from a central time server, ensuring that timestamps for data from each module are comparable. When each module initialises, it requests a time from the time server; The difference of this time from the local time is then used to offset timestamps sent with data from that module. Downstream modules, which process data produced by other modules (i.e. all but sensory modules), include the offset timestamp sent by the previous module in the stream, thus allowing the original time of the original sensory input to be easily reconstructed.

When using a local area network for sensory data transfer, issues related to latency were negligible (see Section 5.). However, the system architecture described here is agnostic to the exact timestamp synchronisation algorithm used and thus a more sophisticated one can be implemented in the case that greater precision is required.

## 4. Framework

We implemented the aforementioned architecture into a framework as a distributed system. The framework is written in Python and gives the developer an easy way to define sensors, preprocessors, processors and other consumers and manages the communication and synchronisation of the data streams for the user. The framework's main responsibilities are: (1) managing the messaging and the data streams between the different components, (2) manage synchronisation between data streams, and logging and recording of the input signals. For the message queue Rabbit MQ was chosen, and for the data sockets between the components ZeroMQ was used. We present an example in Figure 2 for the case where only speech data is being recorded from a single microphone (sensor), processed with automatic speech recognition (ASR) and interpreted by the Natural Language Understanding (NLU) module.

## 4.1. Communication Between Components

There are two primary ways of communication between the components (see Figure 2): (1) messages through the message broker, and (2) direct data streams between processing units. The message broker is a centralised server which other processes can connect to, subscribe to topics and receive messages sent out on those topics by other components. The direct data streams are on the other hand a peer-to-peer connection between components. This allows for streaming data without straining the message broker with a flood of messages for each packet. The main limitation is that the receiver needs to know the IP address of the sender. This is handled by each sensor sending out a message through the broker with its own IP address and information on what kind of data it is broadcasting. The communication via the message broker is performed by publishing and listening for messages on given topics. A topic can, for example, look like *microphone.data.participant_A* and a listening process can subscribe to either exact topics or use wildcards in order to receive a broader range of messages, e.g. *microphone.data.** to receive microphone data for all participants.

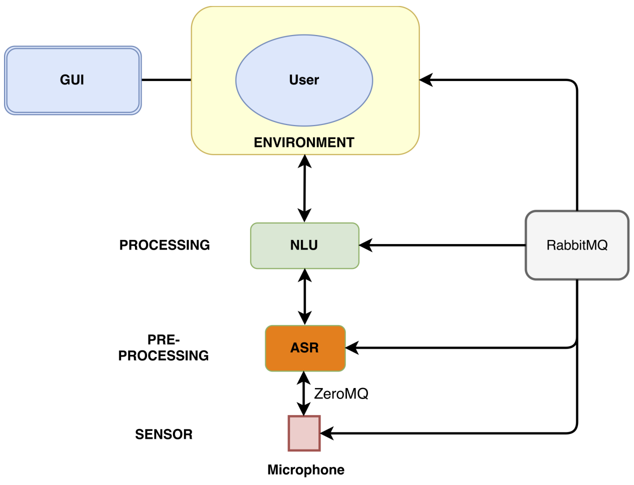

Figure 2: We use a layered structure of data processing. Using the framework we aggregate low-level sensor data to high level information on the human's state and intent from verbal and non-verbal cues. In this example we present a single-sensor usage of the framework that can update the environment in real-time with the incoming utterances.

## 4.2. Synchronisation of Data Streams

The framework handles synchronisation of data streams in a last in - first out fashion. We process sensor data and aggregate to have a common frame rate amongst the sensors that need to combine data streams. To ensure the data streams have occurred in the same time, each node that the data passes adds its own timestamp as meta-data to the pack. As such the time the data is captured on sensor level is kept on track to be able to handle the latency in the system for differently processed data. Pre-processors listen to messages from a sensor in which there is interest to receive data from, but it can listen to many sensors at the same time which creates the need for latency handling.

Using the same frame rate for combined sensors we can track what frames are expected to be processed an if there are any frames lost in the process. The timestamp of the original data packet sent from a sensor is kept throughout and is therefore are easily aligned. Our synchronisation strategy does not assume very timely latencies and will therefore aggregate data streams from combined sensors only when data from all sensors is available. For example if a pre-processor expects data streams from two types of sensors to aggregate and send to a processor, it will wait until all sensors have completed their data streams before it will pass the information to the next layer.

## 4.3. Data Transportation

The data streams are transported in local network for smaller latencies, however in our distributed architecture, several components can be part of the framework that are either in the cloud or in remote repositories. As described above the data packs are transported in two ways: through a message broker and through peer-to-peer connections. Each data pack transported across each layer is logged as a message exchanged between components with the timestamps attached.

## 4.4. Logging and Recording of Data

The framework contains a module that can record and log all of the data being sent between the components. This is done by subscribing to all topics and writing the data to file. It is possible to start multiple recording instances to decrease the load on an individual data recorder, as the quantity of data can become large with high-quality sensor equipment.
As the framework adds a timestamp to each data packet, it is also possible to visualise the flow of data and where potential bottlenecks exists in the system. However, this kind of visualisation is currently not implemented into the framework.
Logging the data as messages adds another advantage to post-processing the recordings. The messages can be replayed as in real-real time in order to reproduce data captured from specific sensors. This helps debugging the recording system, but also replicate human behaviour given specified sensor data.
In the next section we describe how we made use of the proposed architecture by implementing a framework and components that can capture sensor data in a use case scenario performing multi-party multi-modal spoken interaction between humans and a robot.

## 5. Use Case: The Werewolf Game

A good example for utilising the presented architecture is a multi-modal, multi-party group interaction. This scenario requires multitude of sensors to capture the entire dynamics of the group interaction. In this study, we used the this architecture for recording a multi-modal corpus of the "Werewolf" game. Werewolf is a multi-party role-playing social game. Each player's objective is to eliminate the opposing players by deceiving them regarding the player's identity – a citizen or a Werewolf. Other studies have used this

game before to study deceptive behaviour (Chittaranjan and Hung, 2010) and gaze patterns (Oertel and Salvi, 2013).
The game is suitable for studying deceptive behaviour. However, due to its engaging, multi-party nature, it is also suitable for studying multi-party turn-taking phenomena, and therefore provides the possibility to investigate other research directions as well. For instance, the role of eye-gaze coordination during turn-holds and turn-grabs in a multi-party setting, or relation between participant engagement aggregated eye-gaze patterns. The use of a robot in this use-case poses further possibilities for analysing the collected data. For example investigating how participants react to the robot as one of the players. Are they ignoring the robot or are they treating him as an equal entity? Would they eliminate it more often or would prefer to let it stay in the game? What do participants like about the robot's behaviour and where would they expect further improvements?

## 5.1. Experimental Setup

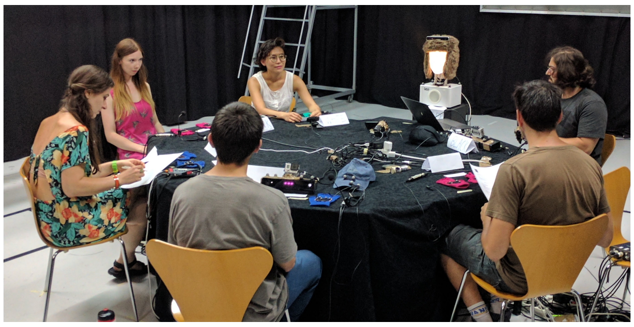

Figure 3: Experimental setup of multi-party, multi-modal interactions with a social robot in the "Werewolf" game.

The recording environment presented here is aimed for a Wizard of Oz experimental setup. Although it can be adjusted to function without a wizard, these efforts are beyond the scope of this work.

### 5.1.1. Multi-Sensory Data and Hardware Setup

During the game, the players sit around a table as shown in Figure 3, so that each player can create a direct eye contact with all the other players. In this setup, the moderator was not seen by the players, but heard through a loudspeaker. The setup was designed for a game with six human players and one robot player. The number of participant is only limited by the amount of sensors available, as the architecture can collect data for each sensor independently.
Each participant had a microphone for recording and recognising their speech, a Kinect v2[2] camera or Tobii glasses[3] for capturing eye gaze, gloves and a hat with motion capture reflective markers for capturing hand gestures and head pose, and a - camera for capturing facial expressions. Each of these hardware sensors had a equivalent software sensor (see Table 1) to process and send the raw data to the rest of the system. In addition, a video camera was positioned above the table to provide the wizard (who sat behind a curtain) a live feed of the session. This camera's data was not sent to the rest of the system, i.e. was not recorded or taken into account while playing the game. All of the raw data from the sensors were recorded and saved to an external storage in real time.
The eye-gaze information from the Kinect was translated into a fixation on a specific player, if applicable, in real time using GazeSense[4]. The equivalent information coming from data recorded by the Tobii glasses was recorded and after combined with motion capture, mapped to 3D space in a similar manner. Furthermore, any information that might be relevant for the robot's decision making was fed into the FAtiMA system (Dias et al., 2014), as described below. The system includes some hand-crafted theory of mind rules (Gardner, 2011), which are designed to detect deception.

## 5.2. Architecture Implementation

### 5.2.1. Pre-Processors

All of the sensors mentioned in 5.1.1. had a corresponding pre-processor, as specified by the architecture. The eye-tracking, motion capture and Kinect gaze pre-processors parsed the incoming data and sent them out as messages through the message broker. Players' spoken utterances were sent as raw data to IBM Watson's cloud-based automatic speech recognition (ASR)[5]. OpenFace (Baltrušaitis et al., 2016) was used for facial feature extraction for extracting action units (Ekman and Friesen, 1978).

### 5.2.2. Processors

In this use case, we mapped the ASR output onto the most likely dialogue act among *accuse*, *defend* or *support*, using a template file with common n-grams characterising these dialogue acts, as found in a Werewolf corpus (Oertel and Salvi, 2013). Since the ASR sends incremental data, we constantly updated this as soon as new ASR results are available. For the facial expression recogniser, we used action units obtained by OpenFace. The processor could collect several high-level facial cues that are valuable for the decision-making component. The cues (e.g. opened or closed eyes, frowning, smiling, etc.) could then, with a basis in literature on deception, be used as behavioural features for deciding whether a player is behaving deceptively or not.

### 5.2.3. Decision-Making

Given the high amount of sensory input and processor units that are present in the architecture, there is a need for a decision-making component that receives processed high-level information from the sensors and points to concrete actions. The cognitive framework Fearnot AffecTIve Mind Architecture (FAtiMA) (Dias and Paiva, 2005) was used to achieve that. In particular, the modular version presented in (Dias et al., 2014). FAtiMA is a flexible system designed to work in an array of different settings where a user wants to

[2]https://developer.microsoft.com/windows/kinect
[3]https://www.tobiipro.com/
[4]https://eyeware.tech/
[5]https://www.ibm.com/watson/services/speech-to-text/

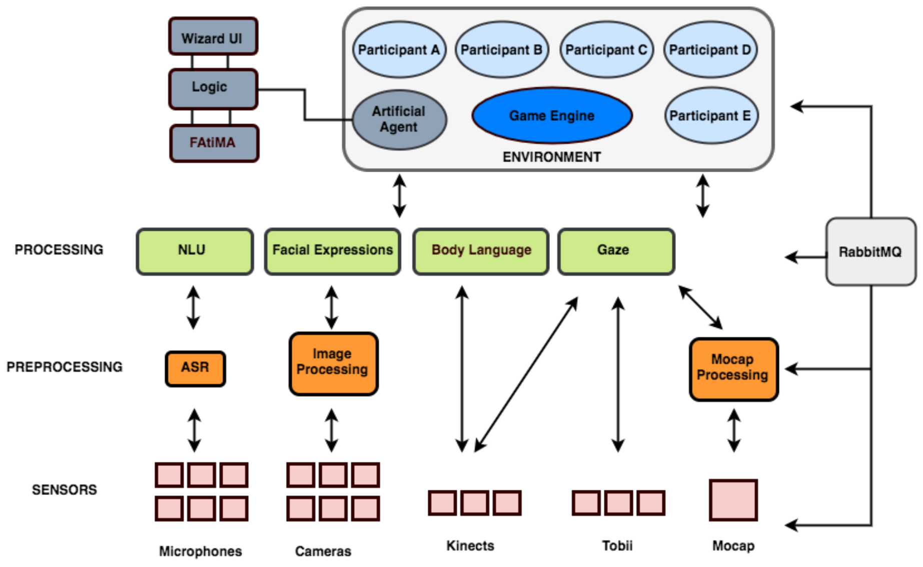

Figure 4: The multi-sensory processing architecture used in the use case study. We have separated the processing of information into three data processing layers: sensor, preprocessor and processor. Each participant has a current state initiated in the processing environment given the aggregated data captured by the previous layers.

create an agent with some degree of higher-level information processing without the need of implementing a system from scratch.

In our system, FAtiMA is based on two components: A dynamic belief system which is updated consecutively, and a fixed set of rules.

In the setting of the Werewolf game, a simple rule could, for instance, be that Furhat should accuse a player of lying if they accused another participant that according to the belief system has behaved truthfully in the previous round. Unless stated otherwise, the belief system will not initially have any information regarding the truthfulness of participant. This information has to be explicitly given to the framework from collected data via the pipeline of processed sensory input. By implementing a series of rules and updating the belief system accordingly through the processes of the framework, a more sophisticated decision-making system emerges. This framework can be used to facilitate the decisions of a wizard, which was the case in the Werewolf scenario. However, the framework does also provide the grounds necessary to represent a fully autonomous agent instead of the wizard.

#### 5.2.4. Wizard Interface

In the current setting, the wizard has access to a few actions via a computer keyboard. These actions correspond to the three dialogue act categories identified in the data collected in (Oertel and Salvi, 2013): *accuse*, *defend* and *support*. For each of these acts, the template file randomly picks an utterance from a pre-defined set. Besides these acts, a few other common dialogue acts and "small talk" utterances are available. Once the game reached the voting round, the wizard uses the an interface to vote in the name of the robot player. While voting, the likelihood of each participant being the werewolf as calculated by the decision-making system is displayed in the wizard interface.

| | |
|---|---|
| Sensors | Microphones<br>Kinect<br>USB cameras<br>Motion capture (Vicon)<br>Eye-trackers (Tobii) |
| Pre-processors | Automatic speech recognition<br>Eye-tracking data parser<br>Kinect gaze data parser<br>Motion capture data parser<br>Facial feature extraction |
| Processors | Dialogue act recogniser<br>Facial expression recogniser |

Table 1: The processing layers and their corresponding processing units that were implemented for the Werewolf game.

## 6. Discussion

The current paper described an architecture for recording multi-modal interactions. Its purpose was threefold. First, it proposed a solution to handle data-stream synchronisation. Second, it was designed to be easily extendable and third and final it was created to be freely available to everyone dealing with multi-modal systems.

The framework was applied to the ?werewolf" use-case scenario. This scenario proved to be quite challenging. Six players were interacting in real-time with a robot. We were tracking voice activity and gaze (respectively head direction) simultaneously for all participants. Moreover, we converted the sensor information into visualisations that were displayed in the wizard interface. We encountered several challenges in this use-case. For instance, we found that some sensors required the complete use of a computer, blocking the use of other sensors, at a given time. This resulted in a very complex system using multiple computers.

The choice for a distributed system architecture seemed appropriate in this case, and its behaviour proved to be robust. However, there is certainly room for further improvements to the framework. For example, currently the framework still requires to start and maintain processes on each individual machine. In addition, system debugging is currently not trivial as messages are broadcasted to the whole system. It is often difficult to track in the code possible bugs. Improved logging features and extended documentation about all of the components is needed and is currently being developed. Despite the fact that there surely are some more challenges to be solved, depending on use case scenario, we believe that the framework we are releasing can be very useful to the community.

## 7. Conclusion and Future Work

In the current paper, we presented a generalisable architecture for multi-modal, multi-party recordings. We illustrated its usability by relating it to research questions of the "Werewolf" use-case scenario. Future work consists of creating new processing units and automating the process of installing and running multiple sensors on multiple computers. The authors started implementing the architecture in other projects. For example, a sensor unit exists in place for data input using an OptiTrack[6] motion capture system.

## 8. References

Al Moubayed, S., Beskow, J., Skantze, G., and Granström, B. (2012). Furhat: A back-projected human-like robot head for multiparty human-machine interaction. In Anna Esposito, et al., editors, *Cognitive Behavioural Systems*, pages 114–130, Berlin and Heidelberg, Germany. Springer Berlin Heidelberg.

Baltrušaitis, T., Robinson, P., and Morency, L.-P. (2016). Openface: an open source facial behavior analysis toolkit. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*, pages 1–10. IEEE.

Bohus, D. and Horvitz, E. (2009). Dialog in the open world: platform and applications. In *Proceedings of the 2009 international conference on Multimodal interfaces*, pages 31–38. ACM.

Bohus, D., Andrist, S., and Jalobeanu, M. (2017). Rapid development of multimodal interactive systems: a demonstration of platform for situated intelligence. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, pages 493–494. ACM.

Chittaranjan, G. and Hung, H. (2010). Are you a werewolf? detecting deceptive roles and outcomes in a conversational role-playing game. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 5334–5337. IEEE.

Dias, J. and Paiva, A. (2005). Feeling and reasoning: A computational model for emotional characters. In *EPIA*, volume 3808, pages 127–140. Springer.

Dias, J., Mascarenhas, S., and Paiva, A. (2014). Fatima modular: Towards an agent architecture with a generic appraisal framework. In *Emotion Modeling*, pages 44–56. Springer.

Ekman, P. and Friesen, W. V. (1978). *Facial action coding system: A technique for the measurement of facial movement*. Consulting Psychologists Press, Palo Alto, CA, USA.

Gardner, H. (2011). *Frames of mind: The theory of multiple intelligences*. Basic books.

Kennington, C., Han, T., and Schlangen, D. (2017). Temporal alignment using the incremental unit framework. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, pages 297–301. ACM.

Kousidis, S., Pfeiffer, T., and Schlangen, D. (2013). Mint. tools: Tools and adaptors supporting acquisition, annotation and analysis of multimodal corpora. *Proceedings of Interspeech 2013*.

Oertel, C. and Salvi, G. (2013). A gaze-based method for relating group involvement to individual engagement in multimodal multiparty dialogue. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 99–106. ACM.

Skantze, G. and Al Moubayed, S. (2012). IrisTK: a statechart-based toolkit for multi-party face-to-face interaction. In *Proceedings of the 14th ACM international conference on Multimodal interaction*, pages 69–76.

Thompson, A. L. and Bohus, D. (2013). A framework for multimodal data collection, visualization, annotation and learning. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 67–68. ACM.

Turk, M. (2014). Multimodal interaction: A review. *Pattern Recognition Letters*, 36:189–195.

Wagner, J., Lingenfelser, F., Baur, T., Damian, I., Kistler, F., and André, E. (2013). The social signal interpretation (ssi) framework: multimodal signal processing and recognition in real-time. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 831–834. ACM.

[6]`http://optitrack.com`